****

**Probos**

Team 14 (Team Mayhem) - Sprint 1 Retrospective

*Alexa Baker, Daniel Raftery, Steven Daniel Vaughan, and Michael Woods*

**What went well?**

Overall, we laid the groundwork necessary for the future sprints as well as covered most of the core functions necessary for a Mastodon client. We covered logging in and out of an instance, OAuth of the device the client is on, viewing timelines, messages, favoriting, boosting, posting messages, replying, viewing profiles, follow counts, profile pictures and banners, and lists of users. Going forward, the second and third sprints will allow us to build a more fully realized and featured solution to set us apart from other Mastodon clients. We also continually committed to at least two meetings a week and met up with each to ensure no gaps in communication hurt the project.

**User Story 1**

As a user, I want to log in to my account(s).

Tasks Required:

|  |  |  |  |
| --- | --- | --- | --- |
| Order: | Task: | Estimated Time: | Assigned To: |
| 1 | Build UI for the login page | 2 hours | Steven |
| 2 | Connect data fields and buttons to functions/API calls | 2 hours | Steven |
| 3 | Test API calls to the server using the login page | 2 hours | Steven |
| 4 | Implement saved credentials for less frequent login requirements | 2 hours | Alexa |

Completed:

* The user is directed to the instance’s login site once the “Get Authentication Code” button is pressed. After they authenticate on the site, the user is prompted to copy their authentication to the app, and the code is used for an access token. The access token is saved using a “SharedPreferences” object.

**User Story 2**

As a user, I want to view “toot” messages sent by others.

Tasks Required:

|  |  |  |  |
| --- | --- | --- | --- |
| Order: | Task: | Estimated Time: | Assigned To: |
| 1 | Build UI for Timelines | 2 hours | Steven |
| 2 | Retrieve messages via API call(s) for display within the UI | 2 hours | Daniel |
| 3 | Implement a method of refreshing the message list | 2 hours | Daniel |
| 4 | Test Timeline UI functionality and API calls | 2 hours | Michael |
| 5 | Build UI for the expanded view of a “toot” | 2 hours | Steven |

Completed:

* After login, another activity (TimelineActivity) is started and the user is directed to the Timeline page. From here, messages are displayed on their respective timeline’s tab. Scrolling up when already at the top of the list allows the user to refresh the current list of messages. The user can also press on the “More” button to be brought to the message’s expanded view.

**User Story 3**

As a user, I want to send “toots.”

Tasks Required:

|  |  |  |  |
| --- | --- | --- | --- |
| Order: | Task: | Estimated Time: | Assigned To: |
| 1 | Build UI for drafting “toots” | 2 hours | Daniel |
| 2 | Connect data fields and timestamp to functions/API calls | 2 hours | Daniel |
| 3 | Test API calls to the server using draft page | 2 hours | Alexa |
| 4 | Include refresh on message list to include new toot | 2 hours | Daniel |

Completed:

* We created a UI for drafting new “toots” and integrated the new screen with two access points: one in a floating action button on the home timeline screen, which allows new general statuses to be drafted and posted, and one that is accessed by tapping the reply button on a status, where the draft will automatically load the ID of the status and post it specifically as a reply. New statuses and the freshly sent replies can be seen on both Probos and web after performing a refresh of the timeline.

**User Story 4**

As a user, I want to be able to log out of or switch the account I am using.

Tasks Required:

|  |  |  |  |
| --- | --- | --- | --- |
| Order: | Task: | Estimated Time: | Assigned To: |
| 1 | Add button in menu UI for logout and return to login UI | 1 hour | Steven |
| 2 | Connect button to API call and remove saved credentials | 1 hour | Alexa |
| 3 | Test API calls to the server using profile page and check that login credentials are removed | 1 hour | Alexa |

Completed:

* After saved credentials were created using SharedPreferences, the ability to log out was then implemented relatively easily via the addition of a “Logout” button that, when clicked, uses a SharedPreferences function to delete saved credentials. The app is then returned to the instance selection and login activity.

**User Story 5**

As a user, I want to be able to follow other users.

Tasks Required:

|  |  |  |  |
| --- | --- | --- | --- |
| Order: | Task: | Estimated Time: | Assigned To: |
| 1 | Build user profile UI | 4 hours | Steven |
| 2 | Connect UI to API calls for both functions and data retrieval | 3 hours | Alexa |

Completed:

* The profile UI was built and connected to the API to contain the display name, user name, profile image, user bio, number of followers, and number of users the account is following. This UI is shown when a profile image is tapped on any screen that contains one, and it contains a back button that the user can press to return to the previous activity.

**User Story 6**

As a user, I want to see my list of followers.

Tasks Required:

|  |  |  |  |
| --- | --- | --- | --- |
| Order: | Task: | Estimated Time: | Assigned To: |
| 1 | Create a UI similar to a timeline to display users in a list you can scroll through (around 50 users initially displayed) | 2 hours | Steven |
| 2 | Connect UI to API calls to retrieve the following list | 2 hours | Michael |
| 3 | Add option to sort the list of users (default, alphabetical ascending/descending) | 2 hours | Michael |
| 4 | Test API calls and sorting options using the following page | 2 hours | Michael |

Completed:

* Upon tapping a user profile’s number of followers, the list is populated with the first 50 profiles returned by the instance (or the entire list, whichever has less members). This same activity can also be updated and used for accounts a profile is following, users who favorited a message, or users who boosted a message, if time allows in one of the next sprints.

**User Story 7**

As a user, I want to scroll to view older “toots”.

Tasks Required:

|  |  |  |  |
| --- | --- | --- | --- |
| Order: | Task: | Estimated Time: | Assigned To: |
| 1 | Ensure the timeline UI is able to adjust to accommodate additional messages. (50 at a time) | 3 hours | Steven |
| 2 | Create a function to retrieve additional messages via API and expand the timeline downwards | 2 hours | Michael |
| 3 | Test that additional messages maintain correct order when appended to the end of the list | 2 hours | Michael |

Completed:

* When the user nears the bottom of the currently-loaded list of messages, the app automatically queries the instance to retrieve the next set of messages up to but not including the oldest one already in memory. This was accomplished by making a set of modifications to the function that loaded messages into the timeline RecyclerView instead of creating a separate function. This feature will be updated to be more smooth and applied to user lists in the future as well.

**User Story 8**

As a user, I want to favorite “toots.”

Tasks Required:

|  |  |  |  |
| --- | --- | --- | --- |
| Order: | Task: | Estimated Time: | Assigned To: |
| 1 | Ensure a button to favorite a “toot” is available in both timeline view and expanded view messages. | 2 hours | Daniel |
| 2 | Connect the UI button to an API call to mark the message as favorited (or unfavorited). | 2 hours | Alexa |
| 3 | Test with web client that messages are favorited (or unfavorited) appropriately. | 1 hour | Alexa |

Completed:

* We added a favorite button with two states that toggle when tapped, letting the user know if a status is favorited or not. When loading new statuses, the client calls the API to see if any given status being loaded is already favorited by the user, and loads the state of the button appropriately. When the button is tapped, the updated favoriting status is sent through the API to update the instance, rather than just showing a “false favorite” on the app.

**User Story 9**

As a user, I want to reply to specific “toots.”

Tasks Required:

|  |  |  |  |
| --- | --- | --- | --- |
| Order: | Task: | Estimated Time: | Assigned To: |
| 1 | Ensure a button to reply to a specific “toot” is available in both timeline view and expanded view messages that opens message composition. | 2 hours | Daniel |
| 2 | Connect the UI button to API calls to get conversation identifiers to maintain conversation threading. | 2 hours | Daniel |
| 3 | Implement a modified toot draft functionality/screen into the reply message field. | 2 hours | Daniel |
| 4 | Test button and toot functionally in different scenarios (timeline view, conversation view, etc) | 1 hour | Michael |

Completed:

* Reply button was implemented in the same sequence of buttons at the bottom of the statuses containing the favorite and boost buttons. It currently uses a placeholder symbol which (if time allows) will be replaced with an improved version in sprint 3. The reply button sends the user to the same page as the draft screen, but includes data about the ID of the status they are replying to, and once the “toot” is sent, the draft is marked as a reply through the API rather than being posted to the timeline and can be viewed from the original status to which it is responding.

**User Story 10**

As a user, I want to boost “toots.”

Tasks Required:

|  |  |  |  |
| --- | --- | --- | --- |
| Order: | Task: | Estimated Time: | Assigned To: |
| 1 | Ensure that there is a button available in both timeline view and the expanded view to boost (or unboost) a given “toot.” | 2 hours | Daniel |
| 2 | Connect the UI button to an API call to mark the message as boosted (or unboosted). | 2 hours | Alexa |
| 3 | Test with web client that messages are boosted (or unboosted) appropriately. | 1 hour | Alexa |

Completed:

* On the timeline activity, each message (“toot”) has a button below that the user can press to boost the message, which is reflected across all Mastodon clients. The button also updates to reflect whether or not the message has been boosted by the user.

**User Story 12**

As a user, I want to change which timeline I am looking at (personal, local, or federated).

Tasks Required:

|  |  |  |  |
| --- | --- | --- | --- |
| Order: | Task: | Estimated Time: | Assigned To: |
| 1 | Ensure the main UI includes three tabs, one for each of the timelines. | 2 hours | Michael |
| 2 | Maintain separate message lists and timeline positions for each tab. | 3 hours | Alexa |
| 3 | Update API calls for each tab to ensure they refresh with the appropriate messages. | 1 hour | Michael |

Completed:

* After login, on the timeline activity, there are three tabs listed near the action bar: Personal, Local, and Federated. Each tab allows the user to view its respective timeline. The tabs can be viewed by pressing on their names or by swiping sideways from within the tabs’ views.

**What did not go well?**

**In general,** the idea of some people doing primary UI and design and others doing API calls and testing did not work out well when we realized it was easier for the person making the UI to also connect it to the API and test from their own branch before pushing possibly buggy code to the master so someone else could test. There was also multiple problems with Android Studio at first being buggy and flat out not working until members uninstalled and reinstalled the application multiple times causing programing in the first week to start very late. Finally the last big problem we ran into was that, while we had a decent API to use, the documentation of said API is almost nonexistent, leading us to have to use trial and error while combing through a ton of the API to find the functions to use for each new call.

**User Story 5**

As a user, I want to be able to follow other users.

Tasks Required:

|  |  |  |  |
| --- | --- | --- | --- |
| Order: | Task: | Estimated Time: | Assigned To: |
| 3 | Test that profile information loads correctly and that following (or unfollowing) works correctly | 2 hours | Alexa |

Not Completed:

* While the follow button and some basic code was added to complete this task, testing was not finished and the current build of the code crashes when any profile page is opened.

**User Story 6**

As a user, I want to see my list of followers.

Tasks Required:

|  |  |  |  |
| --- | --- | --- | --- |
| Order: | Task: | Estimated Time: | Assigned To: |
| 3 | Add option to sort the list of users (default, alphabetical ascending/descending) | 2 hours | Michael |

Not Completed:

* This feature was cut due to time restraints and a lack of support through the API. As far as we have found, it is not possible to request a sorted list of users, so while in theory it would be possible to sort it in the client, to do so appropriately would require retrieving the entire user list first, or sort only in the 50-user increments the app currently loads.

**User Story 11**

As a user, I want to attach media attached to a “toot.”

Tasks Required

|  |  |  |  |
| --- | --- | --- | --- |
| Order: | Task: | Estimated Time: | Assigned To: |
| 1 | Add a button to the message draft UI to attach media to a “toot.” | 1 hour | Michael |
| 2 | Attach a system call to the UI button to open the media picker. | 3 hours | Michael |
| 3 | Ensure that API calls are updated (if need be) to account for the attachment. | 2 hours | Michael |
| 4 | Test messages appear correctly locally and non-locally | 2 hours | Michael |

Not Completed:

* This task was pushed back multiple times due to prioritizing the display of timeline messages and creating basic messages, since they are both needed for this function to work correctly. This task also has the added difficulty of checks and conversions to make sure the media is in an acceptable format for mastodon.

**How should you improve?**

We have planned several changes for the second sprint in order to streamline our production process and end up with a better result. First and foremost, we are moving to a more vertically integrated development plan, where specific team members will be in charge of features from top to bottom, which will vastly improve our workflow as we will no longer be forced to wait for another team member to develop something before we can do our tasks. This different style will allow us to work at our own pace, more efficiently, and deliver features we can be more confident and familiar with. We are also planning on writing our testing document earlier on in the process in order to clarify our goal when developing a new feature. For example, we know we need feature X to do Y and W instead of just Y.

While we are relatively consistent in our ability to meet up for work sessions, we sometimes struggle with focusing on a given point during such meetings, especially when other courses have similar deadlines to coursework, such as CS252. If possible, we should attempt to have other responsibilities taken care of before sprint meetings, or at least wait until after our properly scheduled meeting time has passed before working on those instead. If there is an inability to arrive at the meeting at the scheduled time, there should be a plan to disseminate the required information, or a new meeting should be scheduled for optimal communication and understanding among members of the team.

During the last sprint, another thing that our team could have improved upon was documenting our progress, which so far has only consisted of simple git commit messages and weekly reports. It might be helpful to maintain some sort of continuous update of the backlog for everyone to view during a given time in a sprint, so we know how we are doing in regards to time and progress.

We can also be sure to account for research time needed to implement a given feature in order to try and avoid a heavier workload as we approach the end of the sprint, as this was a big issue leading to several late nights in the final week of this past sprint.