Probos



CS30700

Design Document

Team Mayhem (14)

Alexa Baker

Daniel Raftery

Steven Daniel Vaughan

Michael Woods

Index

* Purpose **3**
  + Functional Requirements **4**
  + Non-Functional Requirements **8**
* Design Outline **9**
  + High-Level Overview **10**
  + Sequence Of Events Examples **12**
* Design Issues
  + Functional Issues **15**
  + Non-Functional Issues **17**
* Design Details
  + Class Design **19**
  + UI Mockup **24**

Purpose

The new social media platform Mastodon has many appealing aspects to it, with its inherently decentralized model being one of the most attractive characteristics. Anyone who wishes to host their own “instance” of the service is able to do so while still maintaining the ability for full communication with other such instances, allowing for users to choose what corner of the federated universe, or “fediverse,” they wish to reside in. It is also attractive to some users because it means that no one person or corporation controls the entire system and its information, such as with Facebook and its data scandal last year.

However, this same decentralization has led to a fracturing of effort in mobile app development. A number of different clients exist for Android, such as Tusky, Mastalab, and Subway Tooter, but many of them lack a handful of the features that Mastodon provides, such as the “Delete & Redraft” option or the ability to create and modify Lists. Our project aims to be an improved client with higher functionality and usability for Android platforms, including various customization options, notification settings, and other features in an intuitive user-friendly interface.

Functional Requirements

1. **Accounts**

As a user,

* 1. I want to register an account on a specific instance.
  2. I want to log in to my account(s).
  3. I want to be able to log out of or switch the account I am using.
  4. I want to be able to go through my instance’s process for resetting passwords.

1. **Messaging in the Timeline / “Toots”**

As a user,

* 1. I want to view “toot” messages sent by others.
  2. I want to scroll to view older “toots”.
  3. I want to view a list of the “toots” I have favorited.
  4. I want to favorite “toots”
  5. I want to reply to specific “toots”
  6. I want to boost “toots”.

1. **Creating a Message/ “Toot”**As a user,
   1. I want to send “toots.
   2. I want to adjust the privacy setting of a “toot” (public, unlisted, followers only, direct message) on a message-by-message basis.
   3. I want to show or hide media attached to a “toot.”
   4. I want to attach a content warning to “toots” as needed.
   5. I want to be able to delete and redraft my previous messages.
   6. I want to save message drafts.
2. **Instance Settings**

As a user,

* 1. I want to change which timeline I am looking at (personal, local, or federated).
  2. I want to carry over the list of profiles I follow to a new account on a different instance.
  3. I want to edit my profile icon and banner images in a specific instance.
  4. I want to carry over my picture and banner to a new account on a different instance.
  5. I want to edit my account “bio” information for a specific instance.
  6. I want to access custom emoji from my instance. (if time allows)

1. **Public Profile**

As a user,

* 1. I want to be able to follow other users.
  2. I want to see my list of followers.
  3. I want to change my display name.
  4. I want to be able to pin any of my “toots” to the top of my profile.
  5. I want to mute/block other users.
  6. I want to change my notification settings.

1. **Feed / “List” Customization**

As a user,

* 1. I want to create new “Lists”.
  2. I want to edit and delete “Lists”.
  3. I want to be able to only view my “Lists”.
  4. I want to enable media only mode for the local and federated timelines. (if time allows)
  5. I want to change my filter for words/phrases. (if time allows)

1. **Notification Customization**

As a user,

* 1. I want to change the frequency push notifications are generated.
  2. I want to choose what types of notifications I receive (favorites, boosts, mentions, new followers).

1. **Client Appearance**

As a user,

* 1. I want to customize the primary and secondary color schemes of the app’s user interface. (if time allows)
  2. I want the app to react properly to different font sizes. (if time allows)

1. **Server/Instance Control**

As a server admin,

* 1. I want to be able to ban accounts. (if time allows)
  2. I want to modify the instance’s default primary and secondary color schemes. (if time allows)
  3. I want to be able to read user-submitted reports. (if time allows)

1. **Security**

As a developer,

* 1. I want to implement a safe method of passing over login credentials to the instance without risking the personal information of users.
  2. I want to avoid storing raw personal login information.

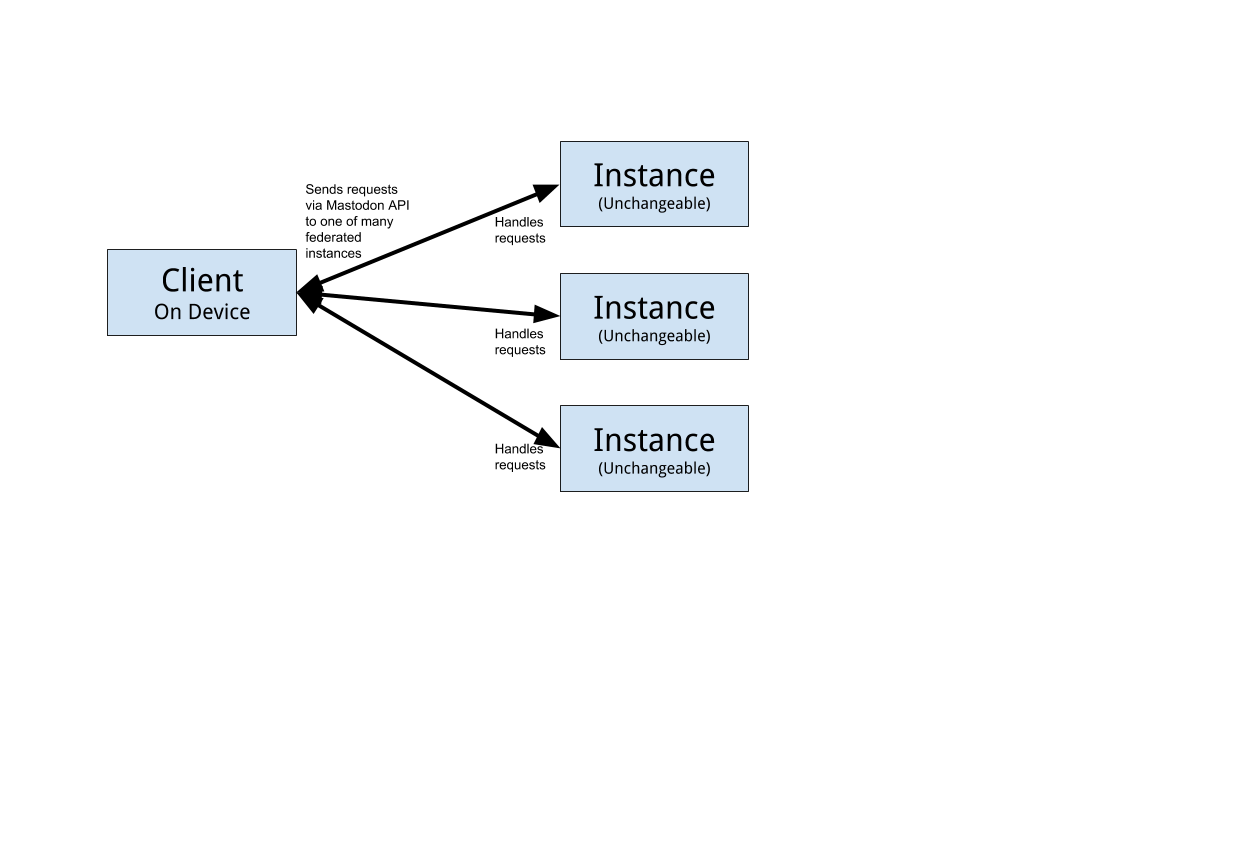
Non-Functional Requirements

1. **Architecture and Design**
   1. The front end will be developed in Android Studio.
   2. Java API for Mastodon will be used for server-client communication.
   3. Multiple versions of Android will be supported.
   4. Application size will be less than 35 MB.
   5. Cache size of media, user lists, etc. will be less than 100 MB (if time allows)
   6. The client should load the instance with low latency as a priority of usability.
2. **Usability**
   1. The interface will be intuitive, sleek, and easy to navigate.
   2. Swipes and other gesture-related controls for touch screens will be implemented.
   3. The interface will adapt to multiple screen sizes. (if time allows)
   4. Portrait and landscape screen orientations will be supported. (if time allows)
3. **Security**
   1. We will build user trust and confidence by storing information in a secure manner and only if needed.
   2. Local storage backup of account information will be opt-in. (if time allows)
   3. We will limit/completely avoid handling of sensitive information by any third party or external service. (save Mastodon)

Design Outline

High Level Overview

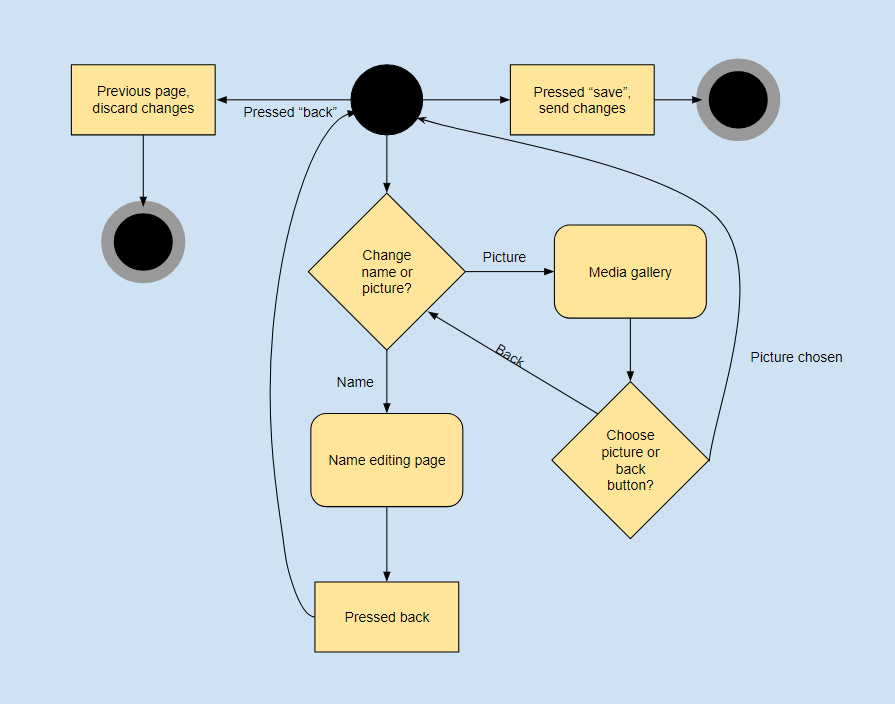
We will be building our project in Android Studio, utilizing the Mastodon API's Java implementation to facilitate communications. Our application fits into the existing client-server structure of the service, in which one of many federated servers handles accesses and requests from multiple active clients to provide service to its users.



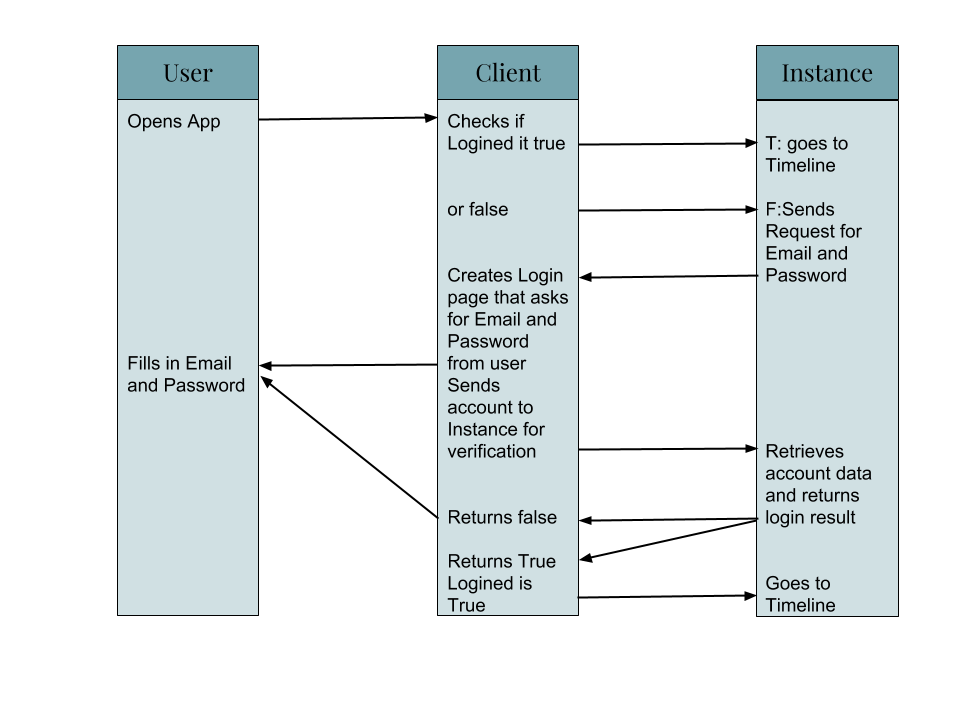
**Design Overview:**

* The client provides a front end interface to the user for use with the existing Mastodon service servers.
* The client sends API requests to the user’s chosen Mastodon instance to be processed, such as posting a new status or requesting to view a user profile.
* The client receives and interprets data returned from the server and updates information where needed.

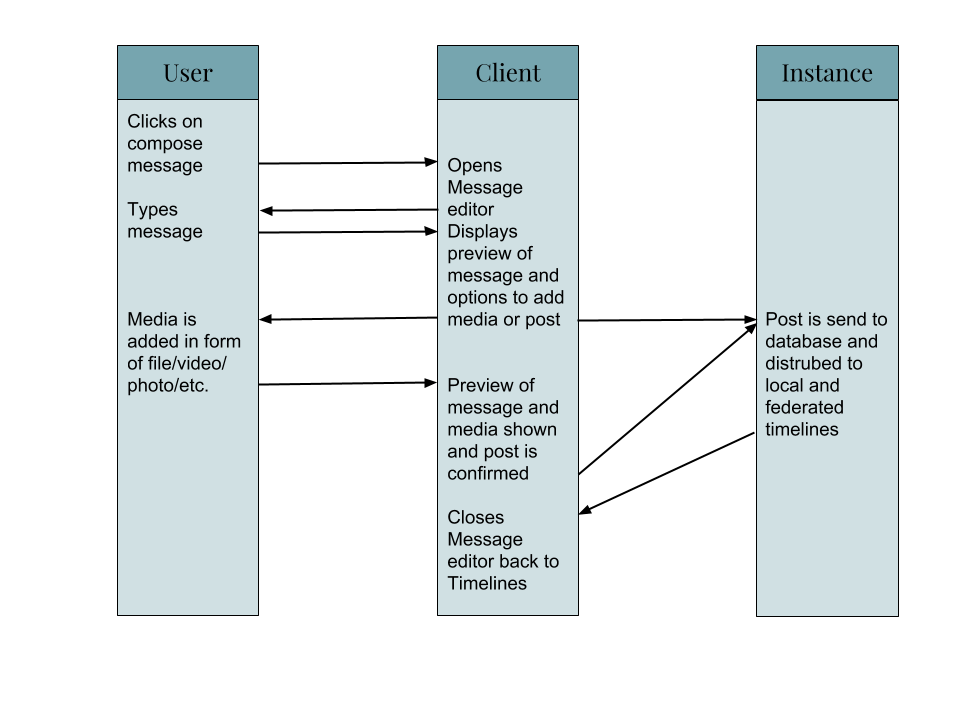
**Example Activity Flow**

****

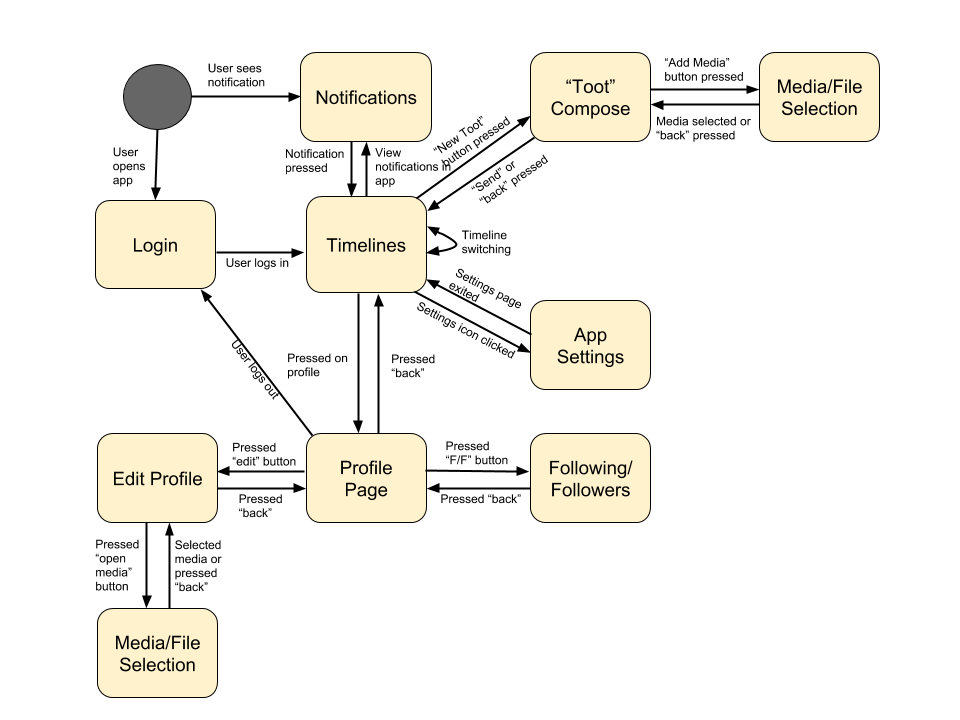
**Sequence Diagram: Login**



**Sequence Diagram: Sending Messages**



**State Diagram**



Design Issues

**Functional Issues**

1. **What information should be required to create an account?**
   1. Option 1: Instance, Username, Password.
   2. Option 2: Instance, Username, Password, E-Mail.
   3. Option 3: Instance, Username, Password, E-Mail, Phone Number.

**Our Choice:** Instance, Username, Password, E-Mail. Reason: Since the Mastodon Server is not accessible to us, we must follow its standard user creation rules and guidelines of what is needed for an account.

1. **Message order**
   1. Option 1: newest at bottom of the screen.
   2. Option 2: newest at top of the screen.

**Our Choice:**  Newest at the top - The Mastodon web client UI displays messages in reverse chronological order, the same as Twitter, with the newest messages appearing at the top of the timeline, so sticking with this format will ease transition both for users transitioning from Twitter and current Mastodon web UI users.

1. **Controls**
   1. Option 1: Swipes only (not logical).
   2. Option 2: Taps only.
   3. Option 3: Mixture.

**Our Choice:** Mixture - While tapping can effectively do everything needed, the ease of access by adding swipes to a few features will make the app feel smoother and make certain aspects easier to get to.

1. **What can be transferred to new accounts/instances?**
   1. Option 1: lists of users followed, blocked, and muted
   2. Option 2: list of users followed, blocked, and muted, profile picture, profile banner

**Our Choice:** Option 2 - We want to allow people to change instances as easily as possible, and to aid this we want to let them bring as many aspects of their last account to the new one as possible. At the very least, we will allow the standard backup feature of the following list and blocked/muted users, and (if time allows) also bring their profile and banner pictures that may be hard to re-apply otherwise.

1. **Notifications**
   1. Option 1: messages that mention you, when a message is boosted or favorited, when followed/requested to follow.
   2. Option 2: messages that mention you, when a message is boosted, and when someone requests to follow.
   3. Option 3: only messages that mention you, when someone requests to follow.

**Our Choice:** Option 1 - This is the current default for the Mastodon web UI, but we will allow for options to customize which notifications are received. Implementing the change in notifications keeps people from getting too many or too few notifications for their liking, and allows them to tailor the experience to their liking.

**Non-Functional Issues**

**Security**

1. **What language should we use?**
   1. Option A: Flask
   2. Option B: Python
   3. Option C: Java
   4. Option D: C++

**Our Decision:**  Java - We are using Android Studio to help maintain a balance of familiarity, as we are comfortable working with Android Studio and Java, and useability as Java and Android Studio can help us easily reach the majority of users without exceeding our skill cap.

1. **What front end language should we use?**
   1. Android Studio
   2. Node.Js
   3. Javascript and HTML

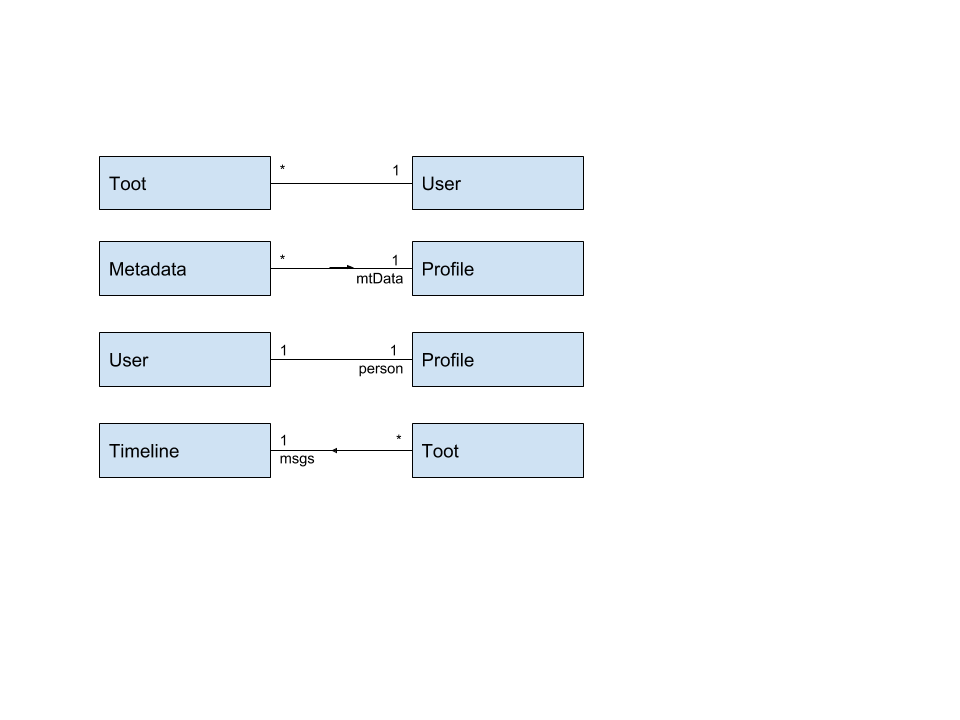
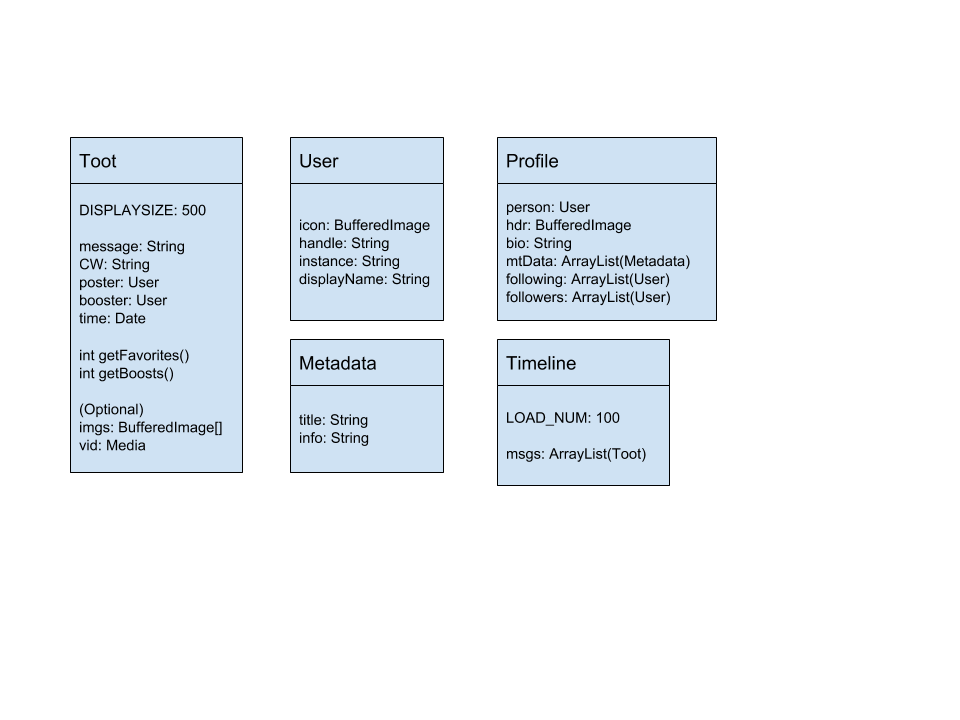
**Our Decision:** Android Studio that interrupts HTML commands- Android Studio has a built-in GUI maker that will allow us to create an easy to use and smooth looking front end but because of how the Mastodon server handles message formats we will have to interrupt some HTML commands such as <p> and <br> and convert them to the correct format

1. **How will we interact with the Mastodon services?**

**Our Decision:** Since we are creating a Java-based application, we decided to use a Mastodon API that supports Java, because most or all of our communication with Servers and Instances can be made through use of the API for clients, and we will support connecting to new or other existing instances, and pull information from them using the API. The server/instance will send items through for the client to receive and/or display, such as toots, media, profile usernames, favorites, etc.

Design Details

**Class Design**



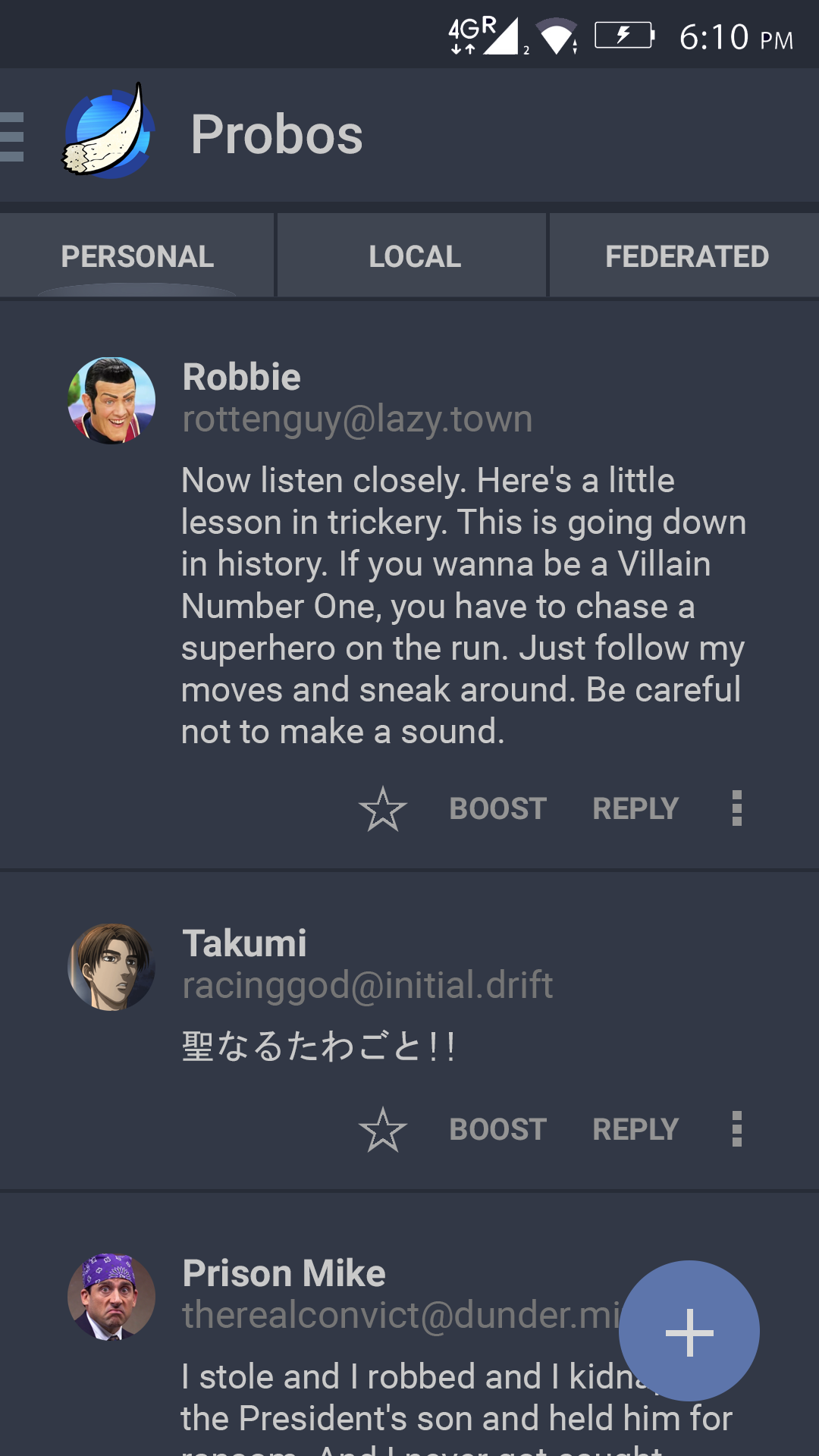
**Descriptions of Classes and their Interactions**

Each class in our program is designed to encompass data commonly received from the instance to facilitate easier operation of the application.

* **Toot**
  + The default maximum message length for an instance is 500 characters, but it is not uncommon for an instance to permit users to enter longer text. Should a received message exceed this length, the substring containing the first 500 characters can be displayed with an ellipses and Expand button to reveal the full length of text.
  + Each Toot contains the basic information about who posted it, who boosted it (if applicable), when it was posted, and of course the message itself. The number of favorites and boosts is retrieved at the time the message is loaded into the client, and refreshed any time the timeline is.
* **User**
  + The User class contains the basic data about a user’s profile, such as their display name, username, instance, and icon. Many locations require only this small amount of data contained within the User class, and multiple of this object type can be stored due to its small size.
  + Timelines specifically benefit much from this class, as multiple Toots they contain can all rely on a single User object to save load times.
  + Profile objects are able to use this class to hold the core data of the user, as there is likely already an object of it for them, and in the case there is not, there will be need for one to be created anyways due to the user profile page’s Timeline, containing only messages they have boosted or written themselves.
* **Profile**
  + Profile objects contain additional data that is only needed specifically for display of the user’s profile page, such as their header image, bio information, and any profile Metadata fields the user may have filled out. This data can then be parsed into the user display window appropriately.
  + The Profile class also contains two User ArrayList objects to store which accounts they follow and are followed by, which can be parsed into a separate list window should the user tap on their respective numbers.
* **Timeline**
  + The Timeline class contains a Toot ArrayList to hold all messages to display on a given page. It is a slight misnomer to call it a true “timeline,” however, as the class actually represents the data structure for the true timeline component of a page and not the page itself. For example, in addition to the personal, local, and federated timelines, it will also be used for viewing Lists and user profiles.
  + The Timeline object should update appropriately when the page using it is refreshed in order to bring the user the most recent updates.
* **Metadata**
  + The Metadata class is a small one, but it is meant to contain information the user has signified as important for their profile. This could be anything from image credits for their profile photo or header to their age to what university they attend or graduated from.
  + While editing their profile, the user is able to specify any number of metadata fields for information, which is why the Profile class uses an ArrayList to maintain a profile’s metadata to display.

**Design Mockups**

* Timeline Example



* Profile Example

