Frame For Android

CS 307 Software Engineering I
Team 1 Design Document

Grant Jochums
Raaghavendar Karthikeyan
Daniel Sokoler
Craig Wilhite
Adam Worthington

Table of Contents

1.Purpose

Anonymously share a moment with your local community through videos and pictures. The local community will be based on the individual's GPS location, so the posts they see and interact with will be gathered based upon a local proximity. The user may then interact with this content through upvoting, downvoting, flagging and commenting.

This design document thoroughly describes our functional requirements:

1. Data is stored in a database.
2. The front-end interface will list the items in a listed order.
3. Users can choose to post between photo mode, video mode and text mode.
4. Users can scroll down to see more in that respective mode.
5. Users can choose the sorting between most recent and top rated.
6. Users can search for content with specific tags.
7. Users can flag content that they deem inappropriate.
8. Users can comment on posts.
9. Users can either upvote or downvote posts.
10. Users can look at posts from other locations.
11. The graphical user interface will be user friendly.

Scalability:

We have opted to use Google's app engine as the main framework for our backend/database. This decision came from our desire to have an easily scalable backend that can handle almost any load and has proven capable of doing so in other applications including Snapchat. This service will allow for reliable servers with the ability to dynamically change how much space/processing power is needed on the fly with no input from us as well as easy integration across multiple platforms.

Accessibility\Usability:

Our application is going to extremely accessible to almost anyone with an android phone. Once the app is downloaded, all a user would have to do is launch the application and dive in. There will be no linking with social media or creating an account. Once the application is open, a simple and intuitive user interface will allow the user to easily view, filter, and search content as well as participate in their community by posting images and videos.

Performance:

Our use of a professional database and backend server will help with the performance but it is not all we are doing. We also plan to implement several optimizations to keep the

application responsive. We are going to push most of the computationally heavy work, such as ordering and cataloging posts, on to the server side of things as well as limiting calls to the server by chunking up the posts so that each time the client goes to the server, it pulls several posts at a time to bring to the client.

Design Pattern Choice:

We chose to use the Model-View-ViewModel (MVVM) design paradigm to make use of data binding within our application to better facilitate the separation of the view layer from the underlying code, removing all GUI code from the view itself and into the viewmodel.

2. Design Outline

Frame for Android utilizes a client-server architecture. The clients shall upload content in the form of images and videos to a central server that runs a database. Clients request from the server content that is geographically local to them, after which a content feed in the application is loaded and may then be interacted with. The client side application shall be built using the MVVM (Model-View-ViewModel) architectural pattern. The server shall communicate via JSON and will implement the REST API as an interface.

1. Client
    a. The client handles all user interactions.
    b. Allows for filtering of displayed content that the user sees.
    c. Handles the actions of users taking pictures and videos, which are then sent to the server to be uploaded into the database.
    d. Allows for the user to interact with displayed content (through commenting, liking, disliking, and flagging content).
2. Server
    a. Handles all traffic and requests coming from clients and queries the database.
    b. Sends the results of queries of database back to client in form of content.
3. Database
    a. Stores all content that is submitted by clients.
    b. Content that shall be stored in the database are pictures and short videos.

Relationships Between Core Components

A variable number of clients will connect to our servers whose work is offloaded to Google App Engine. Clients that request content or push new content to others send requests to these servers. These servers proceeds to either query the database for new content or push the new content to be stored in the database.

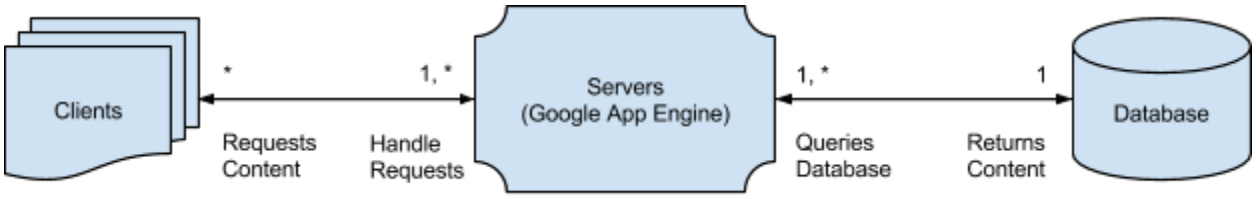

Figure 2.1: Illustration of the relationships between Frame's core components

High Level Interactions of Core Components

The figure below shows a couple of core interactions between our three primary components. Below shows the interactions taken to push new content to the database, to query the database for new content, and the save user interactions to the database.

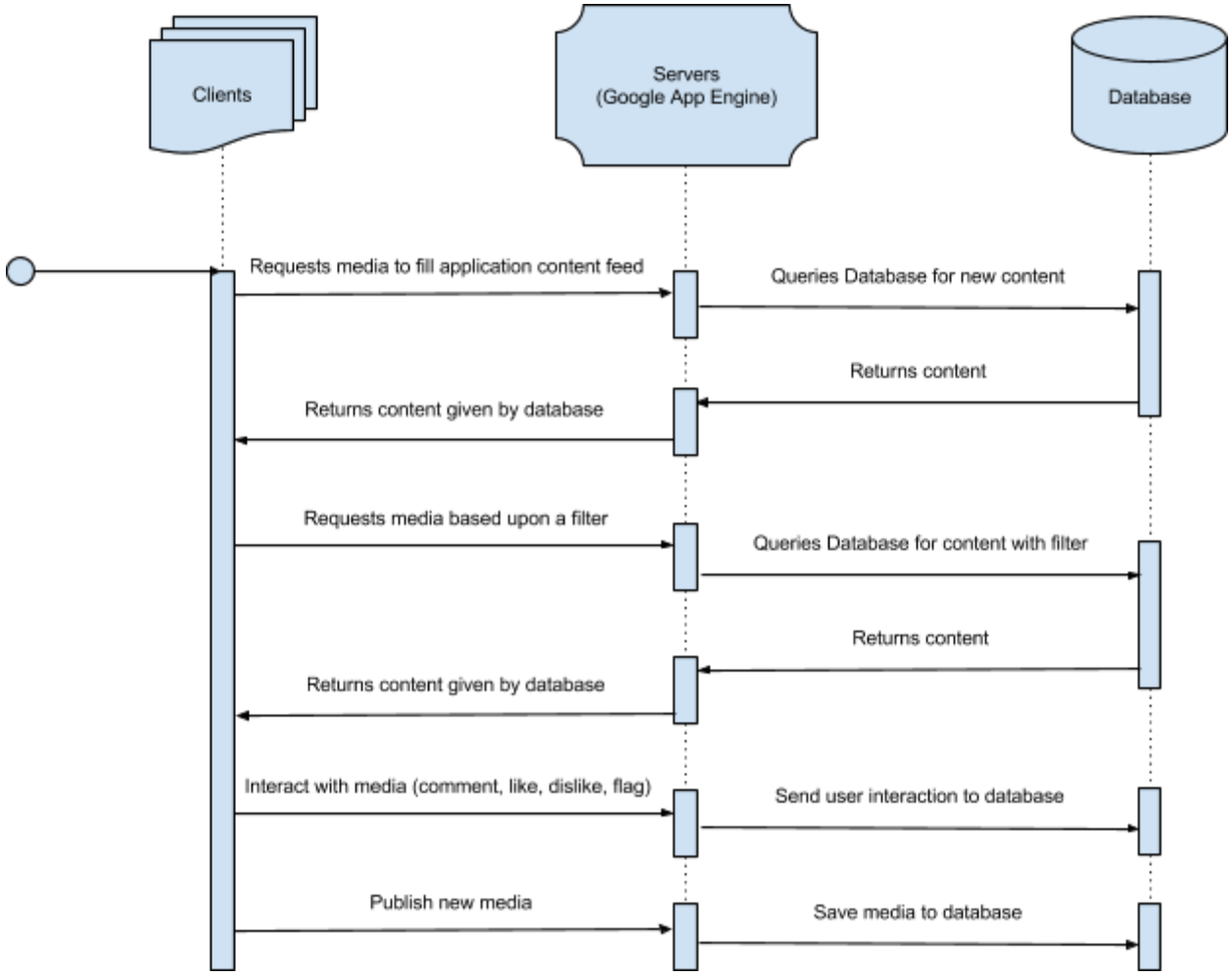

Figure 2.2: Illustration of the core interactions between our components.

Use Case Diagrams

Loading the App and Filtering

The diagram below shows the steps our system will take to when the application is launched and when the user chooses to filter their main feed by something like most voted or newest.

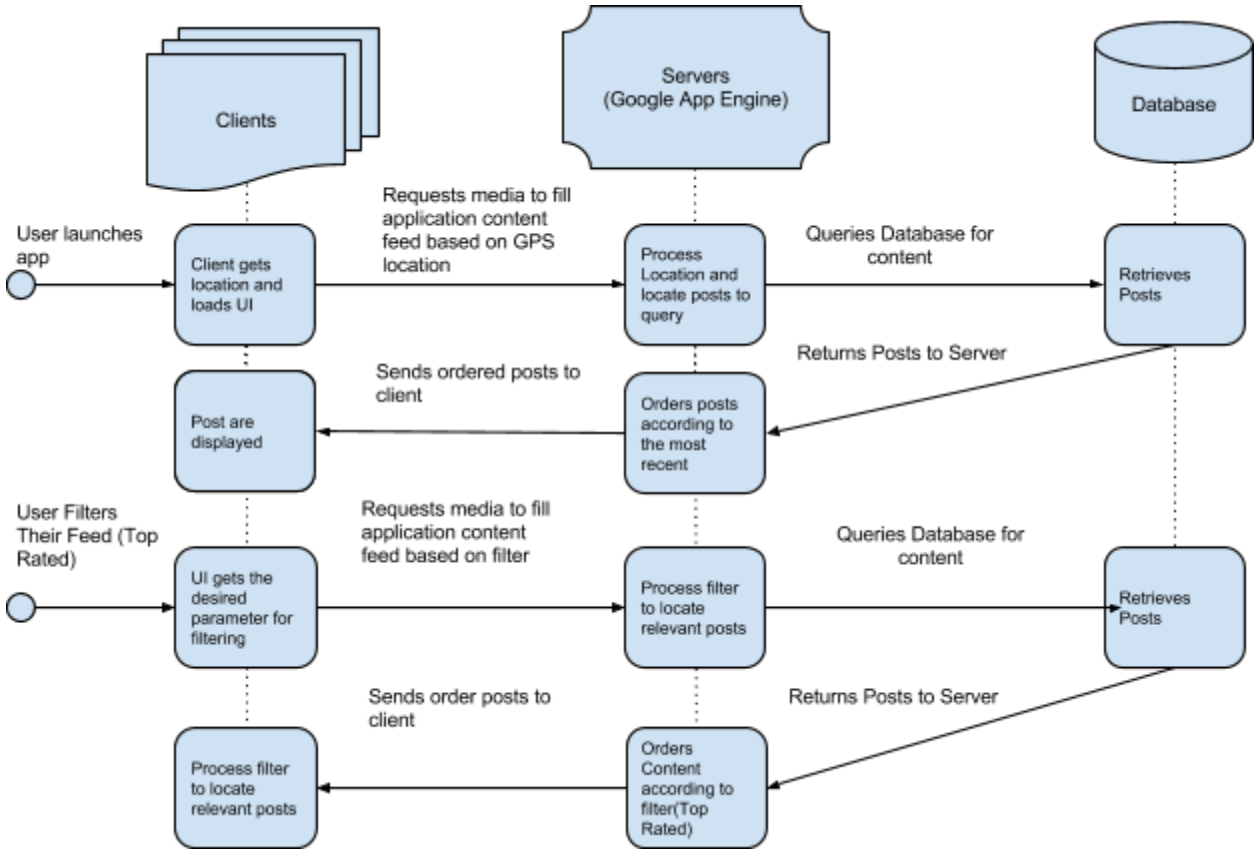

Figure 2.3: Illustration of loading the app and filtering content.

Continuous loading and text to image

As the user scrolls through posts, it is ideal that new content is loaded as the older content is viewed. The below diagram shows the steps taken in completing that task. Also shown is the steps taken when a user chooses to create text to be converted to an image and uploaded.

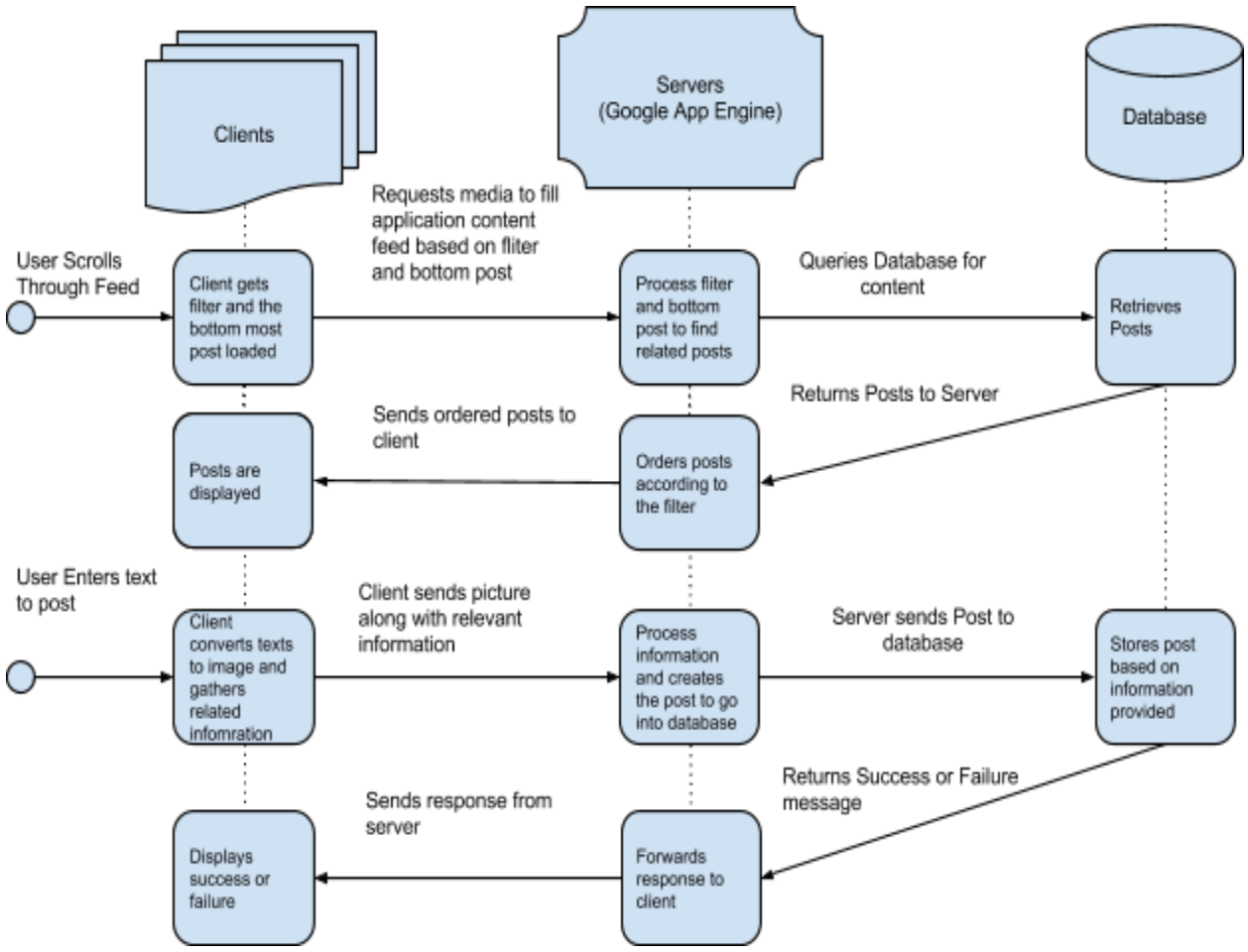

Figure 2.4: Illustration of continuous loading and text to image.

Submitting media

The user will first indicate the he or she wants to share content by opening the media capturing compartment of the app. From here they have a choice to take a picture or video to share with the local community. After review the picture is send to the server to be stored in an appropriate database. The server pings back with a success or failure with respect to uploading the media.

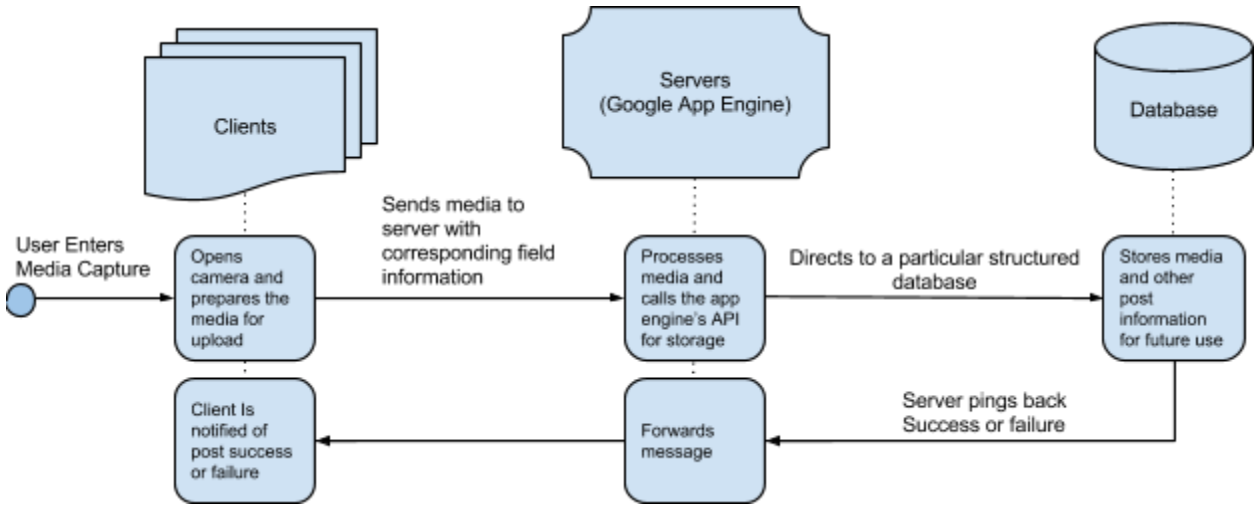

Figure 2.5: Illustration of submitting media.

Searching for Posts Based On a Tag

Allowing users to search for media posts based on a tag they input is one of the features of our app. The user inputs a "tag" into the search bar of the application, and the server/database retrieves posts that have the specified tag associated with them.

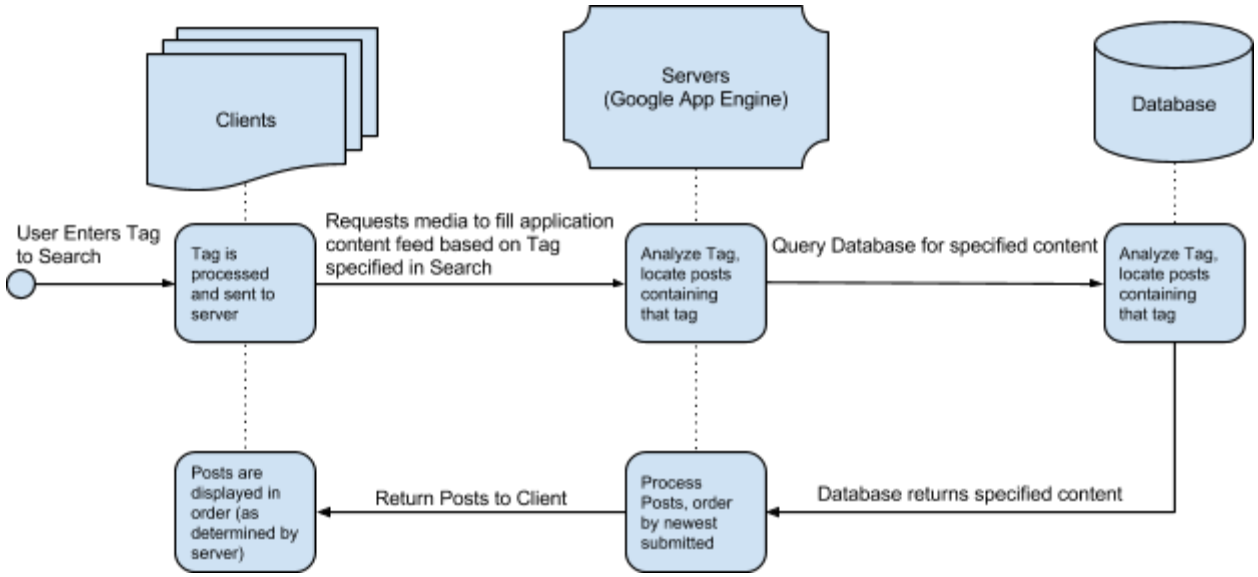

Figure 2.6: Illustration for searching for posts based on a tag.

Searching for Posts from a Different Location

Allowing users to search for media posts from a different location is one of the features of this application. The user inputs a location into the search bar or uses the provided map to drop a pin in a specific location. The server/database then retrieves posts from that specific location to be displayed.

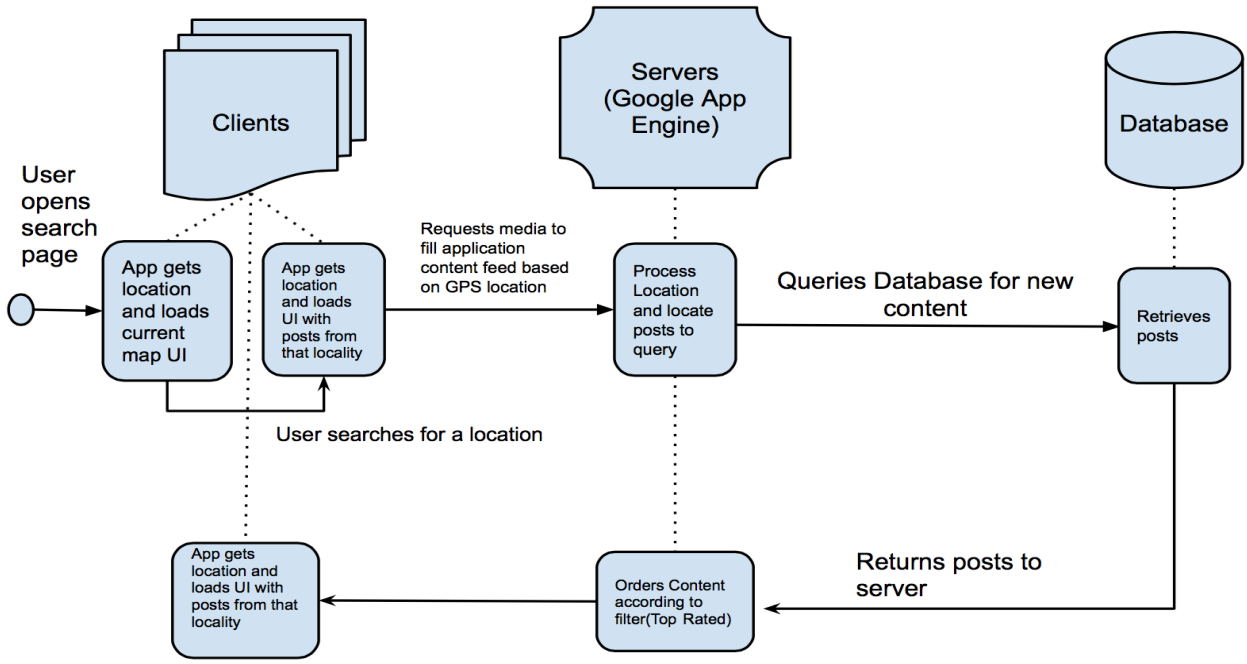

Figure 2.6: Illustration of searching for posts from a different location

3. Design Issues

Functional:

Issue #1: What constitutes the removal of a post?

- *Option 1*: A dynamic threshold of down votes by users
- *Option 2*: A dynamic threshold of reports by users
- *Option 3*: Physical moderation of posts by admins

Decision: A combination of *Option 1* and *Option 2* would yield the best results. A post that has many reports shows that the community viewing it thinks that it is inappropriate for the context it is in and should be removed. A post with a significant number of down votes shows that the community generally dislikes the post, but that it isn't inappropriate. A combination of these options would allow the automatic moderation and removal of posts the community has decided it does not want to see. *Option 3* is not a good decision, the time and cost that would be required to physically moderate all posts would be immense.

Issue #2: How can administrators communicate effectively to all users when needed?

- *Option 1*: Push a notification to the user's phone
- *Option 2*: Place an "admin" post that is always at the top of the user's feed
- *Option 3*: Have a pop up window when the user opens the app with the message

Decision: *Option 2* is the best choice for this scenario. Placing an admin post that always resides at the top of all users' feeds still allows for a streamlined viewing experience while getting the administrator's message across to all users. *Option 1*, while a viable option, has a tendency to get annoying to users, and places the message outside of its own environment. *Option 3* detracts from a streamlined viewing experience and therefore is not a good decision.

Issue #3: What should the max length of vidoes be limited to?

- *Option 1*: Ten seconds
- *Option 2*: Thirty seconds
- *Option 3*: No Limit

Decision: *Option 1* is the best choice for this app. Videos are large files, and the storage of a large number of large files consumes a vast amount of space. Limiting the length (and consequently the size) of the videos uploaded will greatly save on the storage required to keep the necessary videos.

Issue #4: Should users be allowed to view other "communities" that they currently are not in?

- *Option 1*: Yes
- *Option 2*: No

Decision: *Option 1* is the best selection for this app. Allowing users to view posts from another community (but not post to that other community) while their location doesn't currently identify them as a part of that community ensures that users can still access content that may be important to them even if they move outside their home community. For example, a student

travels home for the weekend, but still wants to view posts from the college he/she attends. Asking for a general location (town, college, store, etc) and then using a map API to identify GPS coordinates for this location.

Issue #5: What information should be stored in the database with the media itself?

- *Option 1*: GPS location of where it was posted from
- *Option 2*: Device ID of the user who posted it
- *Option 3*: The time the media was posted
- *Option 4*: Upvote/Downvote counter
- *Option 5*: Pointer to the media's comments
- *Option 6*: Tags (optional)

Decision: All of these options are good ones to be included. *Option 1* allows the filtering of posts based on a user's location. *Option 2* would allow the tracking of stats for that user (upvotes, downvotes, reports, and previous posts). Users would only be allowed to see their own statistics, as per the anonymity requirement of this app. *Option 3* would allow the filtering of old content that might not be relevant to the community anymore. *Option 4* is an integer that keeps track of the number of upvotes and downvotes. Upvotes increase the counter by 1 and downvotes decrease the counter by 1. *Option 5* is used to easy access to the comments on the media. It allows easy storage of comments in their own section as opposed to storing them in the same section as their respective media. See the "Media" class design for more information. *Option 6* would allow for users to search for posts they are interested in viewing. When users upload media, they have the option to associate their media with tags, enabling other users to search for said media.

Issue #6: How much media should be initially displayed to the user?

- *Option 1*: All relevant media for that user's community
- *Option 2*: One piece of media at a time
- *Option 3*: A scalable number of posts depending on how many fit on the user's screen
- *Option 4*: A fixed number of posts organized into pages

Decision: *Option 3* fits the idea of a social media app best. Breaking the media up into chunks of 10 or 15 pictures/videos allows the user to browse relevant content while not having to wait for great lengths of time for the content to load (this would be the main problem with *Option 1* and *Option 4*). When the user has viewed all the content, the app will query the database for more content for the user to view. *Option 2*, while a viable strategy for a media sharing app, would not allow for as streamlined of a viewing experience as *Option 3*.

Non Functional:

Issue #7: What constitutes a "community", in terms of the user?

- *Option 1*: A section of a map (towns/counties).
- *Option 2*: A certain radius around the user

Decision: Option 2 is the better decision here. Allowing the user to see all posts within a certain radius of him/her greatly improves the likelihood that the posts shown will be ones that

the user wants to see, ones from the correct “community”. *Option 1*, while technically a viable option, would struggle with users that are at the edges of these sections of the map. If a user happened to be on the very edge, he/she might be getting posts from a good distance away, while not getting posts from someone across the street. The user’s position would be determined by their GPS coordinates at the time their device requests posts.

Issue #8: Should a custom database be designed, or an already existing package used?

- *Option 1*: Design a custom database
- *Option 2*: Use a preexisting database framework

Decision: *Option 2* is the best choice here. *Option 1* would be very time intensive, requiring both learning database development and database creation. Though it would allow for a great deal of customization, pre-existing database frameworks will get the job done well enough, and the large decrease in time required to implement them vastly outweighs the customization options that come with a database created specifically for this project. Google’s App Engine fits the requirements for this project very well, allowing easy storage of media files, prioritization of important messages (admin notices), as well as alleviating the workload through easy access and use of Google’s other APIs.

Issue #9: Should banning users be a possibility?

- *Option 1*: Yes
- *Option 2*: No

Decision: Option 1 is the best decision in this scenario. If a user has too many posts that are downvoted and/or reported to the point where they are removed, that user is not contributing to his/her community in a positive way, and should be disallowed from using the app. This measure should only be taken in extreme circumstances, as banning users, while not permanent, almost ensures they will not be using the app again.

4. Design Details

Class Design

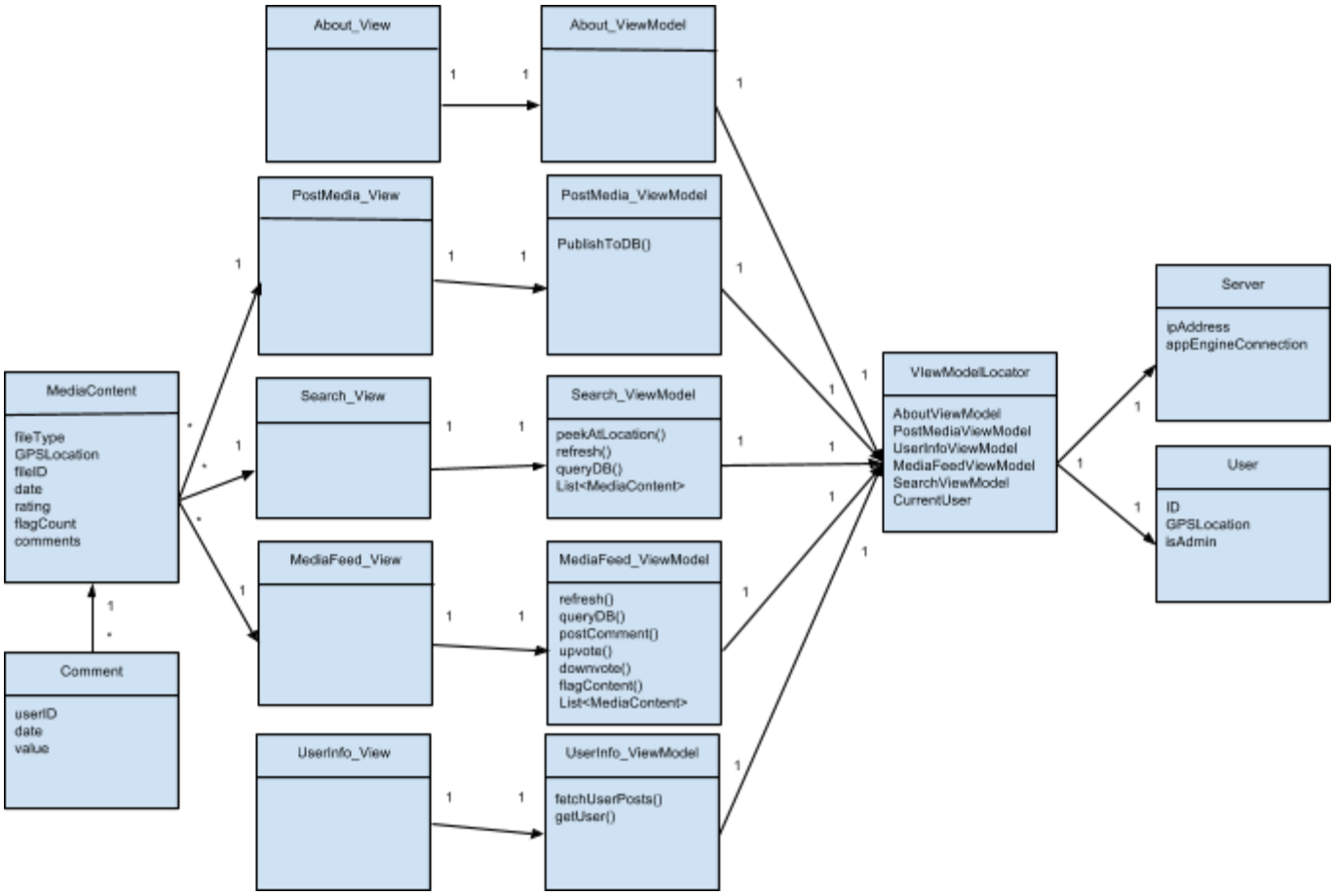

Figure 4.1: The class design and entity relationships between the classes

Class Definitions

User:

- Encompasses the user's basic information needed for the functionality of the app
- The GPS Location shall be used as the basis in determining the locality of posts from other users
- The isAdmin fields allows for the removal of content within the application for users who have a value of true

MediaContent:

- Stores the actual media content within this class
- Has appropriate tags associated with it for identification (date and poster)
- Has a reference to the associated comments with this media
- Rating and flagCount respectively keep track of the like-count of the post and how many times users have flagged the content as inappropriate

Comment:

- Contains the string value of the comment
- Has a reference to the user that posted it
- Has a reference to the date that the comment was posted

Server:

- Contains a reference to the ipAddress of the server
- Contains a reference to the appEngineConnection (because we are using Google App Engine to maintain server work)

ViewModelLocator:

- Contains static references to all view models within the application
- Contains a reference to the user of the application
- Contains a reference to the server to communicate with.

About_View:

- Contains information pertaining to the application concerning the version and how to contact the developers
- Links the user to the app store to rate the application.

PostMedia_View:

- The page that the user sees when taking and posting media to the application.
- This page will invoke the OS camera APIs.

Search_View:

- The page that allows the user to peek at other locations that the user does not belong to.
- Upon choosing a location to peek, this page will have a content feed that is loaded with content local to the chosen location

MediaFeed_View

- The main page of the application. It shall contain the content feed of all posts local to the user within a designated radius.
- This page allows for the content to be filtered
- This page gives the user the means of interacting with the content feed.

UserInfo_View:

- Shows all recent posts made by the user.
- Shows posts that have been liked by the user.

About_ViewModel:

- The respective view model for the about page. Contains all commands and data items required for binding.

PostMedia_ViewModel:

- The respective view model for the post media page. Contains all commands and data items required for binding.

Search_ViewModel:

- The respective view model for the search page. Contains all commands and data items required for binding.

MediaFeed_ViewModel:

- The respective view model for the media feed page. Contains all commands and data items required for binding.

UserInfo_ViewModel:

- The respective view model for the user info page. Contains all commands and data items required for binding.

UI Mockups

Figure 4.2: Homepage mockup. Shows items in a listed order, which the user can click on to comment. The user can also vote on a post from the home page.

Figure 4.3: Post specific mockup. This is where the user can comment, flag and vote on the post.

Figure 4.4: Other page mockup. This is where the user can access the user's previous posts, comments and it also contains the link to the terms and conditions.

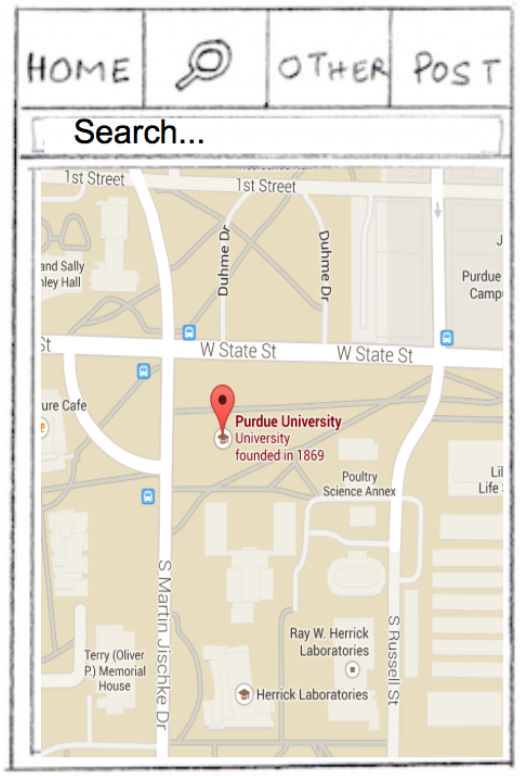

Figure 4.5: Search page mockup. This is where the user can look for other places. The user can either search for a place through text or go through the map and place a pin on a position.

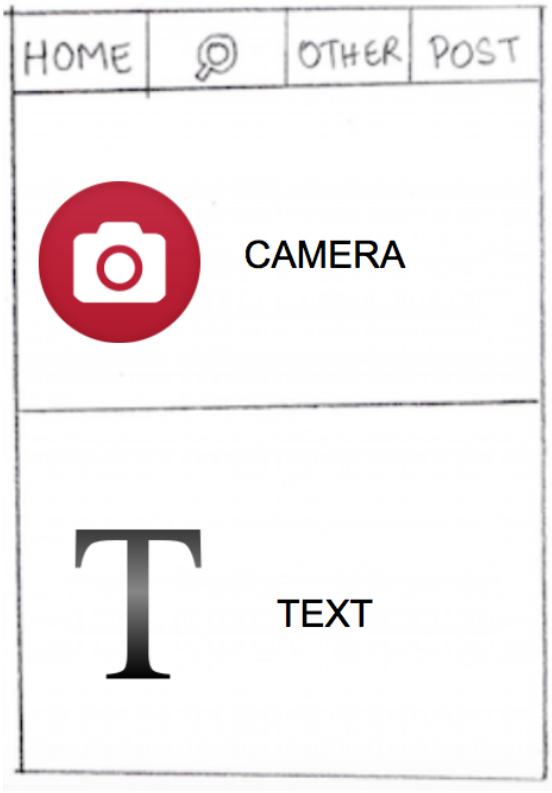

Figure 4.6: Post page mockup. This only lets the user post to the user's locality. The user can either choose to post media in the form of text or pictures/videos, with options to post already created media or take new pictures/videos.