Grapevine

Taokun Zheng
Department of Computing and Mathematical Sciences
tzheng@caltech.com

Aman Agarwal
Department of Computing and Mathematical Sciences
aaagarwa@caltech.com

Aditya Bhagavathi
Department of Computing and Mathematical Sciences
abhagava@caltech.com

ABSTRACT
Grapevine is a photo-sharing mobile application that embraces the virality of photographic media in a fun and intuitive interface. In this paper, we describe the features that make Grapevine unique. We also discuss the front-end and back-end design and implementation process with a focus on the technical challenges that we faced. The application was implemented on the Android platform.

Categories and Subject Descriptors
H.3.5 [**Online Information Services**]: Web-based services; J.7 [**Computers in Other Systems**]: Consumer products

General Terms
Android

Keywords
Photo-sharing, virality, swipe interface

1. MOTIVATION AND IDEA
Grapevine is a mobile app embracing the virality of photographic media through a fun and intuitive interface. It draws upon the best features of today's most popular apps to create a unique, enjoyable mobile experience (figure 1). Users experience a feed of photos uploaded and shared from people across the world. They either like or dislike each photo by swiping it to the right or left, respectively. This corresponds to a crowd-sourced selection of great content, as each action corresponds to spreading the photos they appreciate to the feeds of other users or containing the spread of photos they dislike. Thus, most photos in a feed will be shared by other users who found it pleasing, ensuring quality to passive users through a peer review process.

Active users may upload photos to the network. These are sent to the feeds of particular seed nodes based on the photo content and the photographer's track record as well as the preferences of the seed nodes. Users can witness the spread of their photos across the network as people share or contain their media.

Figure 1: Grapevine combines Imgur's virality, Tinder's interface, and Instagram's design to create a unique, fun mobile experience.

Although users experience photos by strangers, we will promote quality in this network. In partciular, users can see the popularity of their images on their profile, displayed as a ratio of likes to views. This, combined with our transmission algorithm, creates a force for high quality photos.

2. COMPARISON TO STATUS QUO
The value added by this app is simultaneously democratizing photo sharing and embracing virality. Regarding the former, users can experience photos created by people they do not know, and their photos can in turn be viewed by people they have not met. This is not anonymous photo sharing, but every user is empowered to impact the community and have their photo go viral rather than just the elite few with the most followers.

Many photos are posted with the hopes of maximum viewership, so we will make virality the explicit goal of our app. An uploaded photo receives a score based on its spread. Users can see the spread of their and other users' photos on profile pages.

2.1 Snapchat
Snapchat sends content to a particular set of users and then deletes it. By emphasizing privacy, it securely facilitates the transfer of highly personal photos that are not intended for general viewing. Our app is the exact opposite. Users do not choose who to send their photos or from whom they are receiving photos. We aim to create a community that values quality and attempts to create content for the masses. Whereas a user in Snapchat is happy having only his friends see his photo, a user on our app will be happy if his photo cascades across the network.

2.2 Instagram

Instagram allows a user to send photos to the user's friends or family. Hence, it pursues a similar goal as Snapchat, while the only difference is that the photo feeds are not deleted after viewing, and other users may even reload the photo feeds, i.e. like the normal social network feeds. However, for our model, we believe sharing photos with everyone in a defined comunity, so that everyone will have a chance of enjoying high quality photos. Therefore, in Grapevine, the user will not limit their photo sharing to friends or family member only - a larger audience of the community are waiting for the user's high quality photos.

2.3 Flickr

Flickr does not have a unique vision, but rather attempts to serve multiple needs. It's primary use is as a private or public photo repository. As one of the first photo-sharing apps, it enjoys a large community of photographers and quality content. However, exploring others' photos is not Flickr's focus. There is a page showing the most popular photos, but they often only receive about 10000 views. In contrast, we want to create an app singularly focused on exploring content by others. It should also be easy for any user to have his work seen by others, rather than the "rich-get-richer" phenomenon that occurs when only the most popular photos are exhibited.

2.4 Facebook

Facebook is a social network focused on sharing content among friends, messaging and dedicated pages for a variety of interest groups. Our app will focus on photo-sharing in a larger community as opposed to Facebook where photos are only shared between friends. The emphasis on virality is quite different from Facebook's model of photosharing via likes and comments. However, our app will have a photo feed similar to Facebook's Newsfeed.

2.5 Secret

Secret was an app that allows users to contribute anonymous posts to three separate groups: friends (and friends of friends), colleagues at work and those nearby. The groups are completely separate. It also allows private anonymous messaging between users, and messages disappear if they remain idle for 24 hours. While Secret is now defunct, other anonymous sharing apps like it exist.

On the other hand, our app focuses on photo sharing. It is not anonymous and photos are not spread based on group affiliations such as workplace, location, or friend circles. Instead, photos spread and disappear based on their virality.

3. PHOTO FEED

3.1 Functionality

The Photo Feed forms the crux of the Grapevine experience. It is the main entry point to the app after the login page. It displays one photo at a time, taking up the majority of the screen. Inspired by Instagram's design, there is only the image author and timestamp above the photo. Though we do not allow titles, users may enter a description to be shown below the photo (figure 2).

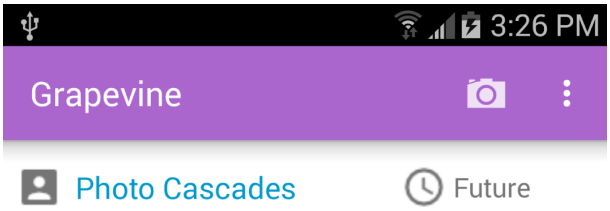

Figure 2: The section above an image is minimal, only displaying the image author and timestamp.

The portion of the Photo Feed below the image handles user interaction and social features (figure 3). Imgur provided a good starting point for many of our design choices below the photo. Icons allow a user to like, dislike, and/or favorite an image. Furthermore, users can click on the comment icon to see and reply to comments on the photo. Indeed, with regard to viral images, the discussion of the image contributes a significant proportion of the content value, making the commenting system critical for Grapevine's success. Furthest to the right is a progress bar showing the popularity of a photo as given by the ratio of likes to views.

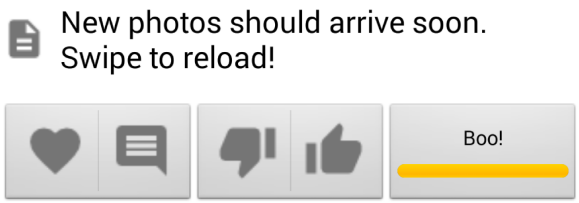

Figure 3: The section below an image shows the image description and facilitates user interaction.

The key distinguishing feature of the Photo Feed is its swipe interface, as used in Tinder (figure 4). After a user is done viewing a photo and its comments, he/she can simply swipe the photo right to like it or left to dislike it. This is reinforced by a green background color when swiping right and red background color when swiping left. The same next photo arises regardless of the swipe direction, as if viewing through a vertical deck of photos. The depth animation also creates a smooth transition by making the outgoing photo increasingly transparent as it is swiped off the screen, while simultaneously increasing the opacity of the incoming photo from underneath. Such an interface facilitates quick and fun perusing through content, which is exactly how users approach viral media.

3.2 Implementation

Implementing the Photo Feed was a hack on two fronts: the underlying data structure and the accompanying animation. Regarding the former, the Photo Feed is built on the `ViewPager` class provided by the Android framework, which supports a horizontal rather than vertical slideshow. Indeed, the content shown by `ViewPager` is meant to be static, so that swiping left corresponds to advancing to the next page, while swiping right corresponds to going back a page [1].

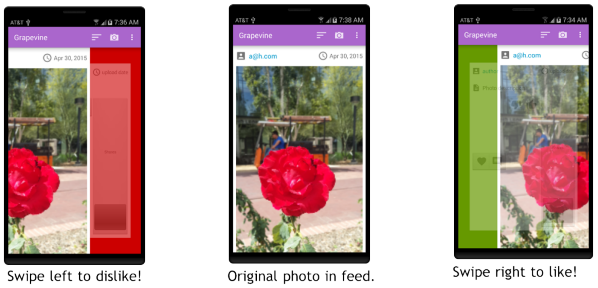

Figure 4: The Photo Feed allows users to swipe left or right to dislike or like a photo, respectively, before moving on to the next image.

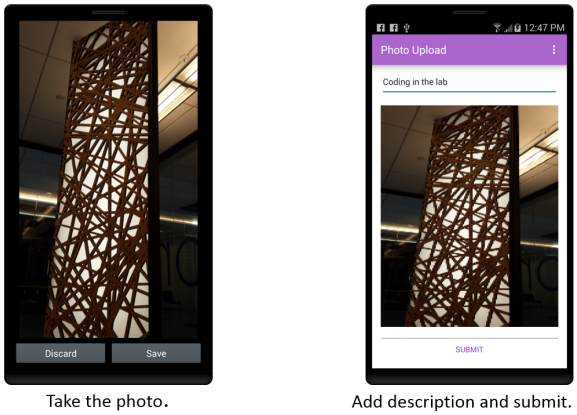

Figure 5: The Camera interface allows the user to click a photo, add a description and submit.

We were able to use this framework by maintaining a slideshow of three photos, with the middle page containing the current photo, and the first (left) and third (right) pages containing the identical next photo. Thus, by maintaining the app on the middle page, the user could swipe in either direction and see the same next photo. After loading the next photo, our app re-initializes `ViewPager` to the center page and loads the subsequent image on the side pages.

Along with the underlying data structure, we implemented a `DepthPageTransformer` to create the illusion of viewing a stack of photos. The code for this animation largely consisted of counteracting the horizontal slideshow animation hardcoded into `ViewPager`, along with tuning page translucences based on their displacement from the center of the screen.

The final incarnation of Grapevine should have a custom `View` class specifically designed for a vertical slideshow, as we expect Tinder does. This would eliminate the redundancy in storing the same image on two separate pages and having conflicting animation schemes. Crucially, it would be more efficient as the view would need to re-initialize after every swipe by the user. For the purposes of this project, our hack allowed us to expedite the creation of a proof-of-concept without compromising noticeably on the user experience.

4. CAMERA

4.1 Functionality

Grapevine provides a smooth and quick interface to upload photos which makes the app fun and easy to use. To promote photos clicked personally by the user as well as provide a simple interface, photos must be uploaded directly from the mobile camera, as on Instagram.

To upload a photo, the user must click on the Camera icon on the Action Bar at the top of the Photo Feed (Figure 2). This directly opens the mobile camera. When a picture is taken and saved (Figure 5), it is displayed in an intermediate screen with a description box above and a `SUBMIT` button below it. The user can then add a description by directly typing into the text box with the picture in view, and then submit (Figure 5). The photo is re-sized to fit the screen as shown.

The final incarnation of Grapevine should have filtering and other photo editing options for photo uploads. This would constitute an additional step after saving the photo and before displaying the `SUBMIT` screen.

4.2 Implementation

The Camera feature required us to accomplish two main tasks: interacting with the mobile camera, and uploading a new photo with description to the cloud.

When the Camera button is clicked, the Camera Activity is started. In the `onCreate` method of the Camera Activity, an Intent to use the mobile camera is sent. When a picture is clicked and saved, the `onActivityResult` method sets up the view (Figure 5) and re-sizes the photo to fit the screen.

Finally, when the user clicks on the `SUBMIT` button, the `uploadBtn` method saves the photo, description and author name to the Parse server. This is done asynchronously in the background so that the user doesn't experience any delay.

5. USER PROFILE

5.1 Functionality

The Profile displays all the photos uploaded by a user labeled by their popularity (i.e. likes/views). The user can open her own profile by selecting the My Profile option in the menu from the Photo Feed. Moreover, if the user likes some photo in the Photo Feed and wants to view more photos from the same author, she can click on the author name above the photo to open the author's Profile.

The photos are displayed in a square grid with translucent labels (Figure 6). This gives a quick glance over a user's activity on Grapevine and also puts the emphasis on how well each photo has been received by the community. Each photo can be clicked to get a full screen display along with the description. The user can return to the grid of photos by simply pressing the Back button on the Action Bar.

The Profile serves as a fun interface for a user to see how her photo uploads are being received in the community. In the final incarnation of Grapevine, the Profile will also be used to follow another user: if a user likes a photo on the Photo Feed, she can go to the author's Profile, peruse the rest of

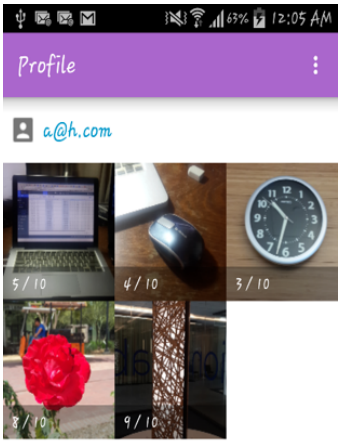

Figure 6: The Profile displays the photos uploaded by a user labeled by their popularity (likes/views). Each photo can be clicked to get full screen display as in the Photo Feed.

the author's photos and then decide to follow the author if she likes.

5.2 Implementation

There were three main tasks in implementing the Profile feature: designing the square grid elements with translucent labels, obtaining photos from the cloud, and managing memory efficiently.

For the grid elements, a `SquareImageView` class extending the `ImageView` class was used. Then `SquareImageView` and `TextView` were combined in an XML file to create the square photo with label design.

In the `onCreate` method of the Profile Activity, the `GridAdapter` and `collectionManager` are set up. The `collectionManager` (described in the Backend section) asynchronously obtains all the photos uploaded by a user from the Parse cloud.

To ensure efficient memory management, when the Profile Activity is paused or closed, the references to the images in the grid are set to NULL so that they can be garbage collected. This is done in the `onPause` and `onDestroy` methods.

6. BACKEND

For our backend, we mostly used the infrastructures written by Parse.com, so that we can just focus on the design our App. To be more specific, we just need to design the contents of each database table, and write some special cloud functions which we want to run remotely in the background. Note, for the database system, Parse has created convenient create, get, query with constraints functions for the front-end to call.

6.1 User

We used the default User table of Parse. Each user has fields of object ID, email, password, creation time, modification time, username, and access control list (specifies the authorization and access level for the user). Thanks to access control list for each user, Parse can help ensure the security of our database and App automatically.

6.2 Photo Feed

6.2.1 Photo Object

The Photo Object table consists all the photos that has been uploaded by the users. For each photo object, we have object Id, author (which is a User Object), description text, the image file (which is also saved in our database as a file), number of likes of this photo, number of views of this photo, creation time, modification time, a boolean field ("hasComment") whether the photo has comments, and a popularity parameter of this photo.

Most of the fields are filled, once a user uploaded the new photo. However, many other fields are modified when another user sees this photo:

1) Whenever this photo is transmitted so a user and the user has chosen to like or dislike the photo, then we will always increment the number of views by one for the corresponding Photo Object;

2) Similarly, if a user chooses to like the photo, we will increment the number of likes field;

3) Whenever a user has commented on this photo, we will make sure the boolean field "hasComment" to be true (which is defaulted as false);

4) Whenever the Photo Object has been modified, in the cloud, we will update the popularity parameter of the Photo Object to be fraction of likes among all views. Note in order to make sure new photos will have some decent chances of being transmitted, we will fix the popularity parameter to be some fixed constant (e.g. 0.5) if number of views of this photo is smaller than some threshold (e.g. 20).

6.2.2 Feed Object

We defined a Feed Object to be uniquely identified by a Photo Object and a User Object, representing the user has seen this photo. Then the Feed Object table can be considered to be the history of all the photos some user has seen. Each Feed Object has the Photo Object, the User Object (i.e. the viewer), an integer status (indicating whether the user has liked or disliked the photo), a boolean field ("fav") indicating whether the user has "favourited" the photo, creation time, and modification time.

Note in our cloud, we made sure to do sanity check that the pair of Photo Object and User Object to be unique among all Feed Objects.

6.3 Comment

The comment table consists all the comments a user has added for some photo. Note for our commenting system, we only allow to one-depth of reply, i.e. you can reply on a comment that is not a reply. For each comment, we have the au-

thor (a User Object), the Photo Object, the comment text, a boolean field ("IsReply") indicating whether the comment is a reply, a boolean field ("HasReply") indicating whether the comment has a reply, number of likes, the parent comment (note if the comment is not a reply, then this field is NULL), creation time, and modification time.

6.4 Cloud Function of Fetching Photo Feeds

As we are serving new photos (according to popularity) each time a user requests, then we implemented a backend cloud function to do the transmission algorithm.

Each time the frontend calls this cloud function, it will provide two parameters: the User Object requesting the photos, and the number of photos it requests. Then in the cloud, the function will first query all the Photo Objects this user has not seen and order them in descending order of the popularity. Then for each photo of the results of the query, we will generate a random number from 0 to 1, and if the number is smaller than the popularity of the photo, we will then choose this photo to be returned to the frontend. We will continue iterating through the results of the query, until we have enough photos or we have exhausted all the results. We will return the list of Photo Objects chosen.

This method not only ensures no photo will be sent to a user twice, but also, because of its randomness, allows photos of low popularity still have a chance (though relatively low) of being transmitted - so the system is fair.

7. APP ARCHITECTURE

7.1 User Sign-in Flow

When the user opens up the App, we will send the user to the user sign-in page. Then we have two separate situations for user sign-in: 1) If the user already has an account, then by entering his email and password, and pressing the sign-in button, we will verify the credentials through our Parse backend; After the user is logged in properly, then we will send the user to Photo Feed page. 2) If the user is a new user, than the user can press the button for "sign-up", and enter his email, password, and username (optional); we will then send those credentials to our backend and create a User Object; if we created the account successfully - note email address should be unique for all users - then we will auto login the user and send him to Photo Feed page.

7.2 Photo Feed

7.2.1 PhotoFeedManager

For the frontend, we have implemented a `PhotoFeedManager`, which locally keeps a queue of Photo Objects, that will be send to the user in the near future. The `PhotoFeedManager` will call the cloud function of fetching to refill the queue, whenever the queue size is smaller than some fixed threshold.

Moreover, the `PhotoFeedManager` is also in charge of updating the feedback of the user to the current photo. Whenever, the user likes or dislikes a photo, the `PhotoFeedManager` will create a corresponding Feed Object, save it to the cloud in the background, and remove this just viewed photo from the queue.

7.2.2 Photo Feed Flow

All our basic functionality stems from the Photo Feed, because as soon as the user signs in, we will send the user to the Photo Feed page. Thanks to `PhotoFeedManager`'s heavy lifting, whenever the frontend want to render some Photo Feed page, all it needs to do is to ask `PhotoFeedManager` for the data of the current Photo Object (since `PhotoFeedManager` always keeps its queue updated and the current Photo Object is simply the head of the queue).

Note, whenever the queue of `PhotoFeedManager` depletes, either because we `PhotoFeedManager` is still in the process of fetching photos from the cloud or we have exhausted all the photos in our database, we will show a default Photo Feed indicating to swipe later for new contents.

7.3 Camera Activity

From the Photo Feed page, whenever the user presses the camera icon in the top action bar, the App will send the user to the flow of uploading new photo:

First, we will open up the default camera App of the user's phone. The user will then take a new photo or choose some photo from his local directory. Next, with a photo image returned, we will show a preview page for the user and a text field where the user can add description to the photo. Finally, when the user presses the submit button, two things will happen: 1) in the backend, we will create a corresponding new Photo Object, and save it to our database in the background; 2) in the frontend, the user will be returned to the Photo Feed page.

7.4 Profile Activity

From the Photo Feed page, whenever the user clicks on the link of the author of the photo feed, we will send the user to the Profile page of that specific author.

Then before we renders the page, our frontend will query our database for all the photos this author has uploaded in descending order of creation time. Finally, we will rendered all the photos, when we have finished our query.

The user can return to our Photo Feed page, by pressing the return button.

7.5 Comment Activity

From the Photo Feed page, whenever the user clicks on the comment icon under the photo, we will send the user to the comment page of this photo.

Before we render the comment page, we will also query our database for all the comments of this photo. Upon receiving the results of the query, we will render all the comments of this photo. Moreover, at the bottom of this screen, we have a special box where the user can enter new comments for this photo. Then when the user presses the "send" button, our App will create the corresponding Comment Object of this new comment, save it to our database in the background, and after we have saved the object, the comment page auto-refreshes itself, so that the user can see his new comment.

Finally, upon clicking the return button, the user can return to the Photo Feed page.

8. FINAL PRODUCT

Our final product is a usable Android application. Although we have not published our App officially on Google Play, we have asked our friends and interested people from the poster presentation session to do some user testing. During our poster presentation session, our App went very smoothly for all functionality.

9. FUTURE WORK

Grapevine currently implements the basic machinery of sharing photos, but does not perform sophisticated data analytics. While our transmission algorithm tracks a rudimentary version of a photo's popularity (ratio of likes to views), it does not tailor photos to users. There is a lot of potential to be tapped in harnessing the data we collect from users and using it to personalize their photo feeds. The following lists some features that would help make this possible.

9.1 Photo Tags

Users should have the ability to tag their photos based on its content. This could be implemented within a photo's description using a hashtag or through a separate tagging protocol.

Tagging photos creates much-needed structure within Grapevine. First, it would let users organize their collection of photos for perusing by themselves and others. More importantly, it could facilitate the learning of users' tastes and preferences based on the tags of photos they tend to like and dislike over time. Tags would also help understand the "state of the system" at a particular moment in time. Grapevine could then not only show the most trending photos, but also display users trending tags like on Twitter and show associated photos upon inquiry.

9.2 Image Categories

Image categories would allow users to take control of their content. While it may be possible to algorithmically learn a user's general preferences, one cannot automatically tune content to moods at the moment. Categories, on the other hand, would let users manually filter their Photo Feed to display photos they are currently interested in.

Image categories would tie well with photo tags. Indeed, a photo tag would be one way of classifying a photo into a category. This need not require photo tags to refer to particular categories explicitly, as this can be learned. The other means of classifying a photo would be through unsupervised learning by associating a photo to a category based on the set of users who appreciate it.

9.3 Followers

Grapevine is designed towards an open network, with users sending and receiving content from people they do not necessarily know. Nevertheless, this vision is compatible with users associating and subscribing to certain other users they particularly like. Allowing followers for a user is one way to let the network form ties within itself.

Following a user would be akin to turning on an image category. The app should handle merging content from various sources by taking into account the quality of the content, the user's tastes, and frequency of content from a particular source. Adding this feature would make Grapevine's functionality more closely resemble establish social networks, however it will still be distinguished by its emphasis on viral content.

10. ACKNOWLEDGMENTS

We would like to thank Professor Adam Wierman for his mentorship. We also would like to express special thanks to Parse for the amazing backend cloud and database support.

11. REFERENCES

[1] W. Jackson. Android's viewpager class: Using viewpager to navigate horizontally. In *Pro Android UI*, pages 497–516. Springer, 2014.