GeoPost System Architecture Document

Michael Davis davism78@uw.edu
Megan Drasnin mdrasnin@uw.edu
Ethan Goldman-Kirst egk35@uw.edu
Matthew Hertogs mhertogs@uw.edu
Neil Hinnant nrhinnan@uw.edu
Katherine Madonna madonk@uw.edu
Andrew Repp ajrepp@uw.edu
Duncan Smith duncan26@uw.edu

System Architecture

System Viewpoints

From the customer's viewpoint, the app consists of a few different pages. The first page asks the user to login via Facebook and then sends the user to a main map page. The map page displays a map with pins on it and shows a post button. When clicked, the pins open up a message pop-up box and the post button opens an overlay for entering a post. The Android "settings" button will slide out a sidebar with a link to the settings page and profile page.

From the developer's viewpoint, each view is an activity object. The map activity is the main page and will direct to other activities based on user input. These activities include the view pin fragment, the post fragment, and the profile activity. The Facebook login activity directs to the map activity. There are two classes for interacting with the database, a database querying class and a database storing class. The map activity interacts with the querying class in order to retrieve pins at given coordinates and send them to the view pin fragment. It also uses the storing class to record which pins a user has unlocked. The post fragment must interact with the database storing class to store new posts. The class representing pins is used by many classes and a class representing profiles is used by the profile class.

Main Modules/Interfaces

MainActivity

This is the main module of the application which lets the user interact with the map and the pins on it. It keeps track of the user id, the coordinate location of the user, and the pins to display on the screen. When the user clicks the post button on this main page, the MainActivity calls the PostFragment class with the user id and the user's coordinate location. When the user clicks on a pin, a DBQuery object is created to query the database for the pin's information and message given the pin's id.

LoginActivity

This module is the first screen that the user sees when opening the app. When the user clicks the Facebook login button, the Facebook server is sent a message with the given Android device id to open a session for the given user. Once the session is opened, LoginActivity makes a call to the Facebook server to get the user's unique id, which the system uses to keep track of user profile information.

Data Storage

Data is stored in a database, utilizing Facebook's integrated Parse database platform. The database stores information about all of the pins and all of the users and will have two main tables. The first is a table that associates each user with with the pins they have viewed and other user data (such as the user's name). The second associates each pin with its location, message, and the user that posted it.

Assumptions

We are assuming that users are familiar with Google Maps and that users have a Facebook account with which to log in. These assumptions allow us to easily use pre-existing tools. A second category of assumptions is that in our initial release, we are assuming we have non-malicious users who do not try to spam post or impersonate other users by intercepting web traffic to the database using services like WireShark. We are not necessarily thinking about security initially because we imagine that we will have a small user base at first, and focusing on the development of the main functionality of our app takes precedence at this stage. That said, the risk to our project is already limited by using Parse. The decision to switch to Parse has significantly mitigated the security problems that would have been faced with direct API calls to AWS SimpleDB, our original database choice. Parse's abstraction of the database removes much of the risks associated with database programming, such as injections or XSS attacks. Further, Parse offers Object and Class level permissions, that prevent or allow objects and API calls from affecting other objects as needed. The main Security concern that is not handled is spamming, and we do not yet have a secure system for development key distribution

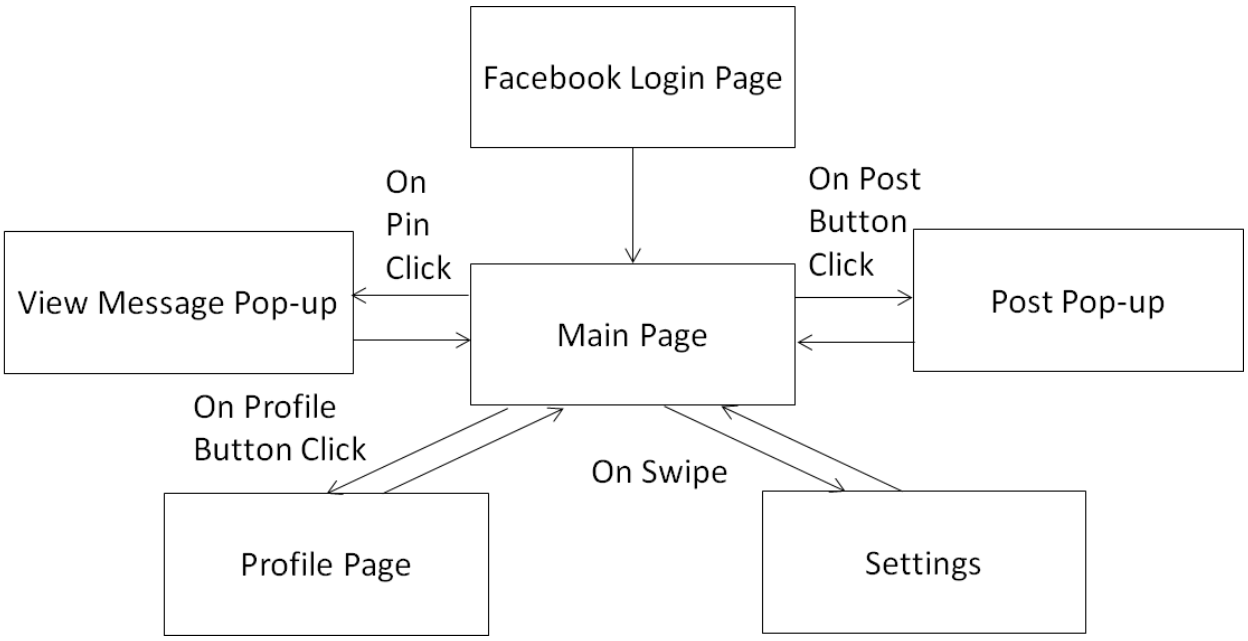
Design Decisions

We considered separating the view from the controller by having a class for each activity class that would act as the controller. This would then make each activity class just part of the view. However, we decided not to do this, because this is not how the Android SDK sets up projects and we wanted to use the built-in structure. We also considered including a table in the database to associate user ids with information such as what posts they made and the number of posts they've viewed. While this could potentially provide us faster lookups of this data, it would also require us to keep this additional table up to date and would require considerably more space. Because we could calculate this information from the other tables we had, we decided it would not be worth the additional complexity of keeping the tables in synchronization.

Diagrams

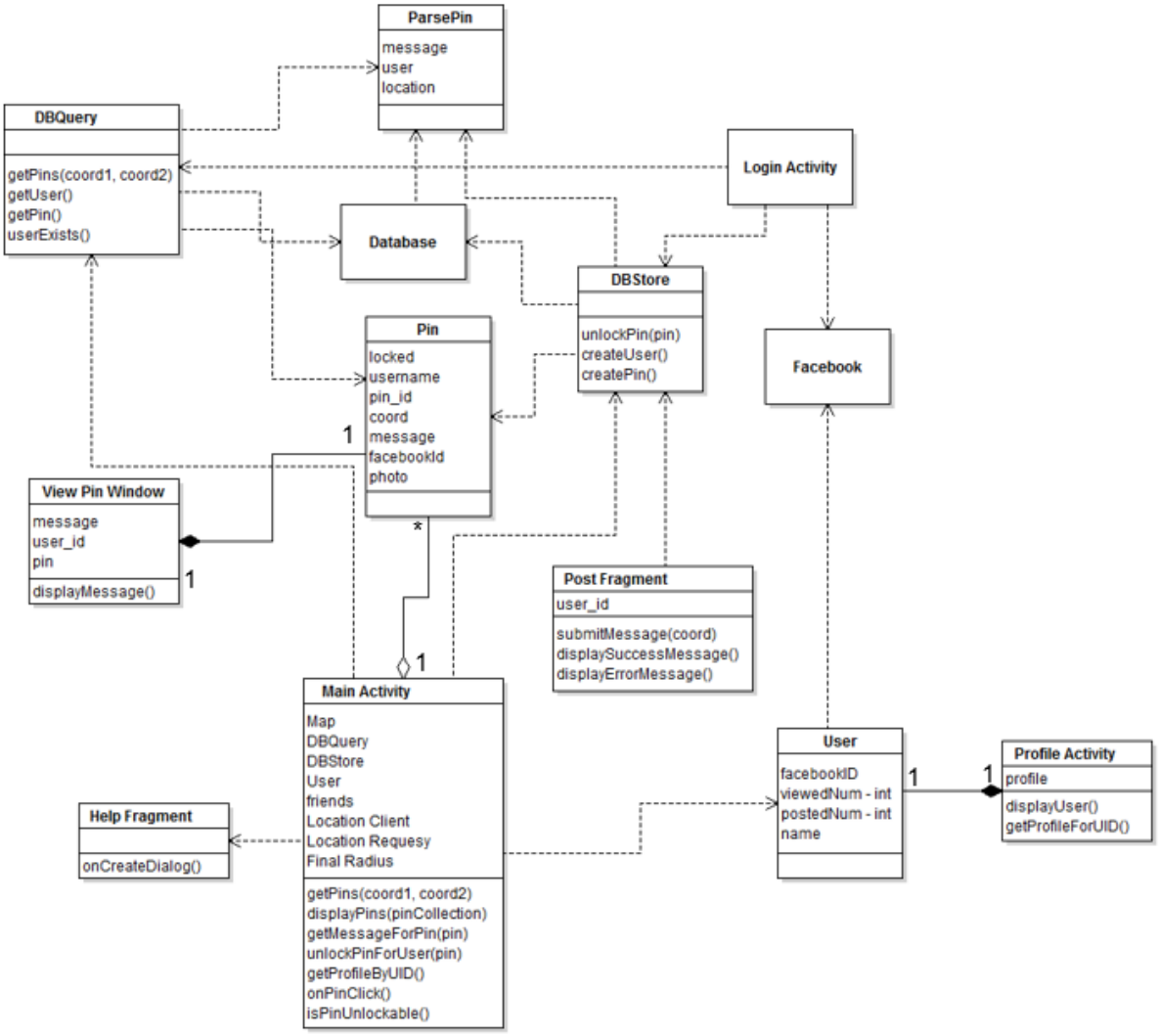
Class Diagrams

Customer View of System Design

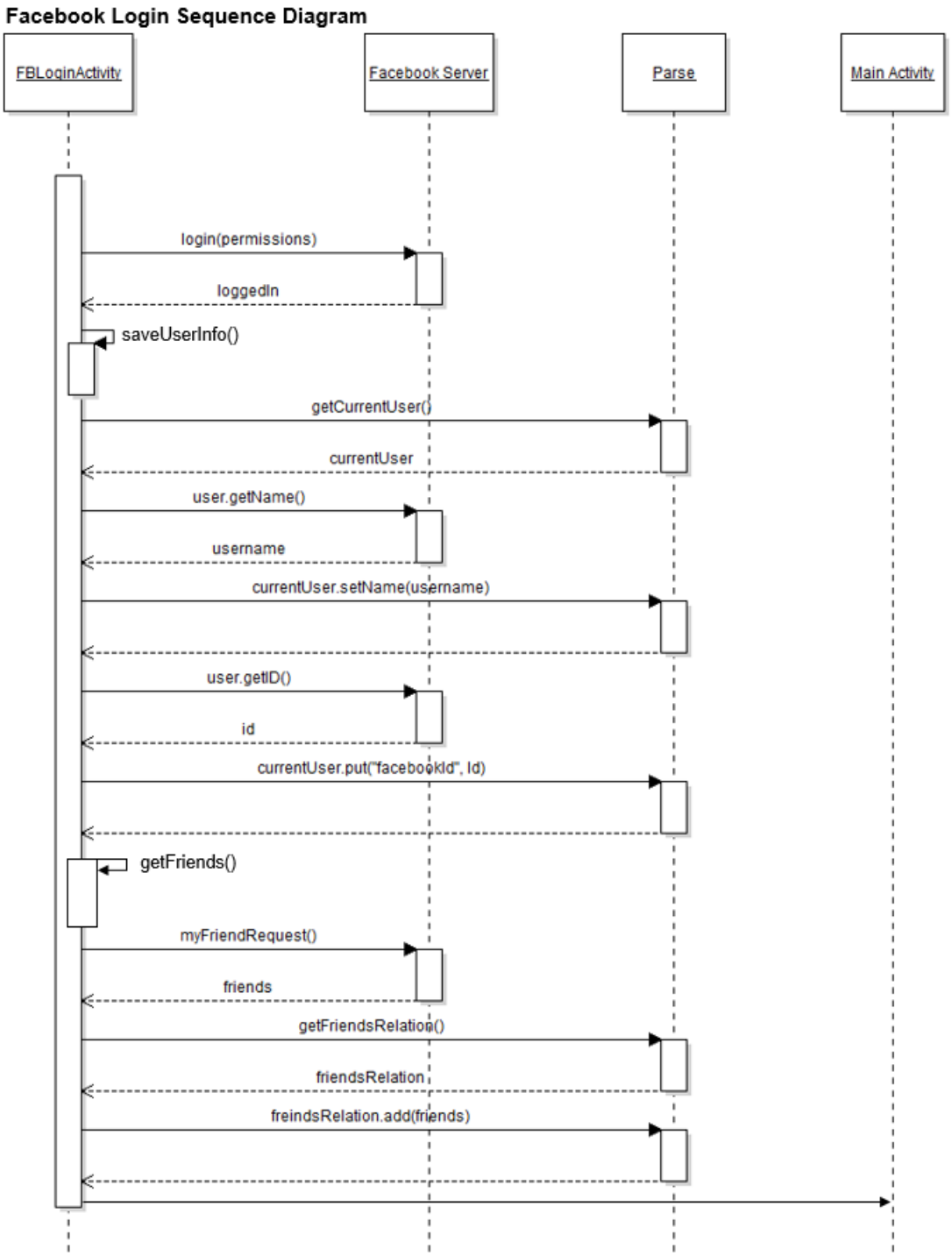


UML Class Diagram

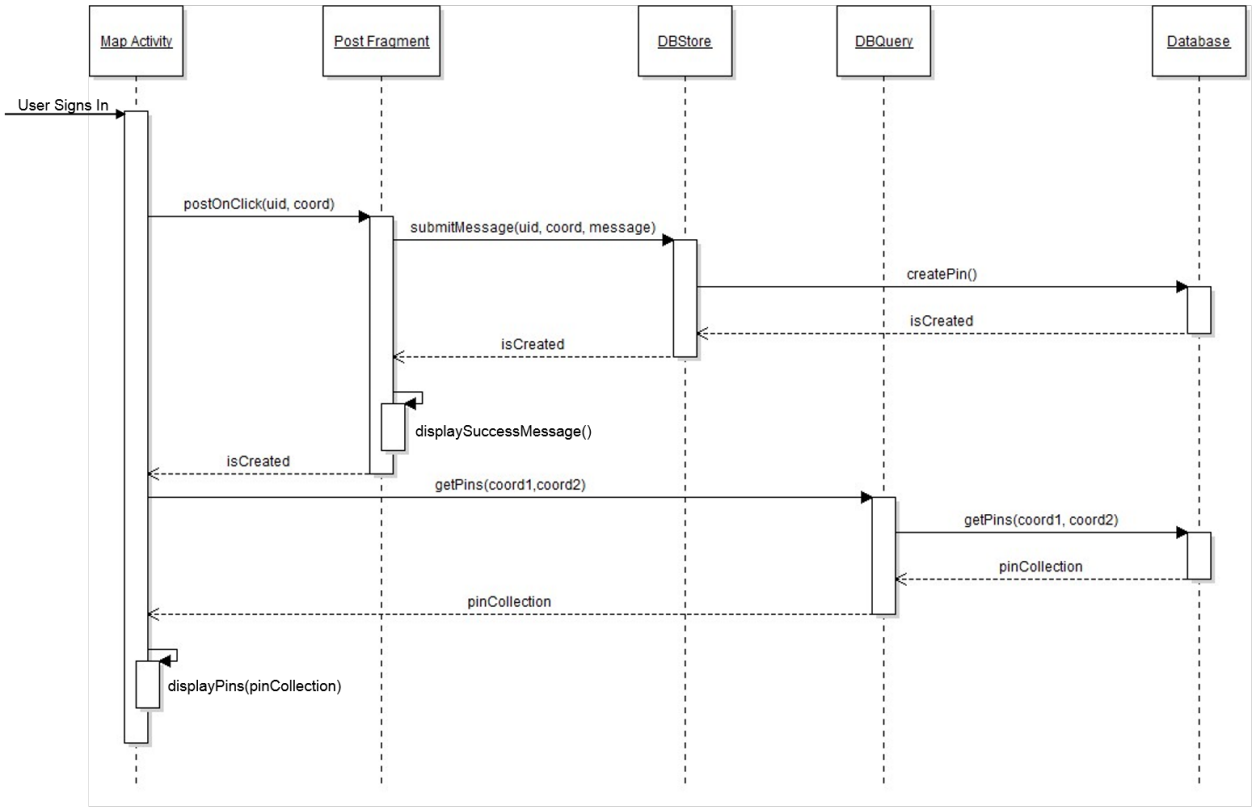
Class Diagram



Sequence Diagrams



Posting to Current Location



- User clicks the post button and a postOnClick call opens up Post Fragment where the user can type the message.
- When the user submits the post, method call to DBStore with the post info and it sends the new pin to the database.
- If the pin was added the post fragment displays a success message and closes, returning back to the main map activity.
- The Map Activity asks DBQuery for the updated pin collection within the bounding rectangle of the map given two coordinates.
- DBQuery gets the applicable pins and passes them to the Map Activity.
- The map activity then displays them on the map.

Process

Risk Assessment

1. **Managing the complexity of designing an application for Android**
    a. **Likelihood of occurrence:** Low
    b. **Impact of occurrence:** High
    c. **Evidence:** We do not have a team member with significant Android experience. We have already spent time looking at the generic Android development process, but it is clear that there is still much to learn. We tasked Mike with finding out how to get the SDK and Eclipse development environment set up, but we will almost certainly encounter some hiccups with Android; it is imperative that we don't allow these to become time drains that waste days. If we cannot figure out how to leverage Android technologies to design our application, then the project is doomed.
    d. **Steps taken to reduce impact:** We are relying on a series of Android tutorials to introduce each member of the team to Android development. Further, when developing the SRS, we tasked team members to familiarize themselves with how different technologies we will use (such as Google Maps and Facebook) integrate with Android. We believe that because Android is well-documented, the chance of us having significant problems with the Android platform is low.
    e. **Plan for detecting problem:** There is no formal testing plan for this risk. It will be obvious if it occurs, as we will not be able to make development progress.
    f. **Mitigation plan:** There is no plan for mitigation. The success of the project depends on leveraging the Android framework.
    g. **Change since SRS:** We believe that this risk has declined since the SRS, as we have become more familiar with Android.
2. **Failure to develop and maintain a comprehensive and efficient integration test suite**
    a. **Likelihood of occurrence:** Medium
    b. **Impact of occurrence:** Medium
    c. **Evidence:** We all have experience with unit testing, but no one has significant experience with systems or integration testing. Further, we do not currently know the Android best practices for developing a testing framework. We have a fairly well-developed idea of how to conduct unit and black box testing on our various application modules, but no strong plan for total systems testing. We are concerned that our coding speed will outstrip our ability to test that code, and in doing so will render our test suites underdeveloped, or worthless. We believe that testing will turn out to be more complicated than expected, but we also believe that we have the manpower to mitigate the effects of this complexity.
    d. **Steps taken to reduce impact:** Duncan has an affinity for testing, so we have assigned him the SDET role, and he will have primary responsibility for the testing framework. He will spend the majority of his time designing tests and the commit framework we will use to prevent bugs from entering the master branch of our git repository.
    e. **Plan for detecting problem:** We need to make sure that we are not committing code ahead of the test plan. We need to enforce a commit-based system of automatic testing to verify that commits that would break the build aren't allowed. Duncan has the responsibility to notify the group if he feels that we are not adequately testing our software.

f. **Mitigation plan:** If we are not able to successfully test our code in a timely and meaningful manner, our first recourse will be to devote more project resources to testing. This will require another team member to join Duncan to form an SDET team. If this fails, we will have to cease development until we can catch our testing up and have designed a more sustainable test plan.
g. **Change since SRS:** This risk was not listed in the SRS. We recognized this risk as we gained a better understanding of what we do and do not know about testing in Android.

3. **Coordination problems amongst the members of GeoPost**
   a. **Likelihood of occurrence:** High (this is almost certain to occur at least once)
   b. **Impact of occurrence:** Variable (long term, the impact would be high)
   c. **Evidence:** We have already had some difficulty scheduling meetings for which everyone could be present. Because there is so little face-to-face time, the potential for miscommunication and ideological drift exists. When we were coming up with the core features of GeoPost, there were frequent miscommunications with regards to which features were stretch or core. We have found that it's important to clearly specify responsibilities for each group member in our team meeting summaries, so everyone has an easy reference for their responsibilities. This coordination issue is sure to manifest occasionally, but as long as we do not allow it to become a long term part of development, we can keep it from becoming a project-killing risk factor.
   d. **Steps taken to reduce impact:** We have several systems in place to help mitigate and even eliminate the coordination problem. We use a mailing list for group communications, we document all work assignments clearly in group meeting documents, and we have access to each other's personal contact information so that issues can be resolved in a quick and decisive fashion. We meet twice a week, and our Friday meeting is relatively open-ended; it may go as long as necessary.
   e. **Plan for detecting problem:** This problem can be difficult to detect because by definition it involves everyone thinking they are working correctly, while in reality, they are not. Our plan to handle coordination is to meet in person as frequently as possible, and to peer review every document or code module we create. In this fashion, we expect that we can maintain a coherent development process team wide.
   f. **Mitigation plan:** If we find that we are frequently gridlocked due to communication issues, we will need to sit down as a group and find a way to spend more time working together, so there is less potential for drift. In practice, this likely means enforcing weekend meetings, and implementing some sort of formal confirmation structure to ensure that everyone is getting their work done on time and on spec.
   g. **Change since SRS:** This issue maintains the significance attributed to it in the SRS. However, since we submitted the SRS, we have clarified how we document responsibilities and established a weekly schedule for meetings.

4. **Issues supporting all chosen versions (4.0 and up) of Android**
   a. **Likelihood of occurrence:** Low
   b. **Impact of occurrence:** Medium
   c. **Evidence:** We know that supporting all versions of Android in the market today is a complicated task. However, our research has suggested that most applications and certainly one as simple as GeoPost, will run on all versions 4.x if the application runs on any 4.x. Google maps, our major Android technology dependency, is guaranteed in its current API level to support Android 4.x. This issue is unlikely to occur, but if it does, it could have moderate to severe impact on the amount of code we have to write to support additional versions.

d. **Steps taken to reduce impact:** We chose to support only Android 4.x specifically to avoid this problem.
e. **Plan for detecting problem:** This issue will be simple to detect; we will build GeoPost for all versions of Android 4. If any version build is broken in a different way than the other versions, we know we have a problem.
f. **Mitigation plan:** If this problem does occur, we will have to decide between eliminating support for the version, or adding conditional code to support the targeted version. This decision will ultimately be made based on the level in question; we certainly need to support the most popular versions of Android 4 at all costs.
g. **Change since SRS:** This issue was not explicitly listed in the SRS, but it was understood at the time. Not much has changed since then; we have planned all along to support only Android 4.0 and higher.

5. **Our database becomes more complicated than expected**
   a. **Likelihood of occurrence:** Low
   b. **Impact of occurrence:** Medium
   c. **Evidence:** We know that our database in initial form will likely look much different than our ideal database with all stretch features implemented. A text-based database is much simpler in principle than a mixed-media database, or one storing data for a complex profile system. Even so, we believe that our goals for GeoPost are not particularly ambitious in terms of database complexity. We certainly can implement all of our beta features without fear. This risk has a medium impact on our project because it may limit the types of media we can associate with map pins.
   d. **Steps taken to reduce impact:** We have planned our feature implementation schedule such that all core features do not require a particularly complex database. Therefore, we can be confident that we can ship the database required for at least a basic version of GeoPost with little trouble.
   e. **Plan for detecting problem:** This problem may be hard to detect. We need to ensure that we frequently test the application after changing the database so that we ensure acceptable load times and that we are able to display database content. In practice, we must perform a full build and test after adding any new layer of complexity into the database.
   f. **Mitigation plan:** Because we only expect to see this problem if we are implementing a stretch feature, our initial mitigation plan is simply to drop the feature. If we have surplus development time, we can return to it later and find a more efficient implementation.
   g. **Change since SRS:** This risk was not listed in the SRS.
   h. ADDENDUM: THIS RISK OCCURRED. WE HAVE SWITCHED FROM AWS TO PARSE. Luckily the change has so far been quite beneficial.

Project Schedule

Milestone Code	Major Milestone	Sub Milestones	Due Date	Effort Estimate (person days)	Implementation Dependencies	Test Dependencies
1	**Zero Feature Release**					
1a		Make basic Android app	4/27	.75	None	None
1b		Add basic Android activity to app	4/28	.375	1a	1a
1c		Determine baseline build process	4/29	1.25	1a,1b	1a,1b
1d		Determine how to make app available	4/29	.75	None	None
1e		Write documentation for users	4/30	2.25	None	None
1f		Write documentation for developers	4/30	2.25	None	None
		TOTAL	5/2	7.625		
2	**Main Map Page**					
2a		Display Google Map	5/3	.75	1	1
2b		Center map at user location	5/3	.75	1,2a	1,2a
2c		Display Post button	5/4	.125	1	1
2d		Swipe out to blank settings screen	5/4	N/A	1	1
2e		Clickable markers on map (No content)	5/5	.25	1, 2a	1,2a
		TOTAL	5/16	1.875		
3	**Database Integration**					
3a		Connect to database	5/3	2.25	None	None
3b		Implement DBQuery and DBStore	5/3	3	3a	3a
3c		Create database interface (setup Parse)	5/4	.625	3a,3b	3a,3b
3d		Allow querying/updating from Android app	5/5	.625	1,3a,3b,3c	1,3a,3b,3c,3d
		TOTAL	5/16	6.5		

4	**Dropping Pins**					
4a		Display pins with content from database	5/9	1.5	1,2,3	1,2,3
4b		Add a pin with content to database	5/10	1.5	1,2,3	1,2,3
		TOTAL	5/16	3		
5	**Facebook Integration**					
5a		Register app with Facebook	5/9	2.25	1	1
5b		Create login page with Facebook login button	5/10	9	1,5a	1,5a
		TOTAL	5/16	11.25		
6	**BETA RELEASE**					
6a		Ensure documentation is up to date	5/14	1.5	1,2,3,4,5	None
6b		Write design changes and rationale	5/14	.25	1,2,3,4,5	None
6c		Prepare demo	5/15	.125	1,2,3,4,5	None
6d		Obtain customer feedback	5/15	.25	1,2,3,4,5	None
		TOTAL	5/16	2.125		
7	**Settings**					
7a		Populate buttons	5/18	.25	2	2
7b		Display legend	5/18	.375	2	2
		TOTAL	5/19	.625		
8	**Profile Page**					
8a		Retrieve and display user pin count	5/20	.5	1,2,3,4,7	1,2,3,4,7
8b		Retrieve and display user viewed count	5/20	.5	1,2,3,4,7	1,2,3,4,7
8c		Retrieve and display user information from Facebook	5/20	.75	1,2,3,4,7	1,2,3,4,7
		TOTAL	5/21	1.75		
9	**FEATURE -COMPLETE RELEASE**					

9a		Obtain customer feedback	5/22	2	1-8	None
9b		Ensure documentation is up to date	5/22	1.25	1-8	None
9c		Test instructions for running	5/22	.625	1-8	None
9d		Set up automated building	5/23	2		
9e		Stretch feature hours	5/23	4		
				9.875		
		TOTAL	5/23			
10	**RELEASE CANDIDATE**					
10a		Perform and document code reviews	5/27	1.375	1-9	None
10b		Perform and document user tests	5/29	1.125	1-9	None
		TOTAL	5/30	2.5		
11	**1.0 RELEASE**					
11a		Perform extensibility exercise	6/1	1.25	1-10	None
11b		Develop demo and presentation	6/1	1.25	1-10	None
11c		Write requirements and schedule postmortem	6/2	1.25	1-10	None
11d		Write individual retrospectives	6/3	2	1-10	None
11e		Write class evaluation	6/3	1	1-10	None
		TOTAL	6/4	6.75	1-10	None
				54.875		
		GRAND TOTAL				

Team Structure

Every member of our group will share in the development. Neil is responsible for project management, so he will do slightly less coding than other group members. Duncan is our group's SDET, so he will focus on testing other team members' code. Katie will manage most of the UI design. Every other team member will develop various parts of the architecture, including implementing and writing unit tests. The team is very open in the sense that every member is expected to take initiative and speak up -- ideas are freely shared and decisions freely discussed so that everyone contributes to reaching a consensus. As PM, Neil is responsible for making a final decision when disagreements persist.

Our team meets every Tuesday from 9:30 – 10:30 and every Friday from 3:30 – 5:00. We hold additional meetings throughout the week when necessary. We mainly communicate via email, and our team's email alias is cse403sp14hermes@uw.edu. Both our team and individual weekly status reports are available through the class Catalyst Dropbox, which is accessible by everyone in the class. Our wiki can be found at our GitHub repository at https://github.com/nrhinnant/GeoPost/wiki.

We have divided the work for this project based on six sequential milestones:

- **Milestone 1: Zero-Feature Release**
  By this milestone, we will create a basic Android application with one activity. Additionally, we will decide on a build process, write documentation for users and developers, and determine how to make our source code available to the public.
- **Milestone 2: Creating the Main Map Activity**
  Goals for this milestone include displaying the map in our app, centering the map at the user's current location, displaying a post button for submitting pins, and creating a menu for navigating to other activities (such as the profile activity).
- **Milestone 3: Database Integration**
  This milestone will require us to set up the database and to implement our DBQuery and DBStore classes.
- **Milestone 4: Adding Pins to the Map**
  Goals for this milestone include displaying pins from the database on the map and allowing users to submit their own pins to the GeoPost database.
- **Milestone 5: Facebook Integration**
  This milestone will require us to register the app with Facebook and to create a login activity for logging into the app via Facebook.
- **Milestone 6: Creating the Profile Activity**
  This milestone will require us to retrieve the user's profile picture using Facebook SDK for Android. It will also require a few database accesses for getting the number of pins the user has unlocked and submitted.

Responsibilities for these milestones correspond to the below team assignments. Teams will be responsible for completing their relevant program functionalities, and for integrating with the other teams as needed.

Database Team: Megan, Andrew, Katie
Main App Control: Matt, Mike, Ethan
-This team writes the Android control logic, the Pin system, and integrates Facebook and Google Maps into the application.
Test: Duncan, Neil

Test Plan

Unit Test Strategy

Unit testing a system component is entirely the responsibility of the owner of that feature. Unit tests are absolutely critical in maintaining software integrity, and as such, thorough and complete test suites per feature will be required. In addition, black-box style tests will be written by our dedicated SDET for each interface that is produced by a team. It is required that the unit tests written by both the SDET and the feature owner pass before the feature's branch may be merged into the master. The entire test suite will be run before every commit. The test script will save a log of the test results. The dev will include this file in their commit, so that we can track exactly which tests started failing at which point in the development cycle. It is unclear at this time if mocking interfaces will be necessary in creating good unit tests.

System Test Strategy

System testing is a completely different beast. These tests will likely have to be maintained at the same rate that the software develops, as it is unlikely that the SDET will be able to predict precisely how the system will develop over time. Because these tests test the integration of each component, they will be difficult to write at the beginning of the project when the system itself is still in flux. Therefore, less emphasis will be placed on these at the beginning. They should, however, be sketched out early. Writing a test that mirrors a sequence diagram might be a good start. These tests will be written primarily by the SDET, requesting assistance from component owners as needed. They will be run alongside the automated unit tests described above.

Usability Test Strategy

Small and frequent usability tests with a wide array of users will be crucial in developing a good user interface. As soon as we have a build up and running, we will want to have a small number of people (fewer than five) test out the app. We will take the problems that they uncover or the confusions they have and fix them. In the early stages of this type of testing, anybody will work as it is likely that there will be obvious, glaring issues that prevent the user from progressing (therefore, hallway testing will work fine for this stage). These agile tests will help us to ensure that we don't waste our time testing one build on twenty or more people, each reporting the same set of issues. As the builds progress, a wider spectrum of users will be needed as testers. A broad range of smartphone expertise, age demographics and likely use cases will allow us to find the largest set of issues.

Bug Tracking

Luckily, GitHub has a wonderful built-in issue tracker. Our SDET will work closely with the PM to ensure that bugs are properly assigned and prioritized. Before an issue can be marked closed, the assignee must write a test that catches this bug, showing the test fails while the bug exists, and succeeding while the bug is fixed. Only then can the issue be marked closed.

Github has a standard format for naming bugs, and classifying them as *Development*, *bug*, *duplicate*, *help wanted*, and several other issue categories. In addition to the standard template for a Github Issue, we'd like all issue reports to follow a particular format containing all the sections listed below:

1. Descriptive Bug Name
   Ex. "No Facebook login button present on running application."
2. Description of the bug
   Ex. When a user runs GeoPost, they are presented with a Facebook login screen that has text boxes for email and password but no login button.
3. Severity
   Ex. High
4. Technical Category (Core application, Maps, Database)
   Ex. Core Application
5. Test Case
   Ex. Run the application, attempt to log in.
6. Reproducibility
   Ex. Always reproducible
7. Contact information
   Ex. Neil Hinnant, nrhinnant@gmail.com

Issues and Concerns

Unfortunately, no member of the team has experience with tools like Selenium or the UI testing tools built in to the Android SDK. It is unclear at the moment how much of an impact this is likely to have in testing the application. Further, how do we automatically test system architecture components that rely on Android components? We will likely need to use mock interfaces in order to test system components that interact with the database, which will be difficult.

Documentation Plan

It is our hope that our app will be so simple to use that no help menu is required beyond a legend for the map as well as a brief 2-3 screen tutorial. Many very successful apps, such as Twitter or Snapchat, have been able to teach the user how to navigate the app's features through using the app as opposed to reading through a lengthy help guide. Further usability testing will likely tell us whether or not this strategy is tenable.

Coding Style Guidelines

https://source.android.com/source/code-style.html
http://google-styleguide.googlecode.com/svn/trunk/javaguide.html

Obviously, there are a lot of rules to remember and follow. Thankfully, 99% of them are common sense. The ones specified in the Android style guide are a touch less intuitive, but correct nonetheless. No code file will be allowed to be considered 'done' until the code has an @reviewed_by tag at the top of the file specifying who has verified this code as adhering to style. This cannot be the same as the @author tag.

Extensibility Exercise

We believe modifying GeoPost to support serialized input via web URLs would be a straightforward task for a competent developer. The major components are a new input menu that accepts a URL and downloads JSON (or other format) data, and the parser that takes this data and converts it into a collection of pins. Once a collection of pins exists, the main logic in GeoPost requires a few simple modifications to add these pins to the map in a new color, and in a persistent fashion. Persistence is achieved easily, we simply never add these local pins to the Parse database, and they'll be cleared upon exiting the application. Furthermore, if we don't add these special pins to the Parse DB, they automatically won't contribute to user statistics. There are some straightforward changes necessary to support things like the new filtering option, and the new pin color. Below are listed the files, and their particular pieces, that would require modification to support this new use case. We work under the assumption that the new pin color would be green.

File Changes:

Main Activity:

-add new temporary option to sorting enum
-change addPin to check for a temporary pin and change its color to green
-change onOptionsItemSelected to handle clicks on the menu item and open a fragment
-change onItemSelected to handle choosing the temporary sorting option
-add a case to getVisible to check for the temporary option
-change drawMarkers to not delete temporary pins
-add a listener on the new fragment that takes in a collection of pins and stores them in the local pin storage map

main.xml (in res/menu):

-add new item to menu bar

New Fragment (off of settings menu):

-allow users to enter a url and hit an "ok" button
-parse the JSON in the url and return a collection of pins to the listener

stringarrays.xml (in res/values):

-add a new item to the sorting array for the new sorting option

dialog_help.xml:

-add a new item to the key with a green pin

DBStore:

-unlockPin() changes to support temporary pins by not querying database if temp pin

Pin class:

-boolean isTemporary, and all the necessary constructor and getter/setter boilerplate