Wager

Tyler Kaye, William Chance, Richard Bush, Michael Swart

Introduction

Producing a full scale, fully functioning iOS application has been a challenging but rewarding experience. While no one in our group had extensive experience in programming in Swift, each member of the Wager team brought some unique experience and skills that in the end made us a very effective team. Through countless hours of hard work helping each other work through bugs and hash out specific design plans, we learned many valuable lessons about programming in a collaborative environment, designing a modular system and performing thorough testing of a large system. In this report, we hope to give a window into our experience, the things we learned, the extra challenges we wish we could have tackled if we had the time, and some (hopefully) helpful advice to our peers who undergo this process in the future.

Brainstorming and Project Planning

When we first met up as a group, every person on the team had several ideas that they had been thinking about since the beginning of the semester in anticipation of this project. This was nice because we had no shortage of ideas to contemplate, but tough as well because it was difficult to come together around one idea with so many strong concepts on the table, from a peer tutoring hub to a location based micro-blogging platform. In the end however, we settled on the idea of a peer to peer social gambling application that became Wager. We settled on this idea because we felt that it was relevant to us in a fun way. All of us have gambled on the outcome of pool games, who gets the higher score on a midterm exam and other random events. Usually, these bets are settled with a begrudging venmo payment and forgotten about. Our vision for Wager was a home for all of these micro bets where people could keep track of all their bets, see their overall profits and losses, and check out what their friends were betting on.

Once we decided on our vision, we began making broad decisions about how we were going to reach our goal. The first decision we made was to develop the application fully in Swift for iOS devices. We considered this choice at length, ruling out a web based application as the type of bets we envisioned on the service were small spur of the moment events, not something that you would want to reach for your computer to place. Further, we decided that committing to a single system, we could develop a more intricate, easy to use, app with a clean interface with fewer issues than if we attempted to develop the application using cross platform framework like React Native. Then, the decision was between iOS and Android. We chose iOS for its near ubiquity on college campuses and among our friends, not to mention the fact that all four members of our group are users of both Macbooks and iPhones.

The next decision we needed to make before getting started was which of the many available database hosting services we were going to use. The plethora of options was overwhelming at first, MySQL, SQLite, Realm and many more, but we began narrowing the field by hunting for databases that are easy to integrate with Swift. In the end, we settled on Firebase, for its ease of access, scalability, user authentication, and a simple and easy to use API that we believed would make it easy to get the functionality we needed for the application. Another big plus was the Firebase console, which provides free analytics and bug reports that we anticipated would come in handy.

The Minimum Viable Product

After making the broad design decisions of which platform and database to employ, we set out to create a list of the attributes that were absolutely necessary for the functioning of our application in order to give a more concrete definition to our vision.

- Profile
    - From the beginning, we wanted Wager to be a home for all a user's bets on so that they could review their betting history and figure out how much they made or lost overall. We decided to acheive this by creating a profile page where the user could view all of his or her bets, their profits, and other user specific information.
- Feed View
    - In order to separate Wager from the rest, we knew that we needed to create an environment where users could do more than just pay out bets directly to another

user, which is what people are already doing using Venmo. Instead, we wanted Wager to be a place where users could discover and browse bets posted by other users to increase engagement in the application. We decided that the best way to enable users to discover new bets was a bet feed view that is the cornerstone of the application. We imagined the feed as a place where bets posted by all users could reside, to match people who have opposing views on the outcome of an event but don't necessarily know in advance someone who would take the other side. The feed view is the crux of what makes Wager a "social" gambling application.

- Add new bets
    - In order to support wagering on any event no matter how big or small, we decided to allow users to create their own bets instead of posting odds on a fixed set of events. Each bet would be connected to a challenger's profile (profile of the user who created the bet) and then to the challengee's profile after the challengee accepts the bet.
- Venmo Integration
    - In order to support social gambling among anyone from any location around the world, we decided that creating an interface with a mobile payment platform was absolutely necessary.
    - We devoted a lot of time to the decision of which mobile payment platform to use. We ended up choosing Venmo because of its widespread use among college students and young adults who we see as our target audience. Venmo is very easy to use and connects directly and securely to its users' bank accounts, making transferring money to a paypal account or other bank specific money transferring platform unnecessary.
    - This decision was complicated by the fact that Venmo took down its public API to access its services from outside the app itself and is now offering an exclusive beta of its "pay with Venmo" service. Unfortunately, Wager was not eligible for this beta service as applications that "facilitat[e] peer-to-peer transactions between two Venmo users are not supported".
    - However, our group became aware of a workaround in which developers can generate a url scheme on iOS devices that opens the Venmo app payment screen with a custom description, payment amount, and payment venmoID preloaded. This workaround means that completing payment of a bet would only take two taps from the user.

Implementation of the MVP

Some things are easier said than done. This is true in many things, and especially in computer science projects. We decided to use GitHub in order to facilitate working as a team on the code for the project. Believe it or not, this proved to be one of the major learning curves we had to tackle as a group during the implementation phase of our project. Several of us had used git locally on our machines for version control but were not familiar with pull requests, pushing, and merge conflicts. There were several frantic messages to our group late at night with concerns about having overwritten someone else's commits, or opaque errors that prevented opening the app storyboard Xcode. With concerted effort and aggressive Google-ing we were able to solve many of these errors GitHub, but doing so proved time consuming and hindered us in our ability to work efficiently. In order to mitigate this, we decided to implement a standard procedure for making changes to code, committing, and merging to the master branch. We learned that if everyone followed these steps exactly, it greatly reduced the number of issues we had with needing to roll back commits, re-commiting and in some cases even having to delete the entire project and re-clone. The procedure we created is as follows.

1. Make a new branch
    a. Git checkout -b <branch name>
2. <make changes to files>
3. Add and commit changes
    a. Git add .
    b. Git commit -m <message>
4. Pull new changes from the master branch
    a. Git pull
5. Merge your branch with the master
    a. Git merge <branch name>
6. Push to GitHub
    a. Git push origin master

By following this process each time we wanted to make changes, we greatly reduced the number of problems we had with overwriting each other's changes. Once we got into our rhythm of working collaboratively using GitHub, we began making progress much more quickly. We decided as a group to meet biweekly to establish an agenda for the week and then assign specific tasks to each person in the group. This procedure greatly increased our effectiveness as a group as assigning a specific deliverable to each individual established an accountability system so that everyone was on top of their work. This also allowed us to dole out tasks according to which topics each person either had experience in or was interested in learning more about.

In the first stage of implementation of the MVP, our task was to set up the basic view controllers we would need and design the basic layout of the feed view and the profile. We originally set it up so that each view displayed static content and then gradually added dynamic content as we refined our setup of the database and continued to define our definition of what we wanted to offer. One of the more challenging aspects of this stage of the process was designing the bet flow lifecycle. The process of bet creation, acceptance, and eventual

settlement began as a very simple set of states associated with a particular bet, but gradually evolved over time as we sought to add more functionality to increase the quality of the user experience. This took time as each time we decided to update the attributes that each bet can have, we had to erase the data in the database and re-add everything.

Next, after all of the content in the profile and feed views was dynamically updating, we needed to deal with the subject of synchronous versus asynchronous loading of data. Obviously, we would want all of our queries to load synchronously so that everything would load quickly but we quickly learned that this is not always possible. We did not want our users to be frustrated from an app with bad performance. At the same time, we knew that we needed to sacrifice speed for functionality in some cases. One prime example of this is in the loading of the profile. We store all of the profile information including the user's name, username, gender, pnl, etc. in our database so we want these values to be present before we make a call to them. If, for example, we try to find the profile's name when there is no profile loaded then the app will crash. We realized that the best and only way to deal with this problem is with asynchronous loading of data. On the other hand, the profile image takes longer to load and is not imperative for the immediate use of the app so we allow that one to load synchronously. If the profile image never loads because the internet is so slow it is not the end of the world. The profile page was the first time that we realized that loading asynchronously is sometimes unavoidable. Throughout the rest of the development process, we were constantly trying to decide if there was a way to load something synchronously or if asynchronous loading was necessary.

Beyond the MVP:

We were fortunate enough to have a team comprised of students who were willing to put in the time to make a truly special project. Furthermore, each member of the team had a unique set of ideas for Wager that they were determined to implement. Below are a few of these features.

1. **Geolocation-based Querying:**
    a. Although this aspect of the application was not in the MVP, most of the members of the Wager team thought that it was an integral part of the platform. People want to see what the people around them are gambling on, and we wanted to make the application very similar to "Yik Yak" which was able to unify campuses using GeoLocation. Getting the User's location was incredibly easy using the CoreLocation framework; however, storing this information was a bit trickier. We ended up using GeoFire which is a separate module developed by Firebase. GeoFire is optimized to be used in conjunction with FireBase and enables queries to be made from a specific location and specified radius. This is done by essentially converting the Latitude / Longitude into a more useful form that can easily be searched and input into the database. Using GeoFire was very useful and simple, but installing it into the project was a bit challenging. There are an abundance of resources, but this is a very fast-paced project that is rapidly

changing and being altered. Thus, the current version in CocoaPods generates errors when being used with the most recent Firebase CocoaPod. Thus, instead we had to link the Objective-C framework file into the project using a Bridging Header which was a bit challenging to navigate. If any future students would like to use this module and it still has not been fixed, the below link explains how to implement Firebase and Geofire.

   i. https://www.youtube.com/watch?v=m2y748P3vPw

2. **Friends:**
   a. People want to know not only what people around them are betting on, but also what their friends are gambling on. Thus we implemented simple friend-functionality to enable users to add / delete friends from their profile page. Then in the filters page, users would be able to select whether they would like to see wagers only from their friends or from everyone.
3. **State Listeners:**
   a. One amazing aspect of Firebase is the ability to implement state listeners which will dynamically be alerted to a change in the database (addition or alteration of an element). Thus, our pages will automatically update themselves when someone accepts a user's bet, or when a bet has been added to the feed.
4. **Search Page:**
   a. We wanted to add in the ability for users to search for wagers and profiles dynamically. Thus, we added a search page to the application that searches through the name and description of wagers in the database as well as the first name, last name, and username of each profile. The results then populate a TableView that will bring the user to the respective page.
   b. Usernames are unique, so any user can be found easily and friended in the search page by looking for their username.
5. **UI Constraints:**
   a. One thing that IOS does not make easy is to place UI constraints in the project to ensure that the application will look good on all devices whether they have small, medium, or large screen sizes. The best way to learn this was to try, fail, and repeat this process countless times. Using constraints can be incredibly frustrating and tiresome, but ultimately they produce a superior product.
   b. One way that we were able to make these constraints work is to limit any input that users can put in. Any time that a user can type something that will be stored in our database, we check it appropriately and then show a popup message if it does not fit our criteria. Using checks like these, we limited input so that it will look good on the screen. For example, the bet name is limited to 50 characters so that when looking at the bet view the whole message can be seen in large enough font.

What would we change:

Given more time (or even starting the project again) there are a few things that the project might do differently.

1. Database Structure:
    a. The current database structure does things in such a way that it makes it easy for us to use. Each bet and each profile is tagged with a unique key, and we often query for items based on this key. If we were able to dive deeper into the Firebase documentation to understand how things are being stored, there could be some optimization with how our data is being stored.
2. Experiment with other Databases:
    a. Ultimately we used Firebase because it was incredibly easy to use. However, it key/value method of storing all data essentially as a JSON object has serious limitations. There is most certainly a lot of wasted space in the database due to this method. Additionally, the Firebase Swift driver does not enable complex queries. Currently, we can only query based on one factor (and then we sort through all of the data to remove the non-pertinent objects) instead of performing complex queries that occur at the server. Nevertheless, using Firebase was the best way to get an MVP finished and implement the extra functionality that the app needed.
3. More Complex Friend Functionality:
    a. Very basic friend functionality was implemented in this project, but given more time some more complex functionality could be added to the application including viewing all of your friends in a TableView and even being able to directly challenge a friend in a wager.
4. Arbitration Process:
    a. This was a constant challenge for this application. In designing the application we spent a lot of time thinking about the “worst-case user”. In the case of Wager, the “worst-case user” would refuse to pay up on bets that they lose. Although safeguards were put into place to limit this user’s ability to scam other users of the application such as the rating system and our ability to remove users with low ratings, it was a major limitation of the application. In order to abide by the law, we were unable to enforce that the loser pays the other end of the wager. Thus, given enough time we would spend a substantial amount of time figuring out this problem because it is a potential source of anger for users if they do not get paid.
5. Social Media Functionality: Likes and Comments:
    a. Given that this application aims to connect people, it would be an improvement if users were able to like and even comment on the wagers that they see in the feed.
6. Notification System:
    a. The final improvement that could be made to the application is a notification system that alerts the user about the state of the bet lifecycle for bets that they are a part of. These notifications would state things such as “Mike has taken your wager” or “You have lost your bet to Tyler”. Using a notification system would enable to user to be updated when one of their wagers progresses through the bet lifecycle.

Thoughts for next year's class:

We would highly recommend having a meeting before even starting a project to get on the same page regarding Git and make sure that each member understands how to use the basic functionality of git (branch, add, commit, checkout, merge, pull, push). In fact, we believe that it would have been helpful to look at git a bit more in class for COS 333. Overall, the main piece of advice it to pick a project that you are interested in because then it feels easy.

Conclusions on the Project:

This project (and this class) has been a truly incredible experience that we would all highly recommend. As students we often have to push away side-projects in order to accomplish our work, but having the ability to essentially work on a side-project that you are very interested in while getting class credit for it enables you to produce an incredible application and learn how to learn in the technology industry. The project will hone your stack-overflowing skills, but more importantly it will teach you how to work collaboratively with others.

Wager is not without problems, and there are still some challenges that need to be solved before the platform can be distributed. However, it had lent us invaluable experience in not only developing mobile applications, but also in learning skills necessary to work in the industry such as communicating clearly with those around us, setting and meeting deadlines, and always keeping the user in mind while developing any product.