Team Final Report

May 4, 2018

Team 3 (Jack Taylor, Mitchel Smith, Elan Kainen, and Jason Wen)

Introduction:

The purpose of this project is to develop an iOS app that allows users to plan and schedule trips with other travelers with similar desired destinations. Functionally, this app will let users make trip propositions and match them with others based on both travel and personal interests. The app hopes to take advantage of the rise in popularity of the sharing economy and success of services like Uber, Tinder, Airbnb, and other such apps.

The software will be using the individual pages of the app as building blocks for the entire app. This is useful for two reasons: 1) It allows us to build up a base and utilize the prototyping model by building a little bit on each page and then testing that everything works together, and 2) It allows us to work on multiple pages concurrently without fear of breaking something else at the same level (i.e. changing something on the profile page won't affect the trip selection page since they do not interact).

The primary communications will be the pages of the app with the server. There are three primary user-visible pages (Profile, Trip Creation, Trip Selection) with the possibility of others being added later. The Trip Creation and Trip Selection pages need to be able to access information on the profile page as well as communicate with the server. This will allow auto-filling of information during trip creation and basic filtering during Trip Selection. The Trip pages need to be able to access the server to check what trips exist. The server never needs access to the app itself. It should only ever be queried by the app never query the app.

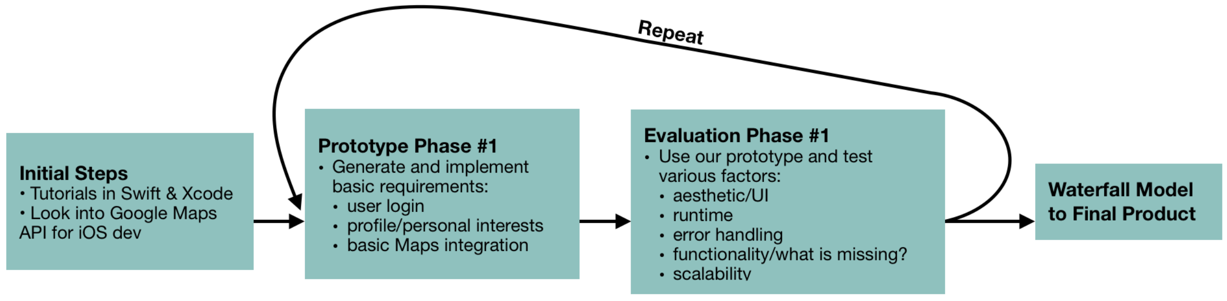
This final report will be a self reflection on our making of this app, comparing what we have planned with what we have built, thus finding flaws in our project architecture and management plans and avoid same errors in future project developments.

1. Project Management

We were pretty far off on the lines of code estimate. The primary reason for this is that we ended up using a visual programming language which defeats the purpose of the lines of code measurement. Also, the use of AWS instead of an Apple produced server allowed for us to use lots of AWS's built in features to do things we had initially planned to write ourselves. Our high level schedule was also not very accurate, primarily because we got a later start on the project than we had originally intended. The main risk that arose was productivity issues, which we had rated as the most likely risk to begin with. Our intended risk management strategy for this was to simply eliminate the non-essential features of the app, such as the Map API, and we stuck to this well. Most of the core functionality of the app is slated to be completed while some non-essential features such as the Map API and more detailed search methods have been deleted.

2. Process Model

For this project, we intended to use the spiral/prototyping model as our primary process model and use the waterfall model as a secondary support model. The waterfall model was intended to be the model we utilized on each of our iterations throughout the development process. The actual sequence of activities that we planned to perform were expected to follow:


Repeat
Initial Steps
• Tutorials in Swift & Xcode
• Look into Google Maps API for iOS dev
Prototype Phase #1
• Generate and implement basic requirements:
• user login
• profile/personal interests
• basic Maps integration
Evaluation Phase #1
• Use our prototype and test various factors:
• aesthetic/UI
• runtime
• error handling
• functionality/what is missing?
• scalability
Waterfall Model to Final Product

We created this outline before beginning the project, and as was to be expected, followed it as best we could. We pursued the spiral model because of its focus on prototyping. This approach seemed to lend itself well to our expectations of ourselves and current abilities at the beginning of the course. The process of creating a quick prototype by implementing a few key functionalities seemed logical as we gained more experience using Xcode. We also expected that it would be a challenge to create a complete and infallible set of requirements, which made the spiral/prototype model ideal for its key feature of leaving space to add requirements on each iteration while avoiding the worst of requirements creep. We also thought that this software model approach would be helpful was its ability to fit our busy schedules and allow us to work independently and meet.

Looking back, this process model did work for a few of the reasons that we selected it for and we were also able to notice some of its shortcomings. For one, the prototype model worked well as we were able to add one functionality at a time, getting it working while not worrying about the next feature to be added. This helped keep us focussed and prevented the team from going down rabbit holes and getting caught up implementing multiple features at once. This would cause problems in the development process as we would not give each feature the focus and time required to implement it well. Problems in this project on the process model front cannot really be attributed to the process model itself. The spiral model worked well for our initial plan, but the unforeseen challenges that occured with each team member learning a new language and coding environment slowed us down at multiple points through an iteration.

We were able to follow the overall process model that we laid out for our project. The main drawback was that we ran out of time to implement some of the features that we planned, however we were able to follow our trajectory during the time we had.

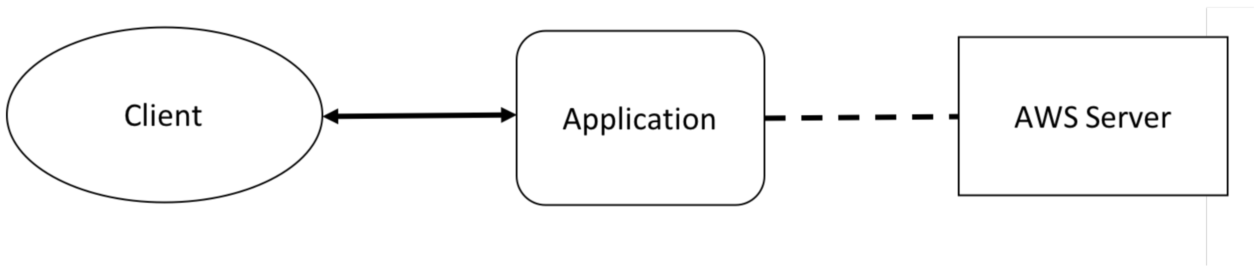
3. Software Requirements

We met most of our requirements. Originally, we had planned on including: Profile creation and management, trip creation, trip selection, and a google map API. Of these, we expect to complete both trip creation and selection. The others were cut due to time constraints and unforeseen difficulties caused by the at-first-glance accessible AWS Mobile server. Both learning Xcode and dealing with undocumented AWS server issues caused many timing problems as we ended up doing more troubleshooting than we ever could have anticipated. So rather than having those and the map API both implemented at a rudimentary level, we made the decision to forego the map API entirely and focus on smooth trip creation and selection trips.

While we did not have any new high level requirements, the details on the trips changed a little bit. The trips now include slightly more information that originally planned, such as cost, but are no longer linked to specific location so will not be searchable by location. However, this implementation of spatial coordinates as a part of trip objects would make an ideal addition to Travel Buddies. Also, we were able to use AWS as a base for profile and trip creation, as well as some basic search features.

4. Software Architecture

In our software architecture specification, we listed three main components: the client or user, the application itself, and the server if we were able to implement it in the time allotted for the project. As shown in the diagram below, the client will continually interact with the application, and the application would make requests to the server when necessary to keep the user and trip proposal data updated. The client and application will interact on a continuous, 2-way basis, where the client observes changes based on their actions in the application. Interaction with the server will be a request-reply basis where user sends a request through the application to the server. The user will never directly access the server nor will the server request anything of the application.



So, we split up into two teams of four, team 1 is in charge of developing the visual, UI component, and team 2 is working on the AWS Server and integrating it into the application. We expected this approach to work far better than it actually did. It was a challenge to completely separate the UI from the server, as our AWS supported backend ended up needing to be very closely tied with the UI. This is the result of the visual coding language that Xcode makes a very important part of iOS development. We knew there was a visual component to Xcode, but did not predict that would be close to impossible to extract the server from it. We are now finishing up the connections between the application component and the AWS server component so that the application can retrieve data and user info from the server.

We don't think there's additional component necessary at this point, keeping the architecture simple and avoiding errors are our priorities right now.

5. Software Design

Our designation of two components, the server and the User Interface, have remained unchanged. Although, our actual design is only about a 6/10. The design of our user interface and the capabilities that such grants to a user are still the same, with the exception that the user is unable to find other users and trips via the implementation of a mapping API. A user is still able to create a new account, login into existing accounts and create trips as well as join new trips (predicted soon), all actions that are being manipulated through a UI component.

While many of the UI components are still intact, we will not be developing them with the same back-end complexity as we initially intended. The server is able to take in new user accounts, store existing accounts, hold created trips and keep track of the number of participants in an existing trip. A user that would like to join an existing trip is able to do so, but our server will not be linking said trip back to the user profile. The server should be able to increment/adjust the amount of participants on a trip when a new user has joined, but will not be attributing a created or joined trip to the specific users (stored on the server) that have decided to create or join them. Such is to say that on the back-end, users and trips are completely separate and are not linked to one another, but rather merely exist within the database.

The reason that we were not able to implement the maps API and sharpen the links between the UI and the server, as evidenced by the users and trips not interacting with one another on the back-end, is largely due to time constraints. Our idea could be a full-blown application with an entire full-time team working to develop it to its full-complexity, but we simply did not have the time as full-time students. Furthermore, none of our team members had ever utilized XCode or written in Swift before, meaning that at times, working past obstacles and troubleshooting errors became overly arduous. Due to our inexperience, there were times when our productivity and efficiency were delayed due to the fact that we were building while simultaneously learning. Overall though, our entire team is very happy to have been able to jump right in to something challenging and engage with a new skill set and language.

6. Summary

It has definitely been a fun "journey" working on this "Travel buddy" app. All of our team members have been very cooperative and productive in doing our own part of the assignment, whether it's learning Xcode skills from online tutorial, working to code software components, or finish each Team report. We learned a lot about Swift and Xcode which is helpful simply for getting more experience with a wide array of tools. We also practiced our teamwork skills, as these types of projects are often not easy to organize.

We can definitely make more progress after the semester ends since this is such a great social media idea and it is in a fairy unexplored space. There are huge opportunities in the market of which we could take advantage. Although it can have a steep learning curve for anything past simple UIs, Xcode is a very valuable tool; we were able to create a smooth three-page interface after watching several tutorials. However, it's hard to debug in a programming language (Swift) that none of our team members has ever utilized before. Difficulties arose when we received an unknown error and StackOverflow posts Google searches could not help us. This was especially the case with the AWS server, as some of our team members would be required to spend hours trying to debug a problem with the AWS implementation. AWS is very well documented for setting things up assuming nothing goes wrong. However after something goes wrong there is minimal support and StackOverflow posts are few and far between. We believe that these issues may be attributed to the fact that AWS can do so much more than we require of it. As AWS is becoming the industry standard, it includes many security and permissions capabilities, some of which came into play in delaying our progress.

We would 10 out of 10 recommend a future team attempt to construct an iphone application just because first of all, Xcode simplifies the coding process so we can focus more on ideas and architectural aspects. Secondly, it is a valuable user interface to know considering how many people run iOS devices and download and use apps every day. We would, however, suggest that the team has a better idea of what iOS development entails than we did. It is not as easy as the tutorials make it sound. Following along with a tutorial things seem clear, with Xcode specifically, generalizing what you have seen to your own new implementation goal can be a challenge. This is, of course, the case with most languages and environments, but we believe it to be even more relevant with iOS development. We also might suggest that students not use an AWS backend, as although it is a valuable skill, it caused us many headaches. Overall though, this experience has been incredibly beneficial for the entire team and we will take away more applied-CS confidence and knowledge of both how software is produced and why it is produced that way.