Scavenger – A Map Based Social Application

CS M117 – Special Wireless Experiment

Connor Kenny 304437322

Jahan Kuruvilla Cherian 104436427

Introduction

When we first heard about the Special Wireless Experiment that we had to perform for CS M117, we thought it was a great opportunity to create a social application that we would actually use. After thinking of and testing a couple of other ideas, we decided to go with a sort of treasure hunt based social interaction map where the individual users could leave messages and images behind for others to see. This content would be pinned to a map, so that one would really only be focused on the events occurring around them. The application includes Facebook login for security and built in messaging and image capturing capabilities. The sleek and modern design definitely makes the app a pleasure to use, and seeing all of your posts, along with the posts of the other people around you, makes the entire experience one great big scavenger hunt. The app began as MemCap, which was to stand for Memory Capsule, because we wanted people to be able to store their memories (in messages and pictures) for others to see, but since it is really more focused on what you can find, we decided to switch to the name Scavenger. Overall, the application utilizes numerous wireless technologies that will be discussed in depth in the following sections.

Technical Background and Implementation

In order for Scavenger to function correctly, we needed numerous wireless technologies to work perfectly in sync. The application connects to multiple servers wirelessly and works on both WiFi and Cellular connections. We chose this type of connection over Bluetooth because we needed an infrastructure based network rather than an Ad-Hoc network. We wanted to be able to connect users around the globe – not only those sitting within Bluetooth range of each other. In terms of our data, it travels wirelessly through the TCP/IP layer for reliable

communication, which is especially important when sent over 802.11n WLAN. The most obvious technology we utilized is a wireless database to store our users' data, including messages and images. On top of that, we need to be able to communicate with Facebook in order to accomplish the Facebook login capability. Finally, we needed to communicate with Google's Maps servers in order to know where the user currently is located as well as place the users' content in the application. Overall, these three technologies each worked in unison to be able to create the application we know as Scavenger.

Firebase – No-SQL Database

In order to implement a database that keeps track of our users and their data, we utilized Firebase. Firebase is Google's tool in order to make and grow your applications. Specifically, we utilized Firebase's database and authorization capabilities. First, we used Firebase's cloud-based No-SQL database to store our users' posts. Since the database is non relational, it does not store the data in tables like other databases with which most people are familiar. Instead, Firebase uses a JSON format and stores the information in a dictionary format. We were able to communicate with the Firebase servers through the application layer because the SDK encapsulates the HTTP GET and POST requests for us. This made all the data communication very easy, because we only had to write a line or two of code to do the entire process of GETs and POSTs. For our application, we stored the following data in the No-SQL database: payload (message/photo), latitude coordinate, longitude coordinate, Facebook name, and Facebook profile picture (URL). We need the payload to display the data, the coordinates to display it in the correct location on the map, and the Facebook information to give the information an identity. Since Firebase is a real-time database, we are able to monitor when a user adds content to the database through the

Firebase website. Since the database is in JSON format, we are only able to store specific formats of data, which is not a problem for the easily-converted-to-text elements like messages, coordinates and names. Unfortunately, photos are not easily stored in text format. We were able to accomplish this by converting each photo in the payload to a string of characters in Base 64, and then converted them back after retrieving them from the database. For the Facebook profile picture, we actually only store the URL of the image, which is easily converted to text, but this will be explained in more depth in the next section.

Facebook Login

One of the most important aspects of our application is its ability to keep track of who is logged in so that the content on the app has an owner. We did this by adding Facebook Login functionality to our app. We use the Facebook SDK to be able to communicate to their servers and then once the user is logged in, we are able to retrieve their name and profile picture, so that we can display them when the user posts on the app. We do this communication through their API, which utilizes the wireless connection through the user's iPhone. As mentioned earlier, this connection can be either cellular or 802.11 WLAN. The user's name is easily accessible when one contacts the Facebook servers with the user's logged in credentials, but the profile picture takes a little bit more work. We actually only store the URL to the profile picture in our database in order to save space as well as increase speed of retrieval. When the application downloads the content from the Firebase database, it simply performs a lookup of the URL in order to retrieve the user's profile picture, which it then displays in the corresponding view. This way, if the user changes their profile picture, we will always get the correct one, instead of having to continuously check for updates.

Google Maps

Our application is based on being able to place both messages and photos onto a map to which all users have access. Instead of creating our own map, which would be insanely difficult and unreasonable, we utilized Google Maps SDK and API. By importing the basic framework, we were able to easily and quickly add Google's technology, so that the users could use the maps that they were already using on a daily basis in our app. Combining the users' location data – which is given to us by Apple's own inbuilt Core Location functionality, Firebase's database, and Google Maps, we were able to keep track of where the users added their content, store it in the database (as coordinates), and finally display the corresponding icon on the map in the correct location. Overall, Google Maps is an integral part of our application because it provides the users with a familiar user interface and allows us to add the functionality we need quickly and efficiently.

User Interface

Early on, we realized that the UI of our application was going to be extremely important because we need its functionalities to be obvious to the user. Here we will give a walkthrough of how a user would utilize the app as well as how each piece of the interface interacts with all the functionalities mentioned above, especially those that included wireless.

When the user opens up the app, they will be brought to a login screen. It displays our application's name: Scavenger along with our logo and a Facebook login button. Upon clicking this button, the user will be brought to Facebook's website in order to login and verify their credentials. If the sign in is successful and the user has allowed us to access certain information about them, like their public profile, they will be brought to the main page of our application.

This page is primarily a view of Google Maps at the location of the user along with three buttons.

The button on the top left is for taking photographs with the user's device. Upon clicking the button, they will be brought to another screen that has the look and feel of Snapchat, although we did not utilize any of their API's. We were able to utilize existing API's made by 3rd parties in order to make this functionality a reality. If the user decides to take a picture and send it – accomplished by clicking the capture button, followed by the send button – the photograph, along with the user's Facebook name and profile picture will be sent to our Firebase database. After, the user will be brought back to the map screen, and will notice a photo icon has appeared at their location. When the icon is clicked, the photo and the user's name and profile picture will appear on the screen.

The bottom left button is used to send messages to other users, in a similar fashion to the photograph sending functionality mentioned above. When the button is clicked, a text box appears, where the user is able to type in their desired message. Upon clicking the 'Send' button, the message, along with the user's name and profile picture will be sent to the Firebase database. Then the user will also notice that a message icon has been added to the map at their location. As one can see, both the photo and message functionalities operate similarly from the perspective of the user, despite have quite a few differences on the back-end.

Finally, the button on the bottom right is utilized in order to check for any recently added content. It does this by contacting our Firebase database and retrieves all the information that it needs to construct the icons on the map for each message and photo that has been added. This way, if users were trying to show their friends' something, they wouldn't have to wait for an

automatic update that could take a long time to come. They would simply click on the update button and see the new content pop up instantaneously.

Clearly, the user interface does an amazing job at guiding the user through all the functionality of the application. They are able to create content, send content to the servers, retrieve content from the server, and maintain their security by logging into Facebook.

Challenges

We ran into numerous challenges and obstacles while trying to complete Scavenger. Being a team of only two people, with one not having any prior iOS development experience, it was an uphill battle from the start. We struggled with figuring out an idea that we thought was truly worth our time. We started off looking into group music streaming, but eventually decided upon what has now become Scavenger.

The first class of challenges we encountered had to do with linking together all the different frameworks and technologies we used. We made our application for iOS, which meant that we used Xcode to write and compile our code, and Cocoa Pods in order to utilize the other frameworks. We came across a lot of problems with the different pods interacting badly with each other, especially the Google Maps one because it is over 100MB. In order to do source control, we used GitHub, but the Google Maps pod needed to be stored on their Large File Storage (LFS) instead of the regular server. Overall, these problems had nothing really to do with the actual coding of the application, but more to do with the bookkeeping and structuring.

The next class of challenges were those that occurred when actually trying to create the project. One of our group members needed to learn swift and iOS development basics along with how Firebase works, while the other took on the brunt of the UI and linking everything together

so that it would work seamlessly. Some of the notable challenges were figuring out what we needed to store in Firebase as well as how we needed to store it. This took a lot of trial and error, and while the method we are currently using works, it may not be the best method when trying to scale. Also, we had some problems with getting users signed in to Facebook along with linking that to Firebase. After reading the documentation for both, we were able to bring them together and get a database full of users and their Facebook accounts.

Future

At the moment, Scavenger is a fully functioning application, but there is always room for improvement. In the future, if we are to continue working on the project, we would make sure to be able to build it to scale as well as maintain security. We would definitely want to make sure that we check the content posted by the users before it is sent out to the database because it would be very bad if users posted explicit or inappropriate content. Also, we would need to figure out if Firebase is the best option for our database, as well as how to store the data in the JSON format. If we had an enormous data set, we might want to split up the data into headers and actual data so that scanning through by location can be done more efficiently. Another option would be to sort the data by region so that we can more easily look through only the data that is relevant, as populating the entire world's map isn't realistic. We would also need to look into removing data from the database after a certain amount of time, so that the content does not overfill the map UI. Overall, we do have a lot of room for improvement, but the application does everything that we want as well as has a beautiful user interface, which makes the application a complete success in our eyes.