final-project-final-opal

Cabell Glancy (bcg7mf) Matias Rietig (mjr9r)

github: https://github.com/UVA-CS4720-F17/final-project-final-opal

---

Radar

Our app, Radar, will function as an all-purpose geospatial message board. It allows users to post messages anonymously to a shared map for other users to discover them. There is a real lack in the market for a strong spatially based messaging app. While Snapchat currently includes features to display user's locations, no current messaging app allows for anonymous public location based messaging. Allowing users to anonymously post messages to real world features and landmarks opens up a whole new method for communication.

Platform Justification

We think that the target audience for our app would definitely be digital natives in Western countries that like similar simple messaging apps and social networks such as Snapchat or Instagram. Since iOS is much more prominent in Western than in developing countries, the choice of development target was straightforward for us. Also, the MapKit of iOS and the new features that were introduced with iOS 11 (MarkerAnnotations that we use a lot) seemed very exciting and opened up interesting design choices. Due to the nature of our app, we knew that we would have to implement persistent data storage and this is significantly more convenient in iOS/Swift.

Major Features

- Main Screen (**Map**): This is the heart of our app and users will spend most of their time on this screen. Users can see all messages dropped within a certain (user-specified) range and click on the annotations to see the message content. Messages are sorted and distinguished by different categories like "Funny", "Cute", "Educational", ... If enabled in the settings, the mapview also provides a "Quickdrop" option, which allows the user to compose a message within the map view and drop it without changing tabs. Also, users can bookmark messages to save and review them later in the "Saved" tab.
- Compose Screen: This screen allows users to compose message in a more detailed way. They can choose a filter, the range in which the message should be visible, and the duration the message should be visible. Unless bookmarked, the message will disappear after the specified time.
- Saved Screen: This screen has two tabs and is divided into "Sent" and "Bookmarked". Both are table views of messages that have been saved or bookmarked. They will remain in the app until uninstall.
- Settings Screen: This screen allows adjusting some of the app's settings. Users can enable/disable the quickdrop function on the map screen, they can set default values for quickdrop distance and duration. Also, users can specify the maximum range from their position that they want to scan for new messages.

Optional Features

- **GPS/Location Services:** GPS is obviously an essential feature of our app. We use it to get the user location on the map view and drop messages at said location.

- **Firebase Implementation:** The main map screen pulls and filters the messages that were dropped by users from Firebase. There is a refresh button on the map that allows the user to check Firebase for new messages in their surroundings. The map view also refreshes whenever the user changes back to its tab. Messages dropped via quickdrop as well as the compose function will be pushed to Firebase and pulled whenever the map view gets refreshed.
- **CoreData:** We use CoreData to save messages that have been sent by the user and to save messages that the user has bookmarked. The second screen in our app, "Saved", displays aforementioned messages in two separate tableviews that are dynamically populated from CoreData. The CoreData entity that we created consists of a message object (the Message.swift class we created inherit from NSCoding so it can be saved as a Transformable in CoreData) and a boolean value that indicates if the message was sent or bookmarked.
- **UserDefaults:** Our app settings are stored with key value pairs in the UserDefaults. For example, we retrieve this data to filter the messages to be displayed (personal defined scanning range) or to show/hide the quickdrop field on the main map screen.

Testing Methodologies

We tested on different devices and in different locations to test and verify GPS data. Also, we tried to catch different cases of permission-related scenarios. We learned not to rely on the Xcode simulator because the experience and functionality can, especially for features like GPS, significantly differ from what happens on the actual devices.

Usage

No specific usage instructions required. But here's how to make pancakes:

- In a large bowl, sift together flour, baking powder, salt and sugar. Make a well in the center and pour in milk, egg and melted butter; mix until smooth.
- Heat a lightly oiled griddle or frying pan over medium high heat. Pour or scoop the batter onto the griddle, using approximately 1/4 cup for each pancake. Brown on both sides and serve hot.

Lessons Learned

- Constraints are the antichrist.
- Debugging for mobile is a slower process in some ways since, especially for more sophisticated features, it requires pushing and installing the app to the device which can take a long time. Also, sometimes the error messages that Xcode throws are rather vague and require a lot of research and trial & error to solve.
- Firebase is great for testing and playing around with different data while running the app because everything happens in realtime. The hardest part is parsing the data correctly to get reading and writing to the app to work correctly. The syntax that is used to define methods in Firebase is anything but intuitive.
- CoreData and UserDefaults are incredibly easy to implement and Swift/Xcode does a great job in abstracting the "annoying" parts of implementing a persistent SQLite data storage for developers.
- Using the MapView and especially understanding how annotations work is really hard to begin with. Again, once you wrap your head around it though, the possibilities of customization are sheer endless.