Team Kernel - Project Final Report

What we did: Compared mobile software engineering life-cycle of **React Native** with native **Android** by creating a mobile application to solve the Navy's need for a tool to help coordinate the running portion of physical exams. The application is able to geolocate and concurrently synchronize time data and relay emergency notifications between all conductors of the run.

Significance to developer community: React Native achieved significant similarity to the Native Android version and is a very powerful tool. Although our tests and surveys show it's a bit slower than its native counterpart, its ability to produce iOS and Android applications with one code base makes it a good choice for small teams or startups that want to support both major mobile platforms. Since it's built on the React framework, it's a great tool for web developers looking to produce mobile apps with very little prerequisite knowledge.

Team Responsibilities

Pair 1
Suhani Singal (ss4925) – Android Application / UI Mockups / Firebase Backend
Angela Fox (amf2227) – Android Application / Survey

Pair 2
Khaled Atef (kaa2168) – React Native Application / UI Mockups
Scott Lennon (sl3796) – React Native Application / Survey

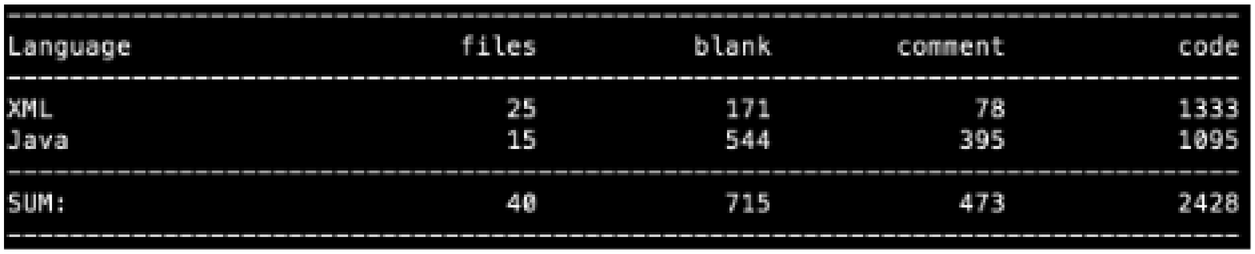
Frameworks used/tested: React Native, Android, Firebase

Deliverables:

1) UI Prototype: https://moqups.com/atef.khaled@gmail.com/tNnkOzvl/
2) SafeRun Android Repository: https://github.com/suhanisinghal/saferun
3) SafeRun React Native Repository: https://github.com/KhaledAtef/SafeRun
4) Final Presentation: Conducted in class on Thursday, April 28th 2016

Project Details

SafeRun Android: Since we (Suhani and Angela) had experience developing for Android in Android Studio, we developed the native Android SafeRun app. Our user interface methodology was to look at the mockups, and the React Native app, and recreate the same user experience using Android Studio. The resulting UI of the application amounted to 2428 lines of code (not including boilerplate or static resources), with 1095 lines for the Java files (implementing transitions between Activities, defining 'on-click' methods etc) and 1333 lines for the XML files (formatting buttons, text items, and Activity layouts). [Figure 1]



```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
XML                             25            171             78           1333
Java                            15            544            395           1095
-------------------------------------------------------------------------------
SUM:                            40            715            473           2428
-------------------------------------------------------------------------------
```

Figure 1: Android: Total Line Count (Excluding Boilerplate)

Overall development experience: Android Studio is based on the Intellij IDEA framework, which means that developers familiar with the Intellij integrated development environment will quickly feel at home navigating Android Studio and making use of its robust functionality. Android Studio provides a flexible Gradle-based build system, code templates, a visual editor for ‘drag and drop’ layout editing, and a suite of lint tools.

Setup: The setup process in Android was relatively painless. The developer simply needs to download Android Studio, and be somewhat familiar with the development and build options available to them. It helps if the developer has access to an Android device for testing. However, even without an Android device, developers can test and interact with their app using Android Studio’s emulator. The emulator is relatively slow and requires a fair amount of CPU processing power, but it remains a good option for developers without Android devices.

App Structure: For SafeRun, each Activity in the app had a java file and a corresponding XML file. Android Studio effectively imposes separation of components, keeping the code base organized, and conventional. This is helpful for developers navigating large app codebases, and very beneficial for legacy code, as it should be easy to identify where each activity's Java and XML and other related files are located.

Developers can make UI changes in the XML file and then view the effects of their changes in the visual editor pane - and even make drag and drop changes within the visual editor. However, dynamically generated views must be previewed and tested on the emulator, as the virtual editor cannot execute the Java components.

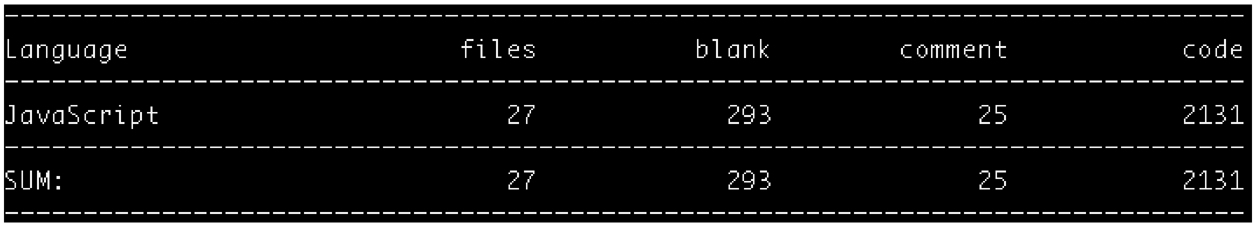
Challenges: Despite the robust development framework provided by Android Studio, there is still room for improvement. During the development of SafeRun, Team Kernel encountered various Android specific setbacks that impeded speedy and intuitive development. First and foremost, Android Studio’s chronometer does not support milliseconds. Given that SafeRun is partly for timing races, millisecond functionality was highly desirable. There are some unofficial workarounds available on GitHub, but we think that should be an integrated functionality.

Secondly, it is non trivial to change run-time rendered UI elements in Android Studio. For example, when we made a decision to change the radio button UI element (as in the list of observers and drivers viewable in the facilitator’s OrganizeNewRun Activity to a clickable TextView UI element), the code in the Java file needed to be changed, and some of the methods that had been members of the radio button object were no longer available or different in the TextView object. This obstacle means that it is in every Android Developer’s best interest to have a completely finalized UI before beginning implementation, which is not in alignment with a more agile, iterative development methodology.

Finally, as we saw in the demo, code that executed without a problem on both the Android emulator and on an Android device, suddenly did not execute during the presentation on a different emulator, causing one student to remark, "That's how I can identify which app is Android, because I've developed in Android before, and sometimes things break for no reason". This inconsistency in how the code performs makes testing challenging, as developers cannot be certain that their code will execute as they expect on all Android emulators and devices.

Conclusions: Despite the three issues mentioned above, developing in Android Studio is ultimately a streamlined and enjoyable experience. When a developer falters, the documentation is there to inform and guide them, and the community of Android developers is active on forums, and generate targeted tutorials, making self-teaching simple. Furthermore, Android Studio makes setting up the environment and building, testing and debugging the code much simpler and faster. Despite the potential to create whatever user interface a developer desires, the resulting UI has a distinctively Android look and feel with regard to transitions and performance that users respond favorably to.

SafeRun React Native: Since we (Khaled and Scott) didn't have any experience developing for mobile platforms, we decided to tackle the React Native implementation. We pair programmed the entire project and occasionally split off to do individual research if we faced a problem we couldn't overcome quickly. At first, our user interface was limited by our XML and CSS inexperience, but with time we progressed to being able to render high quality buttons and layouts. Since React Native mixes the XML layout tags with the business logic Javascript and CSS-like styling, the code is all in the JS files. Total line count: 2131. [Figure 2]



```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
JavaScript                      27            293             25           2131
-------------------------------------------------------------------------------
SUM:                            27            293             25           2131
-------------------------------------------------------------------------------
```

Figure 2: React Native: Total Line Count (Excluding Boilerplate)

Overall development experience: There was a decent learning gap that we had to overcome to get going. Fortunately, since React Native is built on React (Facebook uses it for the UI on their web site) we had some web development experience using HTML/CSS/Javascript and XML markup wasn't too hard to pick up.

Setup: The main issue we encountered was the initial setup. Although the steps seem simple on the "Getting Started" page, we encountered over thirty hours worth of initial errors, troubleshooting, and creating a toy app before we were able to start developing SafeRun. Since React Native's Android project is less mature than their iOS project, (Android released September 14, 2015, iOS released March 26, 2015) the overall lack of support for beginners caused us a lot of initial grief. On the flip side, development in iOS and integration with Xcode was fairly seamless.

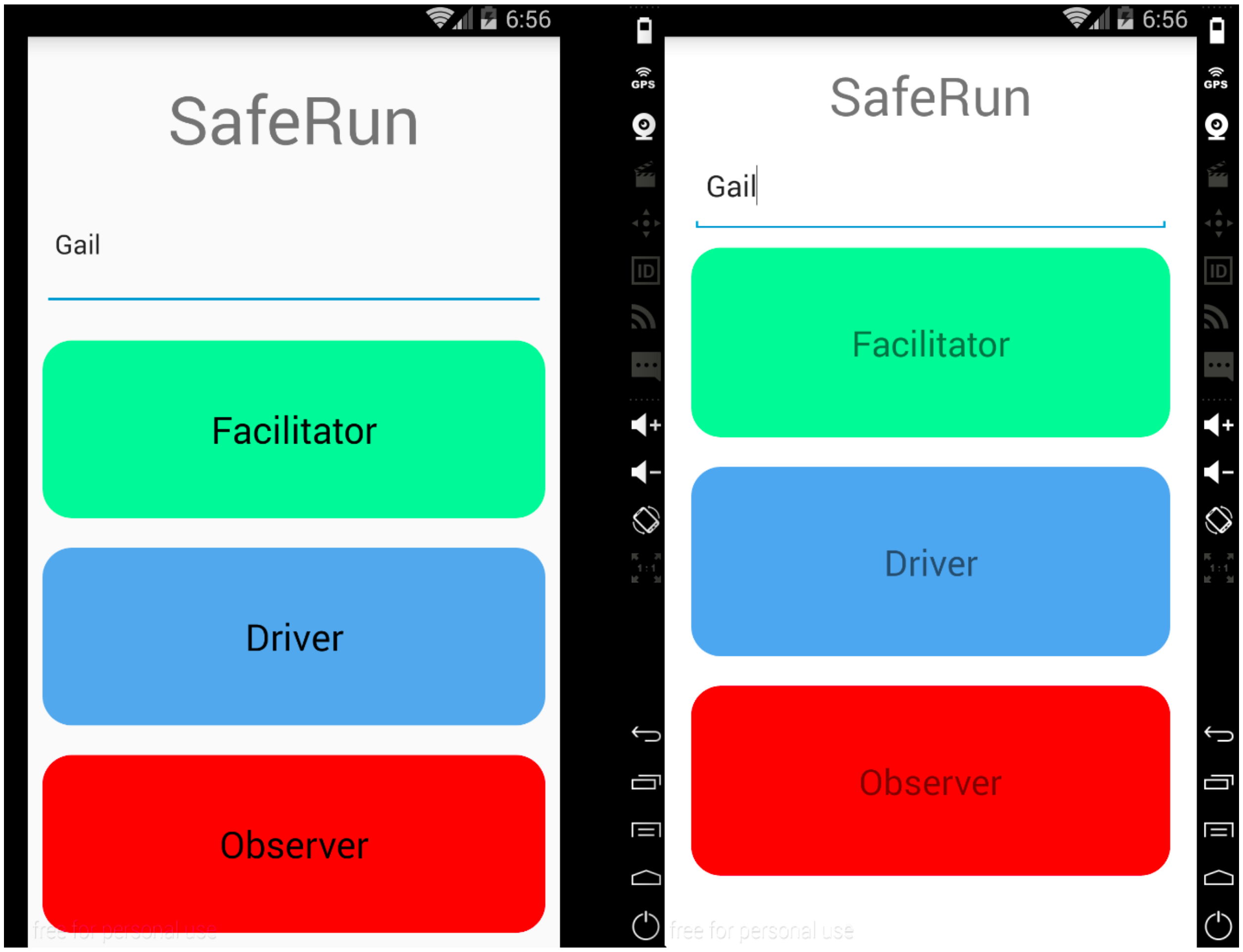
App Structure: As stated previously, the React paradigm is to include the logic, styling, and rendering tags all in one file (which can be defined as a React component). The idea is to create stateful components for use that are self-reliant (manage their own state internally). For example, our ‘Button’ component was used throughout the application and its state of rendering was only reliant on whether the user is actuating the button. Callback functions are passed in to set the behavior of the buttons. This helps UI designers modularize complex UI interactions with ease.

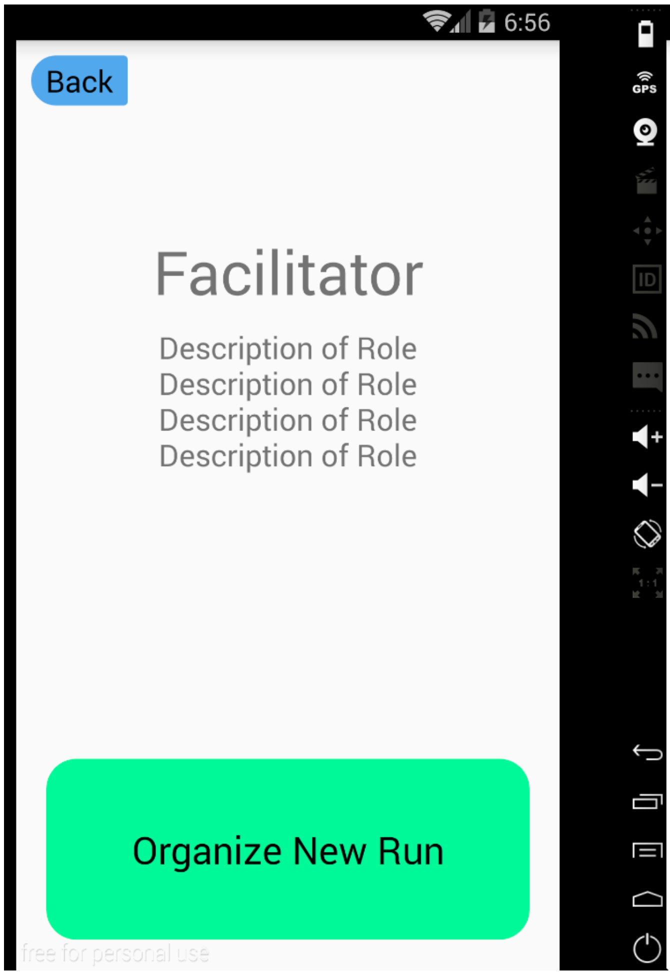
Challenges: Besides the initial setup for Android, the main challenge we encountered was the lack of support for beginners in general. Whether it was the limited scope of the Stackoverflow questions/answers or the lack of examples within the official React documentation, we struggled at times to do the simplest things. For example, the code to disable the hardware back button on Android phones was 37 lines, but it took us over several hours to dig up the information to make it work properly.

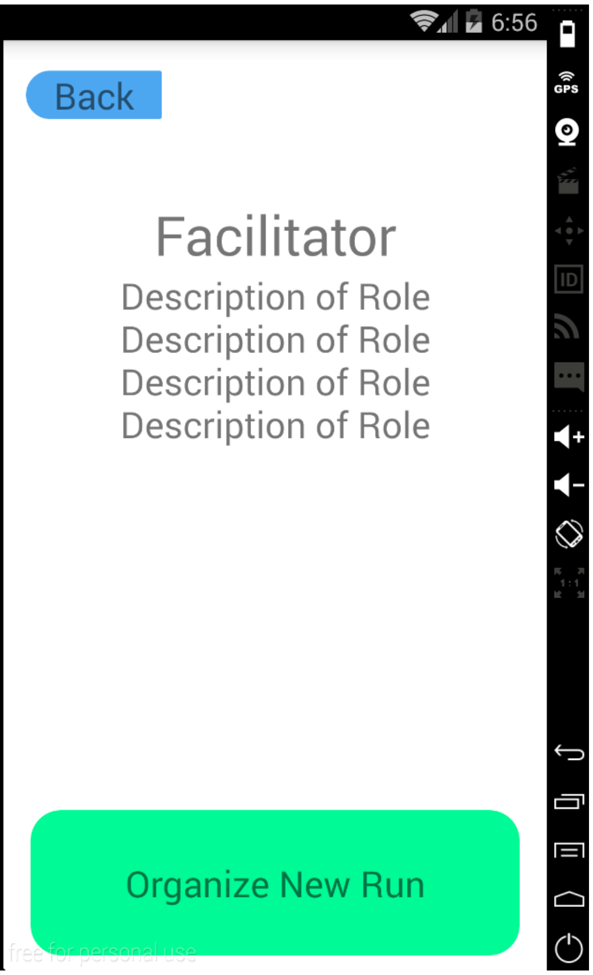
We also were not able to figure out how to switch workstations simply using a Git repository. Our current solution requires us to copy over all the files and perform an initial setup to get a React Native project up and going. Since React Native uses Node packages, we had to install/setup Node Package Manager as well.

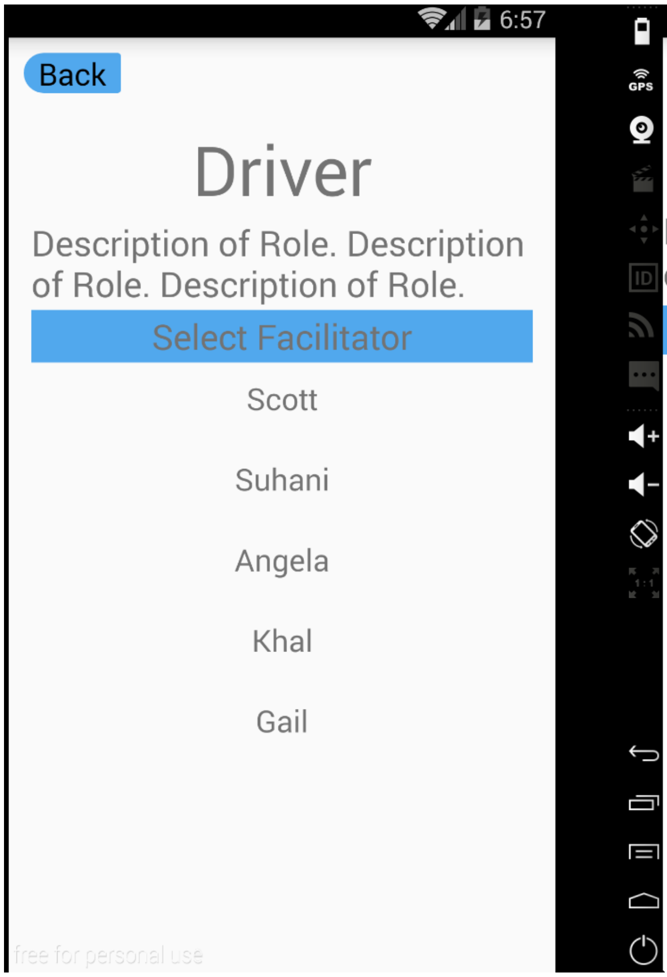
Conclusions: Despite the difficulties we faced, React Native has proven to be a powerful tool. For a little less code than the Android project, we are able to produce Android and iOS version of the SafeRun application without a hassle (given the development environment is setup already). Web developers familiar with React would be able to pick up React Native very quickly and develop cross-platform apps effortlessly.

<u>Snapshots of both applications (left side = **Android** / right side = **React Native**):</u>




6:56
Back
Facilitator
Description of Role
Description of Role
Description of Role
Description of Role
Organize New Run
free for personal use


6:56
Back
Facilitator
Description of Role
Description of Role
Description of Role
Description of Role
Organize New Run
free for personal use


6:57

Back

Driver

Description of Role. Description of Role. Description of Role.

Select Facilitator

Scott

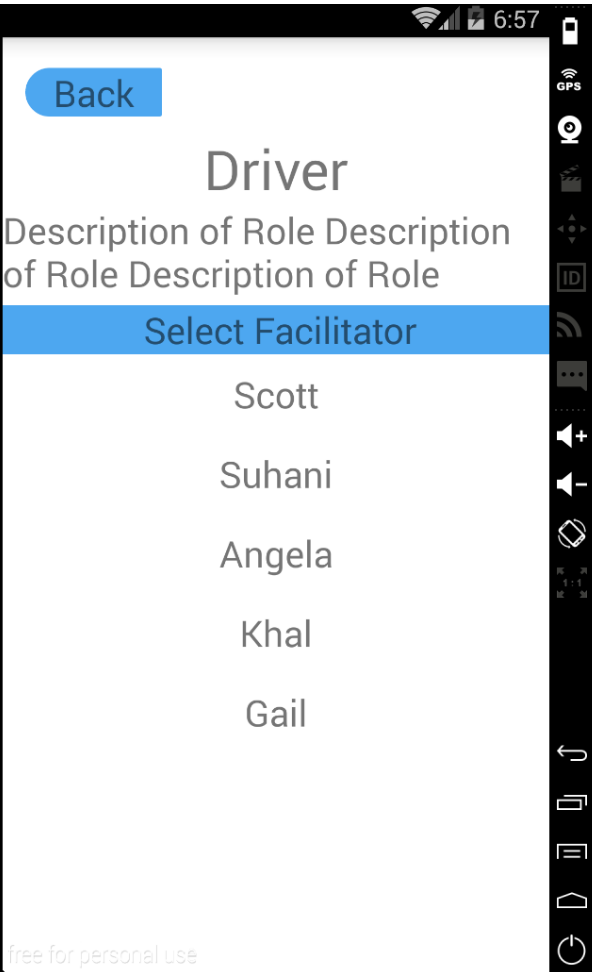
Suhani

Angela

Khal

Gail

free for personal use


6:57

Back

Driver

Description of Role Description of Role Description of Role

Select Facilitator

Scott

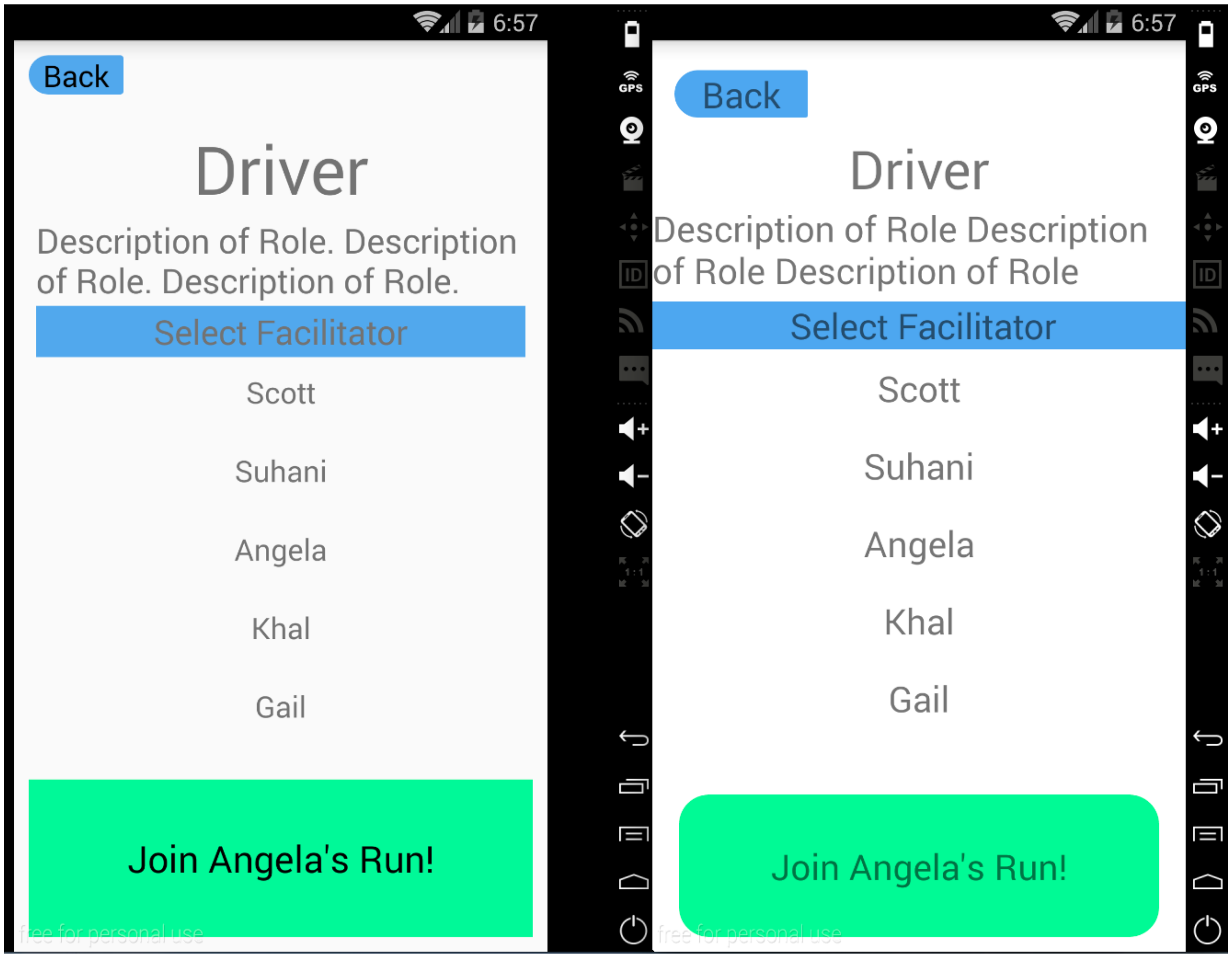
Suhani

Angela

Khal

Gail

free for personal use



6:57

Back

Driver

Description of Role. Description of Role. Description of Role.

Select Facilitator

Scott

Suhani

Angela

Khal

Gail

Join Angela's Run!

free for personal use

6:57

Back

Driver

Description of Role Description of Role Description of Role

Select Facilitator

Scott

Suhani

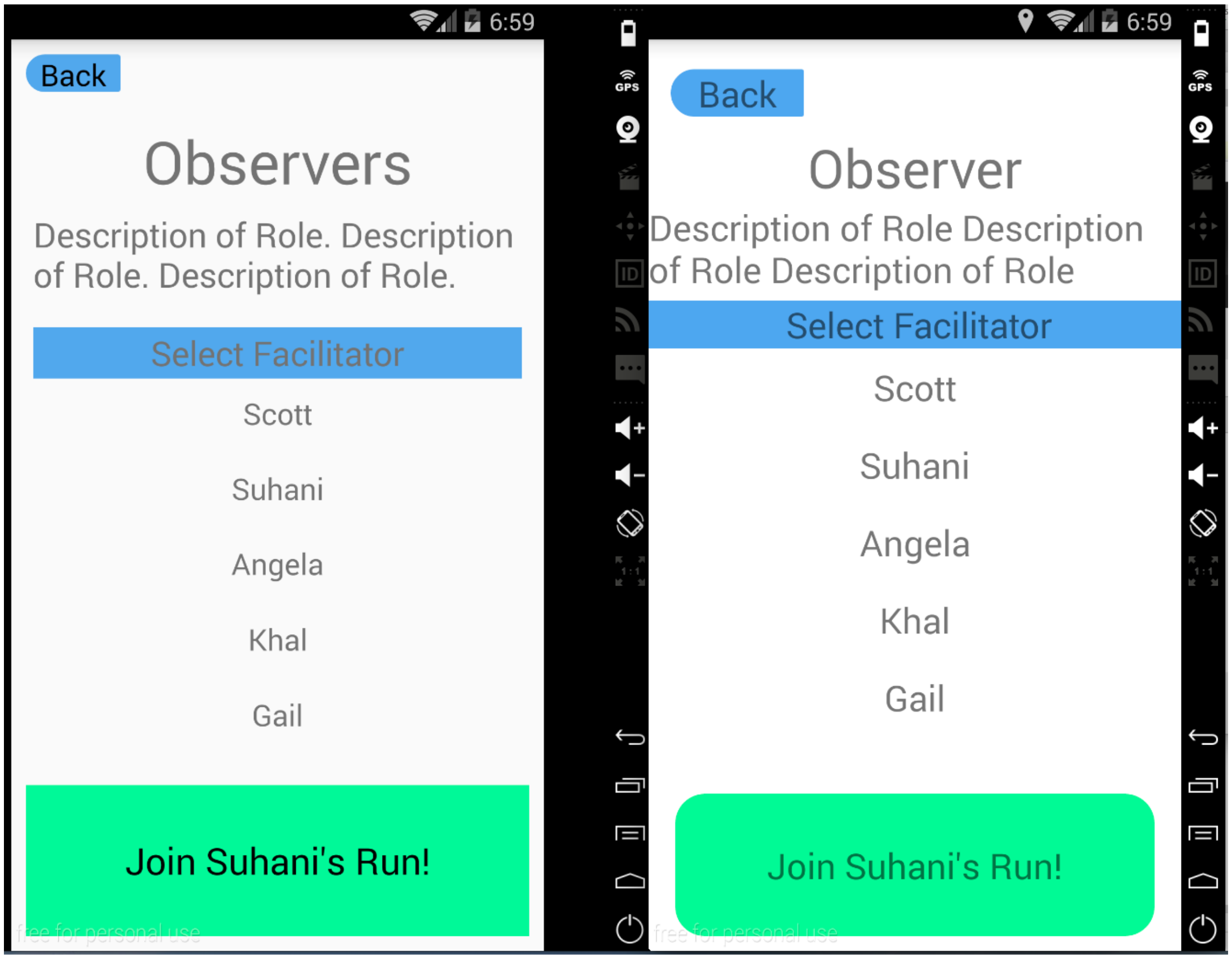
Angela

Khal

Gail

Join Angela's Run!

free for personal use



6:59

Back

Observers

Description of Role. Description of Role. Description of Role.

Select Facilitator

Scott

Suhani

Angela

Khal

Gail

Join Suhani's Run!

free for personal use

6:59

Back

Observer

Description of Role Description of Role Description of Role

Select Facilitator

Scott

Suhani

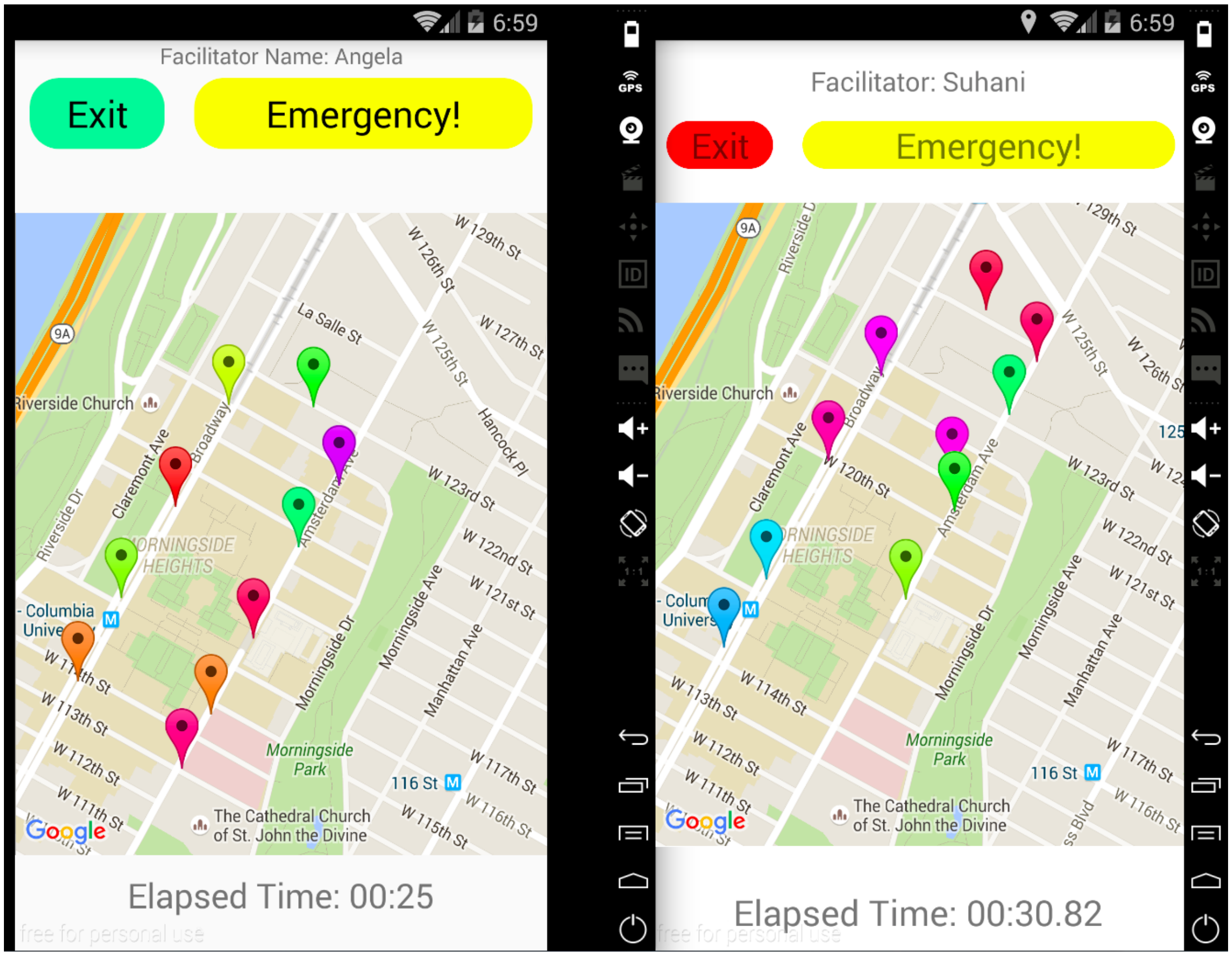
Angela

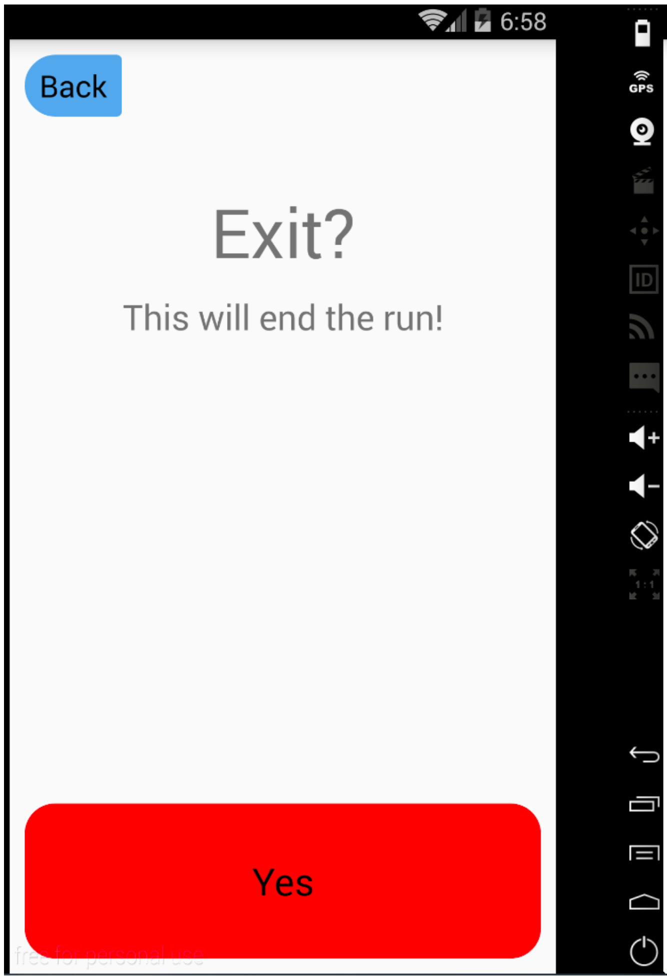
Khal

Gail

Join Suhani's Run!

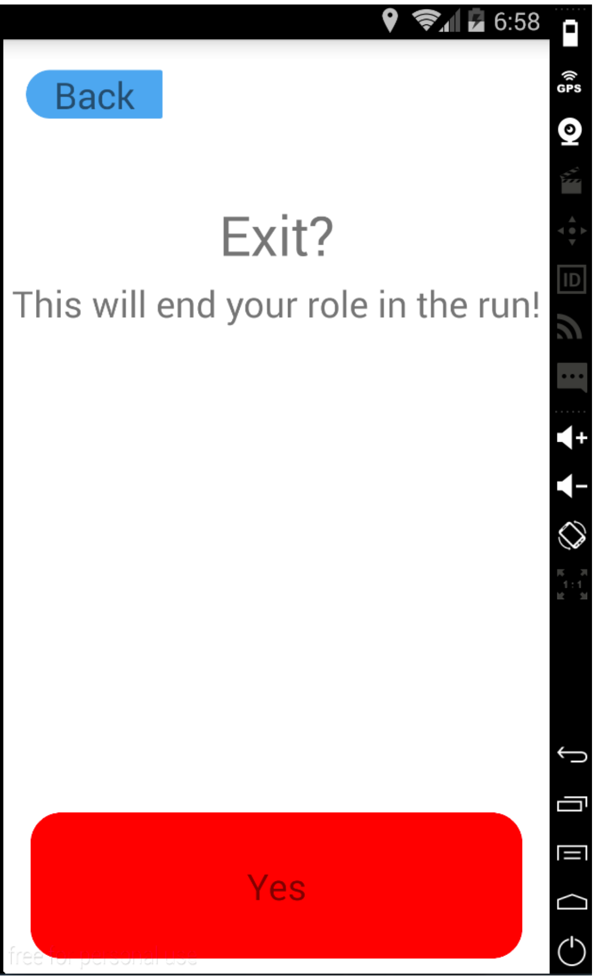
free for personal use


6:59
Facilitator Name: Angela
Exit
Emergency!
Elapsed Time: 00:25
free for personal use
6:59
Facilitator: Suhani
Exit
Emergency!
Elapsed Time: 00:30.82
free for personal use



Exit?

This will end the run!



Exit?

This will end your role in the run!

Comparison:

Metric	React Native	Android
Line Count	2131 lines	2428 lines
Development time (accounting for total hours of each member in the pair)	83 hours	38 hours
Stack Overflow questions (as of May 4th, 2016)	3,892 + 14,457 (React.js)	831,838
Speed of screen transitions (average of 10 transition measurements)	8 ms	6 ms
Initial load time (average of 10 load measurements)	2.5 secs	1.2 secs
Map load time (average of 10 load measurements)	1.8 secs	1.5 secs
Timed measurements conducted on Samsung Galaxy SIII OS: Android 4.4.4 Battery: Fully charged for all tests by connecting to external power source CPU: Quad-core 1.4 GHz Cortex-A9 GPU: Mali-400MP4 Resolution: 720 x 1280 pixels (multitouch) GPS and Location services fully enabled with Wi-Fi connected		

Survey:
Link: http://goo.gl/forms/HjBIXOnAjE

The survey we conducted included a total of 94 participants. Out of the 94 participants we had 15 programmers, 12 of whom had experience developing for mobile applications. We utilized the same mobile phone we used for the timed speed tests for consistency. Our surveys were conducted in the Columbia University Computer Science Lounge, the Carleton Lounge (in MUDD Building on Columbia University's campus), with random students walking around Columbia University's campus, random residents of the Upper West Side (in Scott's building), students in a non-technical class Scott is taking outside of Columbia, and with employees at DRN data, Inc. (Scott's business). We had a laptop with our custom Google Form ready for their anonymous feedback upon completion of using both applications.

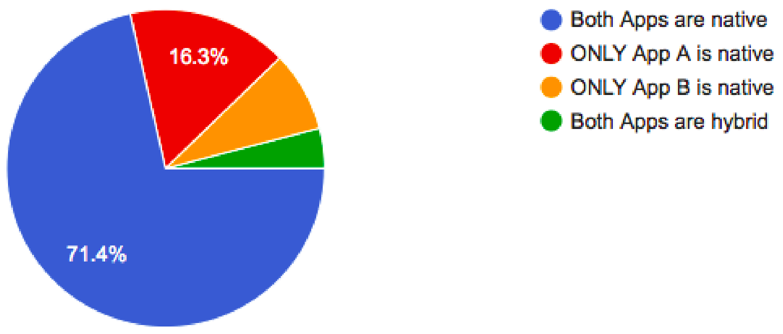
Most participants were able to identify the Android SafeRun as the faster app compared with the React Native SafeRun; however, most could not identify a distinct difference between the two with 71.4% stating "Both Apps are native".

Interestingly, when only analyzing self-identified mobile developers, 100% (12 participants) of them were able to correctly identify the native Android and the React Native versions of SafeRun with ease. Most cited the transition effect of the React Native as them main giveaway.

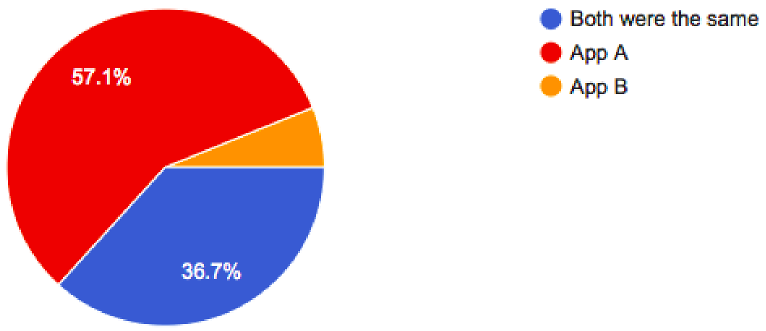
<u>Results:</u>

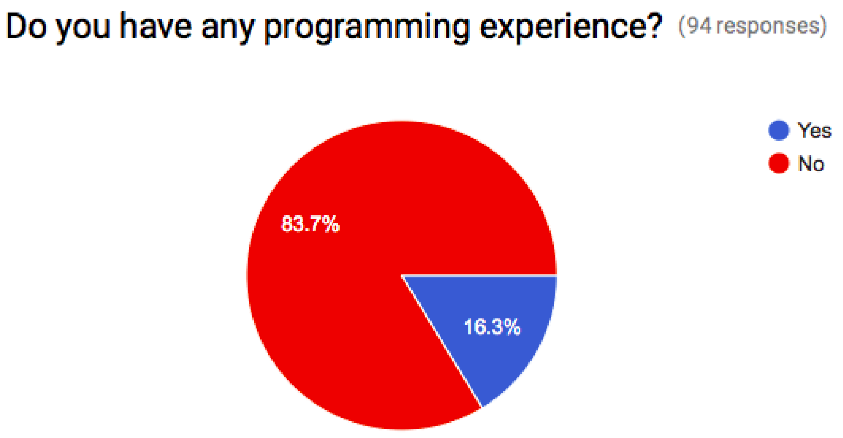
App A = Android SafeRun
App B = React Native SafeRun

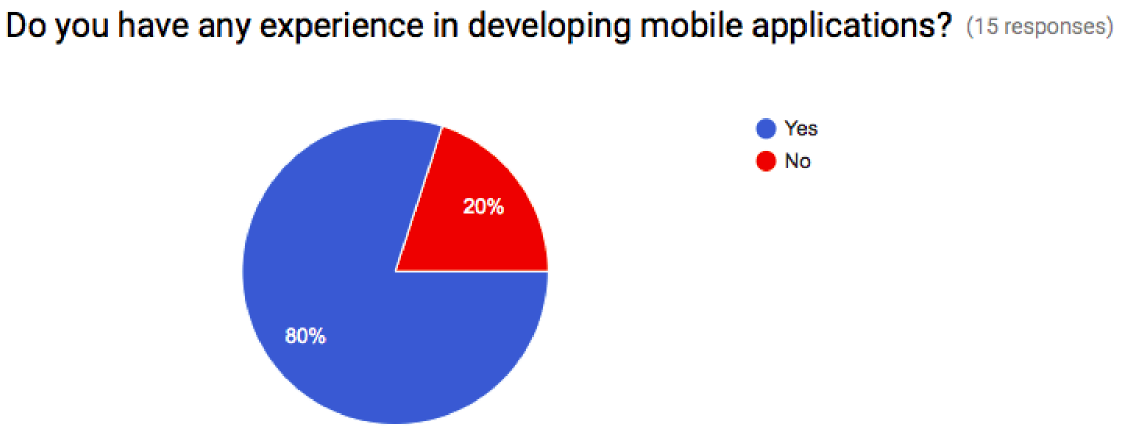
Are the Apps Native and/or Hybrid? Please ask the facilitator of this survey for clarification of these terms.
(94 responses)



Which App is faster? (94 responses)







Performance and Survey Conclusion:

Speed is still the main issue within the React Native platform. Between the speed tests and the surveys, it's clear that native Android development ran faster. However, the most significant contribution React Native has made to the cross-platform community is their ability to render native UI elements. The majority of survey respondents (71.4%) determined both apps were native.

Concluding Thoughts / Lessons Learned:

Angela: Developing the Android App for this project made me aware of how strong my developer bias can be. Because I have learned and become familiar with Android Studio and developing native Android apps, I instinctively prefered that coding environment and end result UI to the less familiar (but ultimately not too dissimilar) React Native development environment. This could be in part due to the fact that I am still relatively inexperienced as a developer, but I think my experience indicates that investing in documentation and tutorial resources pays huge dividends in the long run. If you can make your platform intuitive and accessible, beginner developers (such as myself) will capitalize on the resources available and become confident and comfortable with your tool, and favor it over others. Furthermore, if more people are developing mobile applications in Android Studio, more users will become accustomed to the native Android look and feel, and come to expect and prefer it.

Khaled: I'm of the opinion that the predictions made half a decade ago about HTML5 becoming powerful enough to replace native mobile development are wrong, cross-platform mobile development is alive and well because Google and Apple won't give up their App Store kingdoms any time soon. Although Facebook's React Native needs more work on speed, I'm impressed with the overall UI rendering ability. A big part of my motivation to work on this project stemmed from personal interest in developing mobile apps for multiple platforms without the work needed to maintain separate codebases. I've found a viable solution for my laziness.

Scott: I had the least amount of 'technical' computer science background in the group, especially in regards to mobile development. I really enjoyed putting the 4156 practices to work during the design iterations of our project. This was my first chance to use the tools learned in 4156 for a 'software engineering' project. I can now officially conclude that the methodologies we learned bring value not only to 'life's implementations' (as i researched in my paper), but also to software engineering. It was a nice complement to my case studies I am participating in for my research outside of the software engineering space, and I could actually experience the parallels. Getting the opportunity to pair program with Kal, I learned not only the fundamentals of ReactNative, but also Javascript, user interface design, as well as some basic Firebase concepts. I also found the group project experience rewarding by observing our group work towards a common goal, despite everyone having different skill levels and working as pairs in two separate groups. I was also very skeptical at the onset of our project that ReactNative could produce an app that looks, behaves, and uses native constructs. In conclusion, their platform has made it possible for one codebase to produce two projects that are very close to being native. Despite the small tradeoff of having a few native-specific elements, ReactNative achieves a valuable goal for companies and developers wishing to save time and costs by producing more output (although not completely perfectly) from less input.

Suhani: I believe that hybrid apps are good for startups to reach a larger userbase, while minimizing the cost of development. ReactNative did perform well, however the steep learning curve, and lack of easy-to-use components like Maps, makes me stick to Android development. It was interesting to see how simple Firebase was to use in both React and Android, despite being called from different languages. I found the performance and UI difference negligible, and I believe that the time and effort taken to learn ReactNative is the biggest challenge. Learning implementation differences in both languages made me appreciate android in some cases, and appreciate React in others, especially with respect to data binding. The fun part was comparing these two apps and seeing how they performed against each other.