****

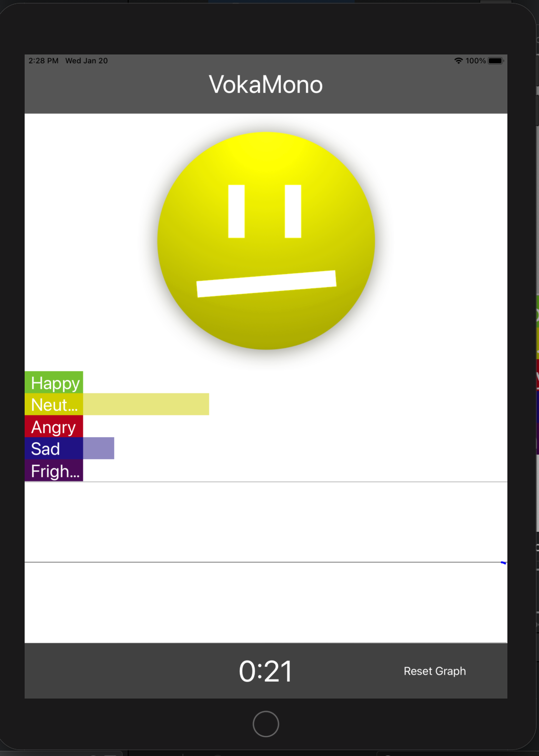
*We were asked in the brief to design a human-like voice interaction, we decided to have emotion detection to fulfill those needs.*

**Initial Prototyping**

Initially, I used Vokaturi which is an open source SDK that detects emotions based on the user’s voice. It is a multi-layer neural network that consists of 9 features, with an accuracy of 66.5%. It can detect 5 emotions: happiness, sadness, anger, fear and neutrality. Due to the lack of documentation and the complexity of the SDK, I worked on the provided demo application and built on top of it. The demo was able to detect emotions in real time, and give a score for each emotion.

As I began working on an early prototype, I encountered an issue with Android as the emotion detection wasn’t functioning in parallel with Android’s native speech recognition. A family member suggested that I try it on iOS, as maybe the bug would not be present on that platform.

Therefore, I installed XCode, which is Apple’s development tool, loaded up the iOS version of the demo and added a speech recognition module. Thankfully, they both worked simultaneously, allowing us to continue our work.

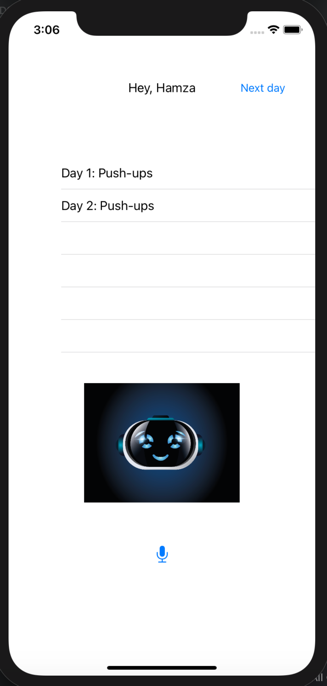
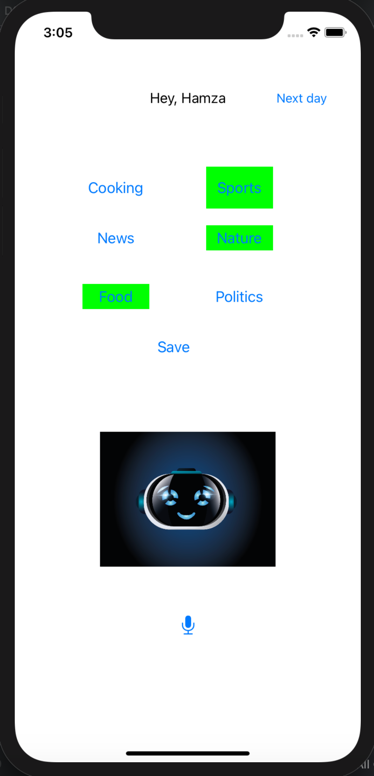
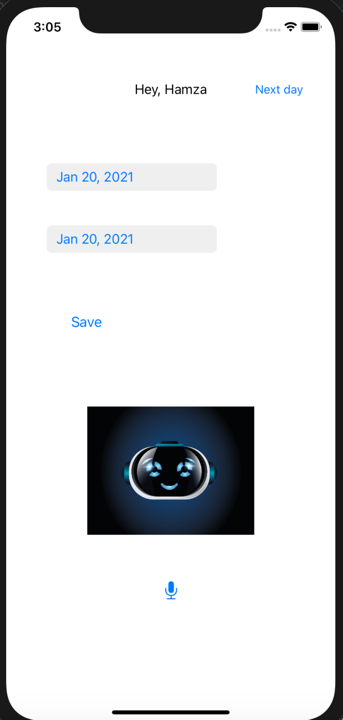
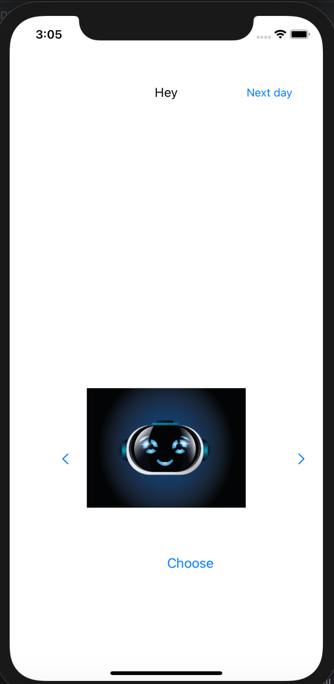
****

*Figure 1: Vokaturi demo detecting emotions in real time on iOS*

Eventually, I attempted to implement a simple conversation flow. It was made by asking the user a simple question like if they wanted to do a certain activity, and then analyzing the answer to guide the conversation. I used Swift’s own *contains(\_:)* function to determine whether the user’s speech input contained a specific keyword.

After that, I implemented the setup part of the app, where the user chooses his quarantine dates, interests and his preferred robot. Originally, as I was working on top of the provided demo by Vokaturi, I had to keep everything on one screen. So, what I did was I programmatically instantiated and displayed views, and when they were no longer needed I simply changed the *isHidden* attribute of the UI elements to true. This wasn’t ideal and poorly implemented in terms of memory management but due to the scale of the app it was impossible to tell.

As per requested by the team, I had originally also added a feature that consisted of showing a table of daily activities. This was removed in the final app because we preferred keeping the app as mostly a voice user interface and not an app with daily tasks and activities. We thought having the voice user interface as the main feature would match the requirements of the brief better. We replaced the calendar table with random activities based on the user’s interests in the final app.



*Figure 2: Prototype screenshots of the app setup*

**Changing emotion detection system**

At the beginning of the project, due to the fact that Vokaturi wasn’t working with speech recognition on Android, I had already researched alternatives to Vokaturi. There were no other vocal tone emotion detection frameworks that were free to use, so I kept in mind that we could use text-based sentiment analysis as an alternative.

I encountered several issues with using vocal tone analysis. The first one was that it detected fear too frequently, even when it was quite obvious that fear wasn’t the emotion in my voice. Another problem was that the emotion was detected in real-time, so if the user was expressing an emotion in his speech, if the last word or syllable expressed had an opposite emotion it would still output that last emotion. The last and most important issue was one that was noted by our conversation flow designer was that the user could be expressing an emotion in his words, but a different emotion could be conveyed in his voice tone.

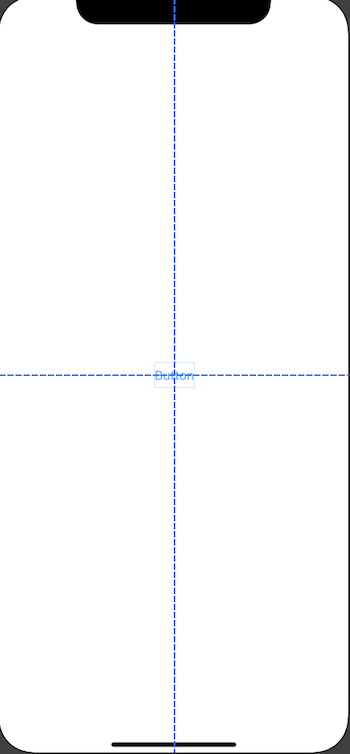
This was pointed out in one of our weekly meetings with our supervisor, and I suggested that we could use text-based emotion detection. We agreed to continue developing the app with text-based analysis, specifically using IBM Watson’s tone analyzer. This would give us a more natural conversation flow, as requested in the project brief.

The Tone analyzer was fairly simple to implement, as IBM provided rigorous documentation for our programming language Swift. When sending a tone analysis request, the framework would return all the emotions detected and their respective scores. The IBM Watson tone analyser detects emotion tones and also language tones, the latter being tentative, confident and analytical. These language tones weren’t useful in our case, so we then decided with the conversation flow designer to order the emotions in terms of importance. The classification of these emotions was done on their prevalence: the more common they were the lower they were classified.

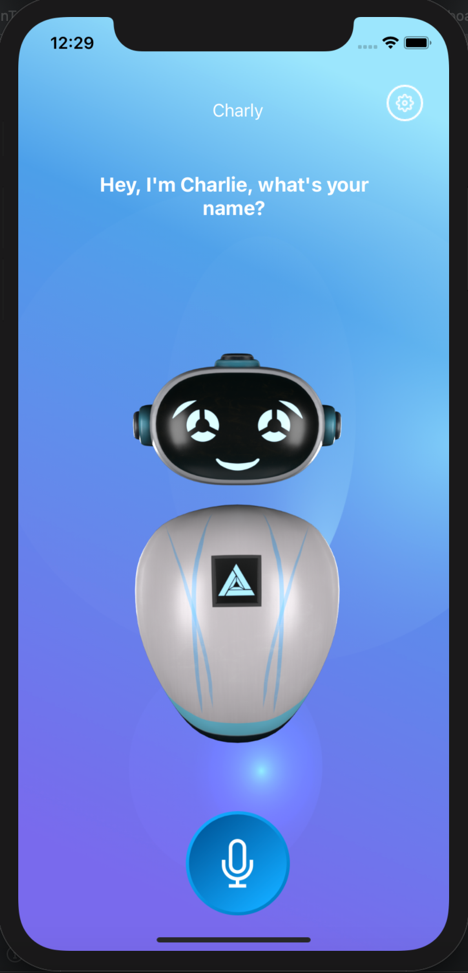
I was then able to completely implement the conversation flow, which was guided by both keyword presence and tone analysis. I decided to put the tone analysis into action only when more complex answers were expected from the user.

**Putting it all together**

The final app had the design based on what the designers had prototyped. Our design lead provided me with the UI assets (background and buttons) and the 3D modeler shared the 3D robot models with me. The screens were designed whilst taking into consideration Apple’s human interface guidelines. When adding certain elements to a screen, dotted lines would appear to guide the developer where these elements would fit best.



*Figure 3: Layout guidelines on XCode*

**

*Figure 4: Screenshot of the main screen from the app*

**Reflections**

Working on a group project was a very educational experience, it allowed me to work on my communication and put my skills into practice whilst working with team members from different backgrounds. Following a project brief from brain storming to conception gave me insight on how projects could be managed in a work environment. Having a supervisor and weekly team meetings made sure the assigned tasks were completed in time and on par with the expected quality. Finally, it is a project that I will proudly showcase in my portfolio.