

# **GNG2101 Deliverable F Report**

## **Eye Gaze Camera Cover**

Submitted by

[TEAM IDENTIFIER: B13, Team 13]

[Marwan Mashaly, 300228296]

[Brandon Yeung, 300250806]

[Wilt Moise, 300241883]

[Abd Ennour Souit, 300256956]

[Anna Brimacombe-Tanner, 300245481]

[Ian Mondoux, 300247598]

November 6<sup>th</sup>, 2022

University of Ottawa

# Table of Contents

- 1.Introduction.....1
- 2.Summary of Client Meeting 3.....1
- 3.Critical Assumptions.....2
  - 3.1Prototype 2: Learning from Previous Failures.....3
  - 3.2Prototype 2: New Design.....4
  - 3.3Prototype 2: Documentation.....4
- 4.Testing and Results.....5
- 5.Conclusion.....5

## Introduction

Through our client feedback and thorough analysis of our critical assumption points, our team has been able to learn the best way to optimize our software solution for our client. Through our software prototype testing, we can see exactly our points of improvement and what is working in our solution's favor. We continue to diversify our software infrastructure as we progress in our product.

## Summary of Client Meeting 3

Through the third client meeting, our team was able to discuss plenty of subjects that will help us improve our work on our project immensely. When we first began our meeting, we said our greetings and moved on to the main course of discussion. To start, our client ran through the basics of the Tobii Device which helped us realize how our ideas would fair while using the actual product. Furthermore, the ability to work on our program was enhanced as we talked more about the inner workings of the product and how the menus would function with our program. This conversation went on for plenty of time which allowed us to dive into our questions and concerns for the upcoming work to be done. After this, we talked about our progress and what changes we had to make to it over the course of its development. Luckily, our client was happy with our results and helped us formulate some more steps on how to progress our product further through the menus on the Tobii Device itself. The guidance we received throughout this section of our meeting easily helped the team to produce an even more successful plan of attack that agreed with our prior estimates. Our plans were finally appearing to become finalized but just to make sure we ended up scheduling another client meeting to test the implementation of our program to make sure it can fully function on our client's device.

## Critical Assumptions

The following are critical level assumptions made as we continue to progress in the development of cross-platform communication between the IOS and Windows applications, along with the communication between the Windows application and the Tobii device, to turn on and off the cameras.

This assumption's basis stems from whether or not both the Tobii Dynavox and IOS devices are running the same Wi-fi connectivity. Additionally, a similar assumption is whether the Wi-fi connection is stable. The principal method of communication between both Windows and IOS devices is through a google development platform called Firebase. This platform is the medium that will allow us to store information about the signal sent from both devices, and allow for inter-communication. Both platforms need to be connected to the same Wi-fi network for the Firebase to properly extrapolate all necessary data being sent from both applications. As such, we are assuming that under no circumstances will the client ever have two separate Wi-fi connections between his two devices since there is no feasible solution for such a case. Furthermore, the communication between Windows and IOS devices does not handle exceptions such as slow Wi-fi connectivity. We can't account for such behavior, so just like having a fallback for similar Wi-fi connectivity, our fallback for this solution will be along the lines of constantly sending signals back and forth until the Wi-fi manages to stabilize and receive the signal. This methodology requires further testing with the Tobii device, but ultimately, accounting for faulty Wi-fi is beyond the scope of our current abilities. Therefore, we are hoping such fallbacks are good enough countermeasures set for potential encounters, which include faulty Wi-fi connectivity, or not sharing the same Wi-fi connectivity between both devices.

This next assumption involves being able to communicate with the Tobii machine via the Windows application. The results of such communication would involve being able to turn on and off the cameras of the machine through the press of a button within the Windows application, which will be in turn communicating with an IOS application waiting to receive the necessary signals to initiate the said task. As stated previously in deliverable E, we can simulate tests between the IOS and Windows application, bearing fruitful results, however, when it comes to simulating tests between the Windows application and the Tobii Dynavox device, those remain inconclusive, since access to the Tobii Dynavox device is not provided. As we reach the later stages of product development, providing conclusive test results between the Windows application and the Tobii device to the client is not feasible. Not being able to test this integral part of the process, creates a massive assumption, and a false impression that we can complete such a task. As mentioned in deliverable E, it is possible to simulate this juncture by replicating the task with something similar in nature to the Tobii Dynavox device. As such, we will test on the webcam of a standard Windows device, and by doing so, this will give us an idea of the process that we can mimic over to the Tobii Dynavox device. However, as previously mentioned, these tests are non-conclusive and do not guarantee a 100% success rate if the same tests were carried onto the Tobii Dynavox machine. Ultimately, we are hoping to gain access to the Tobii Dynavox machine within the near future and finalize our testing.

Lastly, this assumption considers the unpredictable compatibility of our software with future versions of the Tobii device. We are unable to account for the compatibility of our software with future upgrades of the Tobii device since the development of this software is not corroborated or associated with the Tobii enterprise. As such, the current build for this software will only be compatible with the current version the client is utilizing. Furthermore, we are also assuming this software will still be compatible with newer software versions of the current Tobii device the client is using. Since the Tobii Dynavox device will be having consistent software updates to maintain the device's configuration, protection, and functionality, all these aspects can either slightly alter the functionality of our program or sabotage our program. As such, accounting for such an edge

case is beyond the scope of this project, therefore the user will need to keep the same device version for it to be compatible with our program, while we contemplate a potential solution for such an edge case, even if the outcome might bear no fruit.

## **Prototype 2: Learning from Previous Failures**

Following the completion of our initial prototype, many of our critical assumptions pertaining to the back-end framework had fallen short. Due to not being part of Apple's developer program meant that we were not allowed to tinker with the Bluetooth APIs already present on iOS. Although we were allowed to implement them within the actual code, building and publishing the final product would be impossible. This would mean that we would never be able to download the completed application onto our client's device. As a result, we were left to our own device's when it came to pairing the desktop equipped with the Tobii Device to our mobile application. We had initially considered alternative frameworks such as GameKit and CoreBluetooth. Unfortunately, we quickly found out that GameKit only enabled Bluetooth connectivity between iOS devices so the idea was quickly shutdown. The latter initially seemed promising, having support for Bluetooth Low Energy Protocols (BLE) between multiple platforms. However, we later found out that BLE did not provide a 'socket-like' paradigm, making it an ill fit for our use case.

Since we had exhausted all the options for Bluetooth that would allow us to complete the project within the deadlines, we decided to go back to the drawing table and come up with a different solution for enabling communication between the mobile application and the Tobii Device. Following our meeting, we concluded that Wi-fi would be our best option. Through the analysis of several metrics including the standardization of TCP/IP protocols, the idea proved itself to be far more promising than a makeshift Bluetooth API. Above all, Wi-fi would open a massive door for the project, the ability to make use of web services on both the desktop app and the mobile app.

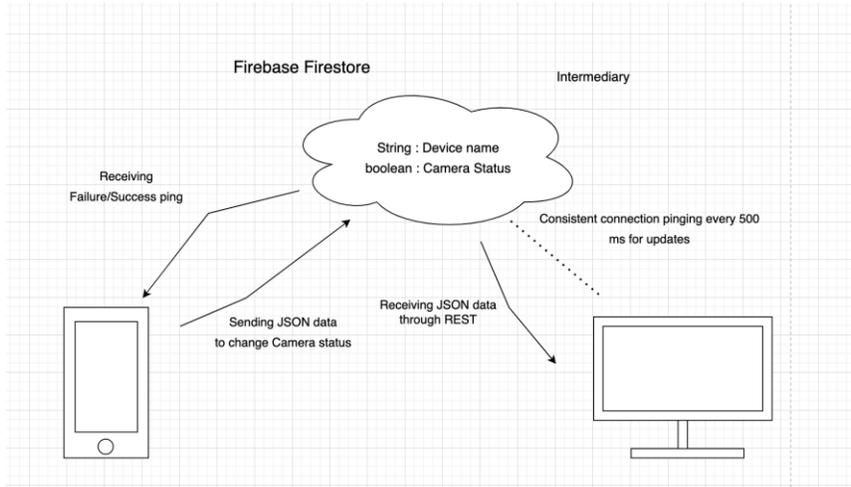
With this new design philosophy in mind, we had a specific web service in mind that could make the communication between devices a triviality. By introducing a third man, we would not only be able to have the state be actively stored, but we would also be able to send more complex information using REST APIs. The web service we had chosen to accomplish this feat was Firebase Firestore, the NoSQL hosting service for any application lead by Google.

## **Prototype 2: New Design**

The main difference between our current prototype and the previous lies within how the information is being transferred and how it is stored. As previously mentioned, our current prototype accomplishes communication between devices using Firebase. Rather than the Tobii device and the mobile application communicating directly, they instead communicate with a NoSQL document database called Firestore. We now host both the state of the device, and the currently connected device within a singular document within the database and both parties can modify and update it as it sees fit. This allows to not only pair both devices but for them to be in sync.

## **Prototype 2: Documentation**

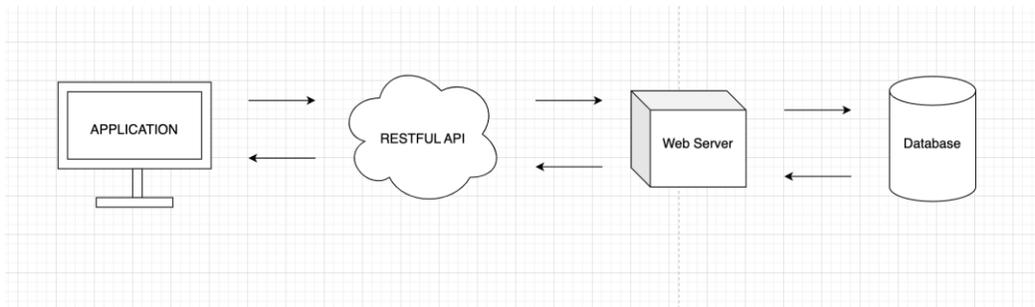
The chain of events of our solution has drastically changed with the introduction of a third party. It can be summarized in part by the following the graphic.



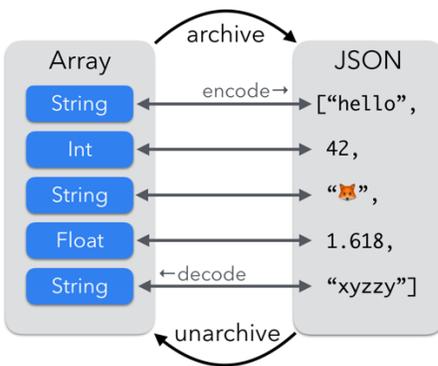
For starters, the chain of events is less of a chain and more of a cycle now. Rather than the mobile application triggering the flow of information, it is triggered the second either the desktop application or the mobile application is launched. Storing our data in Json format on the document held on the database, both devices are essentially pulling this document, reading this document by extracting and decoding the information received, perhaps updating it, and putting it back in place at a rapid pace. To limit the amount of traffic exercised on our database, we have decided to limit the pinging to every 500ms, resulting in slight delay between the mobile and desktop applications.

eyegazecameradb	Camera Status	rh4foKVGnKpCFzMnb7Ss
+ Start collection	+ Add document	+ Start collection
Camera Status >	rh4foKVGnKpCFzMnb7Ss >	+ Add field
		Connected Tobii Device: "Test Desktop"
		Enabled: false

For the moment, our database stores only two fields of data. The first is the device name as a string, and the second is a Boolean describing the state of the camera. The mobile application only reads and write to the latter field while the desktop application is capable of altering both. The Connected Tobii device field is set the minute the Tobii equipped desktop device is launched and is set to "Null" when the desktop app is closed or disconnected. This step was very important in letting the mobile application know when it has permission to write to the database and update the state of the camera.



This process is achieved using REST APIs. For starters, API is simply a mechanism which enables an application or service to access a resource within another application or service. Restful APIs can be made using just about any language if they follow 5 REST architectural constraints. Such constraints include: a uniform interface, client server decoupling, statelessness, cache ability, layered system architecture. The REST APIs communicate via http requests in order to perform our required database functions such as reading and writing.



The information is stored as JSON data but does not actually flow between devices in that form. Instead, there are several encoding and decoding procedures that occur throughout the sending and retrieval of data. Although our data is being stored as JSON on the database, it first needs to be converted to binary to be processed as an http requests. Once the data arrives to our applications, it then needs to be reconstructed as JSON and then decoded to the language specific data type. A very similar process occurs in reverse order when it comes to sending data from our two applications to the database.

```

let data = document.data()
var x : Bool? = data!["Enabled"] as! Bool?
if(x! == true){

```

Thanks to the Firebase web service, we mainly only need to handle the conversion of the data between JSON and native data types through casting.

## Testing and Results

Testing was a very important step in this prototype since we have completed the implementation of the connection between the two devices, and we wanted to ensure that it works seamlessly and without any hiccups that can deter the client from our solution. As shown in Table 1.0 we started by testing the metrics in addition to some metrics related to connection speed and strength.

**Table 1.0: Target Specifications with Expected Outcome**

#	Metric	Unit	Expected	Actual	Description
1	The amount of shake associated with the head mouse on the screen	mm	<3	<5	Due to taking longer times to turn off the eye gaze camera there is more shake than what we anticipated for
2	The amount of time taken to for the app to turn off the eye gaze camera	s	<1	<7	The amount of time taken to turn off the app is essential as this is what causes the shake with the mouse having a delay of 1 second would be ideal however our times have been a bit variable and can take up to 7s.
3	Client does not turn his head more than a certain angle	degrees	<15	0	Since we adopted a new solution, this metric no longer applies, and the user will not have to tilt his head to any degree.
4	Total cost	\$	<75	<10	Our current prototype can cost anywhere from 0 to 10 dollars since our costs are based on how long the user uses our product per day.
5	Amount of people needed to assist the client	People	0	0	The number of people need to help assist the client has to be one as the problem is that he cannot turn off the device by himself and needs 1 person to help, so he must do it himself.
6	Amount of time taken to connect to the database	ms	<300	<450	This metric depends on how strong the Wi-Fi connection is.  We tested it on our university Wi-Fi, and it was ranging from 600ms to 450ms but based on what our client said it should range around 300ms.

### Wrike snapshot

<https://www.wrike.com/frontend/ganttchart/index.html?snapshotId=Y5l81hoYRvxocBmRi2tqULN1AaXYnAb%7CIE2DSNZVHA2DELSTGIYA>

### Conclusion

Following our insightful third meeting with our client, we have continued to develop and refine our process in which we are creating our solution. Through our new programming test we have seen exactly what our critical points of improvement are and how we can further our technical program as we progress. We look forward to continuing to work with our client's needs to optimize the best solution possible.