2000 words approx.

**Design and Implementation (focus on user)**

* Give an **overview** of your **final** design and implementation, including any changes from your initial ideas.
* Provide justifications for any significant decisions you made. (why we changed something)

PLAN

Design stage >> implementation stage

**“any changes from your initial ideas & why you changed them”**

PLAN – DESIGN STAGE

* Android app & raspberry pi tracker (Bluetooth and cellular)
* Tracker is thin enough to be attached to very small items (keys & wallets)
* Accuracy at short range, infinite range -hybrid approach
* Serverless, the tracker is a server with a direct connection to user app
* App was supposed to adapt to user (explain why we could not do this)
* future developments- because tracker is adaptive, it can be used in industry asset tracking. From livestock tracking, shipping and securing the location of high value items (computers in a lab)

PLAN – IMPLEMENTATION (EXPLAIN EVERYTHING for each step)

* Switch from android to ios & android app
* Assemble prototype physical tracker at smallest size possible, reality check, cannot make tracker smaller without manufacturers and funding
* Working prototype can only be used with baggage or bag
* Bluetooth is more complicated than was estimated, we could not get estimote, native Bluetooth on user or puck.js to work with the tracker. This is a similar difficulty with other developers. (nothing new)
* Bluetooth in practice, when used to pair with a phone is not reliable and would fail all the use cases. We did the required testing to come to this conclusion & Bluetooth feature was dropped temporally.
* Short-range accuracy not currently possible with available resources, feature needs funding and expertise, resources which we did not have.
* Hologram nova worked, but due to the service requiring the hologram service to negotiate with many phone carriers, their cloud servers were required, we could not run a server on the raspberry pi, we needed a server middleman to push notifications to the user. **Actually it's completely alright to run a server on it, but why it's not suitable for our case?**
* We first tried the igor servers, but these servers are not worldwide neither scalable
* We then decided to use the amazon s3 servers, scalable, reliable, worldwide and cheap (list countries its available in)
* ~~Long range function provided by the hologram nova fits use but is not accurate enough for short range~~
* Ios app is now able to send http requests to the tracker , and the tracker can send its data to the amazon s3 server, which the ios app accesses.
* Using the unix philosophy all the tracker functions were made using simple bash scripts, testing shows reliability, we used crontab, grep and made a daemon. Initially python- (but unnecessary complication)
* iOS & Android app implementation process (UI ..)
* ~~Android app backup (we had plan b) process~~
* Tracker requires case, first iteration is wooden block
* Working on slim 3d printed plastic case

**Design Stage**

Our final design for the iLost tracker consists of an Android app and a Raspberry pi with the attached Hologram Nova. The User interacts with the Android app, whereas the raspberry pi provides the short-range capability of the Bluetooth and the Hologram Nova cellular modem attached to the raspberry pi provides the long-range capability.

Short range is defined as less than 30 metres from the target and long-range is defined as above 30 metres to 400 km and beyond.

[insert picture of tracker concept at January ]

This Hybrid approach, compensates for the weaknesses of the Bluetooth range and the weaknesses of accuracy of the Hologram Cellular modem, by leveraging the short-range accuracy of the Bluetooth and extremely long range of the cellular modem. (there are cell towers in nearly every part of the world)

In our final design, considering the strong demand from users to have the tracker as small as possible (similar to the Tile Mate). We endeavoured to ensure the tracker is compact as possible.

We also initially designed the Tracker to be serverless. In this way, the tracker would communicate and share information directly with users iLost app.

Moreover, we set out to have tracker adapt to the users need. The iLost tracker should be able to be smart enough to understand that a user has left behind an item regardless of the item type (luggage, handbag, laptop etc..)

However, our design plan changed rapidly as we started implementing the tracker and as we consulted with more users and professionals (both in marketing and in engineering).

[image of Tile Mate]

At the computer fair, in which we consulted with various industry professionals, we realised that we should focus on a minimum viable product for duration of this project. Certain product features such as having the tracker as physically small as a sticker (similar to the Tile Mate) is not possible in the scope of the project as we need specialist manufacturers, engineers and funding (we already looked at all the possible routes making the tracker smaller). Most users clearly want a tracker that is small as the Tile Mate, so we looked at next viable use case and the minimum physical size of the tracker that can fit that viable use case, this was baggage use case.

It is feasible in the time frame and available resources in this project, to build a physical tracker that is small enough (insert dimensions here) to fit in the user’s handbag, rucksack and travel luggage (the baggage use case). While also available to fulfil the user criteria in terms of accurate, long range tracking (over 30 metres)

At the design stage, we initially designed the tracker to be serverless i.e. the physical tracker solely communicates via cellular data to the app on the users phone, reducing cost and increasing security for the user. However, for the cellular modem to function properly, it must communicate with Hologram Nova cloud servers, this is so that the Hologram can negotiate with multiple phone carriers, this functionality allows the user with one sim card to able to have a tracker that can triangulate and transmit data anywhere in the world which has cell towers.

Currently the tracker communicates with the Hologram Cloud servers, which in turn communicates with the user’s app.

Also, in the design stage we realised that our tracker can be developed into an enterprise framework. If our product is successful at item tracking for consumers and is fully tried and tested, this same technology can be used for enterprise applications, such as asset tracking, shipment tracking, even livestock tracking. However, this is not in the scope of this project.

**Implementation Stage**

We initially designed the user app to be hosted on the Android platform, instead we implemented the user app on both the iOS and the Android platform. We did this for two reasons. Firstly, from market research, roughly 50% of users do not have an Android smartphone. Secondly, developing the Android app allows us to have a plan B, in case we cannot develop the iOS app in time or if there are compatibility issues with the tracker. The Android and iOS app both have different programming languages and paradigms.

The reason why we developed an Android/iOS app instead of a web app, is that as part of the iLost design, the user needs notifications in real-time (when an item has been left behind). Real-time notifications are not currently possible with web apps.

Another significant change to the iLost tracker was the discarding of the Bluetooth function. Initially, the iLost tracker was supposed to use the Generic Bluetooth adapter on the Raspberry pi and the user’s smartphone.

[insert diagram for Bluetooth short range function – explaining paragraph below]

The Bluetooth function is needed for the tracker at short range (less than 30 metres) to remind the user if they have left an item behind, if an individual recently just stole their item or for tracking down the location of an item (to a range +- 5 metres). This is after the long-range function (i.e. the cellular modem) locates a rough location of the item (30-40 metres from actual location)

In a situation whereby, the user is greater than 30 metres away from the physical tracker but requires a more precise location of the tracker, the Bluetooth function was supposed to fulfil this use case. My measuring Bluetooth signal strength coming out of the tracker, (2.4 to 2.485 GHz radio frequencies) the user can determine how near they are to the tracker and thereby their lost/stolen item.

However, although the Bluetooth concept is simple, as many developers (we talked with one personally in the department) implementing reliable Bluetooth functionality is problematic (looking at most Bluetooth products on the market, many products have connectivity problems)

The main disadvantage that led us to cancelling the Bluetooth function, is that in order to support the iOS platform for the users, the iLost tracker must comply with [MFi](https://developer.apple.com/programs/mfi/) (Made For iPhone) program. The MFi program restricts the types of devices that can connect to Apple products. The iLost tracker is a Raspberry pi with an attached cellular modem, the Raspberry pi is not certified under the MFi program. This means the iLost tracker Bluetooth function (it’s a Raspberry pi zero, which has a Bluetooth module by default) cannot interact with the user app and we cannot release the iLost tracker for the iOS platform.

Subsequently, we tried to implement the iLost Bluetooth feature with the Android app. Although, the android platform did not have a compliance program like the iOS, there were other serious issues. We simply could not find the functionality to be able measure Bluetooth signal strength emitting from the iLost tracker. We assumed that the mac address of the iLost tracker would always be the same, but this is not correct. After Bluetooth 4.2 was released, the mac addresses of any Bluetooth device is constantly randomised. Because the mac addresses (we need this address to communicate with the iLost tracker) is randomised we cannot from android app side be certain which Bluetooth device is the iLost tracker.

We also looked at other Bluetooth products on the market, such as most headphones. They were also plagued with stability issues (such as signal not receiving or pairing breaking), if we were to implement the Bluetooth feature we would need experienced developers who specialise in implementing reliable Bluetooth communications between user apps and physical computing devices. Unfortunately, there is also no library or service (like Hologram) which simplifies the Bluetooth stack.

[insert picture of raspberry pi 3 model B with hologram nova attached]

Our next step, was to implement the Cellular functionality of the iLost tracker. The cellular functionality consists of a cellular modem (called Hologram Nova) provided by the company Hologram. The Hologram Nova, via USB is attached to the raspberry pi (for testing and early prototyping we used the Raspberry pi 3 model B) and this what we refer to as the iLost tracker. We manged to establish a protocol to communicate with the iLost tracker.

This protocol consists of sending HTTP requests to the tracker, the tracker then responds via HTTP post with the data back to the sender.

At first, we sent HTTP requests via our computers (using the Postman application) to the iLost tracker, we received responses that showed that the communication method functioned.

After, we checked that the cellular modem method of location worked and that it could return useful data. The Hologram Nova uses nearby cell towers to triangulate its location. It did work, but the accuracy was not satisfactory, on average location was 40 metres away from the bag we were tracking.

Even though we could send HTTP requests to the tracker as well as sending simple worded messages (such as “RL” for request location), we could not request the tracker to send its location back to the sender.

So, we needed to solve this issue. At first, we tried to set up an apache web server on the raspberry pi, open a specified port on the server and send messages to that port. We built a python script that would reply to any specified message we sent to the server. This did not work, any messages we sent were not received by the server. What we didn’t realise was that when we were sending a message to the tracker, the payload was being sent to the Hologram Cloud servers, these cloud servers would then forward the payload directly to the tracker.

We could not find a way around the Hologram cloud servers, in fact the Hologram cloud servers were essential. They reason why their cloud servers are needed is that in order for us to communicate with the Hologram nova (cellular modem) and be able to triangulate the location of the tracker (i.e. Raspberry pi), the Hologram service as a whole needs to negotiate with multiple mobile network carriers. Being able to triangulate location using cell towers and communicate with a cellular modem internationally at a regular price (it currently costs us $1 a month) is only possible currently using the Hologram service. Previously developers had overcome many administrative, legal and economic issue in order to work with cellular networks.

Through all our testing and use the Cloud servers did not fail and reliably delivered any payloads we sent to the tracker. Without the Apache server, we then used Hologram SDK (which includes a Command Line interface) to open a server at port 4010 (on the Raspberry pi), any messages we sent via HTTP were now received (we could view the message on a terminal) we then used Bash scripts to automate useful responses. For example, if the message “RL” was sent (Request Location) then the tracker via the scripts would triangulate its location, store this data (longitude, latitude, altitude) as string data and send this back to the sender. Scripts were also created for the tracker to shut itself down, power itself up, disconnect from cellular networks (use cases such as the user traveling on an airline)

We also tried to host a server on Igor instead of the raspberry pi, however we found a more efficient solution, this was to leverage the Hologram Cloud servers. The Igor servers are not available worldwide and offered limited scalability.

The reason why Bash scripts were used instead of python is that there is native support for using CLI commands provided by the Hologram SDK. Another advantage of using Bash scripts were that we were able turn these scripts into daemons. Daemons are process that run without user supervision and in the background on Unix/Unix-like operating systems. The cron (specifically crontab) program is used and whenever the tracker is turned on, all of the Bash scripts that enable tracker functionality run as background processes, lowering the power consumption of the tracker.

Using a Bash script, and using inbuilt Linux programs such as grep, cron, is line with the Unix philosophy of small programs which do one specific function efficiently, being chained together to build a larger program.

Even though we could request the tracker to send its location back successfully, we still needed to be able to store this data, so the user app could access it and use it. We researched into possibility of running a small server on the user app, which could receive this data. This is not possible; any server must operate outside of iOS and Android platform.

We then decided to use the amazon s3 servers which are scalable (servers can be replicated or moved to any AWS region), reliable (good track record, enterprise grade), available worldwide and inexpensive. The tracker now sends location data to our s3 server. We were then able to have the user app request location data from the s3 server and display this data/their location of the tracker to the user.

We also encrypt this data using AES-256 servers side encryption keys, each record is encrypted with a unique key, even the key itself is regularly encrypted. This service is provided by amazon.

[insert diagram of current stack: user app – s3 server – Hologram cloud – tracker]

Next, we needed to create the UI for the iOS app and Android app. We approached this by keeping everything as minimal and simple as possible for the user. The user would take a picture of their item they want to track, assign a category for it. When the user wants the location of their bag for example they would tap on the on the “[locate]” button, a google map of their item would show along with any information such as directions and distance.

[insert screenshots of iOS and android app]

For our tracker to be useful and satisfy its use cases it had to be portable. When we were testing our tracker, which was a raspberry pi 3 model b, it had be plugged into the mains. We replaced the raspberry pi 3 model b with the raspberry pi zero. The raspberry pi zero is 2x smaller than the raspberry pi 3, uses much less power and still uses the same operating system (so all the Bash scripts and configurations still worked)

We first tried to use small power banks to power the tracker (normally used for mobile phones) to test the average power consumption of the tracker. The tracker requires 130 mA. Using power banks is not practical for the user, so instead we tested lithium-ion batteries, these were slim and provided more power… [not finished yet]

Moreover, a physical case was needed for the tracker which would satisfy the user use cases. The case needed to be durable, simple and functional enough to be attached to a bag. A prototype was created using MDF wood, that was interlocking. [not finished yet]

[insert diagram of proposed case]

[insert image of prototype case]

[insert image of final case]