Goldsmiths, University of London

Software project final report

iLost

Ahmed, Muhammad
Chowdhury, Thairan
Davies Minta, Dylan
Fakrul, Mahmudul
Farkhani, Hussein
Jheng-Hao, Lin
Pecorella, Mariano

supervised by
Tim Blackwell

April 20, 2018

Contents

1 Introduction

Losing belongings is a common problem, roughly 70 worth of items are lost in the public per year. [1] We conducted a survey about lost items to find out if we could create a useful application based on this concern. Following our survey, this statement was found to relate to university students as 89% of our participants claimed to be susceptible to losing items. This information motivated us to reduce the loss of personal belongings; leading to the development of an item tracking application and a portable tracking device. Our app aims to circumvent the loss of bags and valuables. The user will attach the portable tracker to their bag, allowing it to be tracked using our app. The app will notify the user when the distance between the app and tracker reaches 30m.

This report will cover five main categories:

- Development Record We explain the methodology used by our group to decide the technology with which to implement the apps functionality alongside the development methods that were used. We also describe the way that the development was handled by the team and provide a reflection on its efficiency.
- Formative Evaluation This section of the report describes our assessment with users during development. The formative evaluation details what we did during user testing and its outcomes.
- Design and Implementation An overview of our final design and implementation, including any changes from our initial ideas with justifications for the significant decisions we made.
- Quality Assurance We detail our approach to Quality Assurance and the testing carried out, including our results. An assessment of how well our final system imitates our initial requirements with justifications of any changes we made to the requirements.
- Summative Evaluation A descriptive evaluation of the methods, results and conclusions of our final software.

2 Development Record

2.1 Teams

During the implementation stage, we set up an organisation **GSoft** on GitHub and we were divided into three teams, **iOS app**, **Android app** and **Backend/Tracker team**, which shows in figure 1. Each team was grouped by 2 to 3 people who were more interested in that topic or technology. Some of us were interested in more than two areas, if this was the case; member could join more then one team. The list of the team members can be found in table 1.

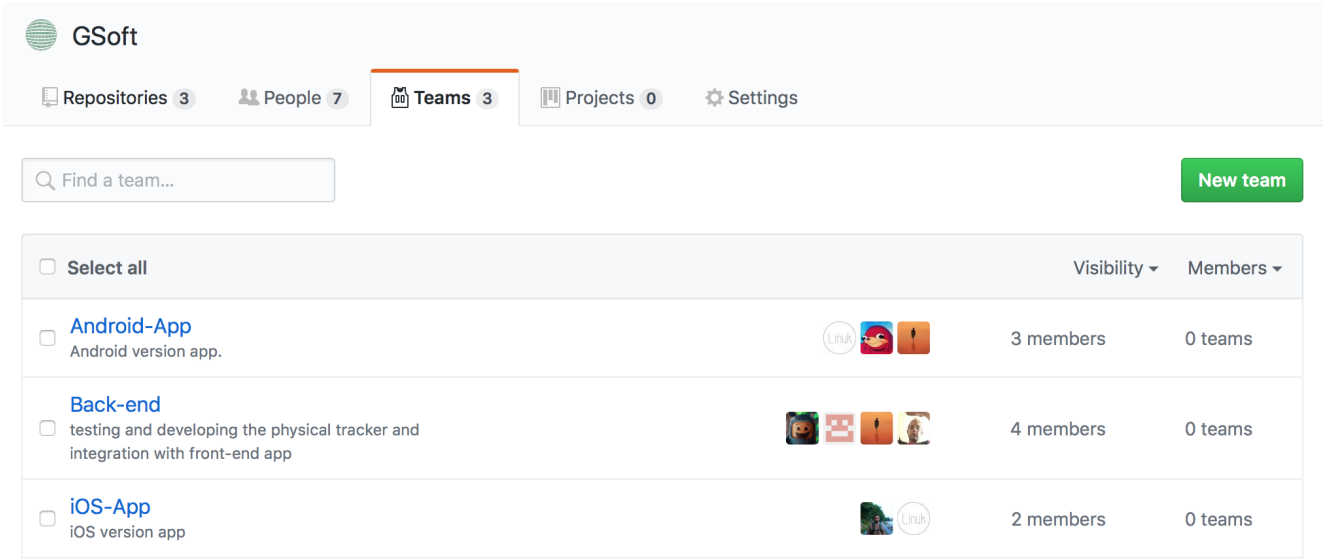


Figure 1: Development Teams on Github

Team	Members
iOS app	Jheng-Hao(leader), Muhammad
Android app	Dyland(leader), Mahmudul, Jheng-Hao
Backend/Tracker	Hussein(leader), Thairan, Mahmudul, Mariano

Table 1: Teams

2.2 Technology Selection

Development Teams: Each team was responsible for how and what technology to use, as long as the production could be made on time. We believed this methodology had these advantages in terms of the limited development time:

- React Agilely: Compare to have a poll with all members of our group, it was agiler to come to the decision within 2 or 3 people within one team. Since each team could react to the situation and resolve the issues in a more efficient way.
- Specialities differ: Our group was divided by our interests and specialities, we trusted each team could make the best decision for the whole team with their research and experience. For example, the iOS app team would not interfere how the tracker team implemented the physical components at all, and how the Android app was implemented would not be the backend/tracker team's concern. All teams were trusted in which they would make the best decisions.

Even though the team was separated, it was important to keep everyone on the same page. So each technological decision or propose was documented as an architecture decision record(ADR), which enabled us to have an overview of all the technological changes. According to Michael Nygard, each ADR contains five columns[1]:

- Title: A brief description of the propose.

- Context: **Why** do we need to make the decision or change.
- Decision: **What** is the response toward the decision.
- Status: current status of the decision which, should be **accepted**, **rejected** or **deprecated**.
- Consequences: **What** become better or worse because of the decision.

Please find more, complete architecture decision records in A.2.

To make sure that the whole group had the latest information; We used several channels on Slack to keep everyone updated. Such as the **iOS channel** was the place containing all updates related to the iOS application development, which shows in figure 2.

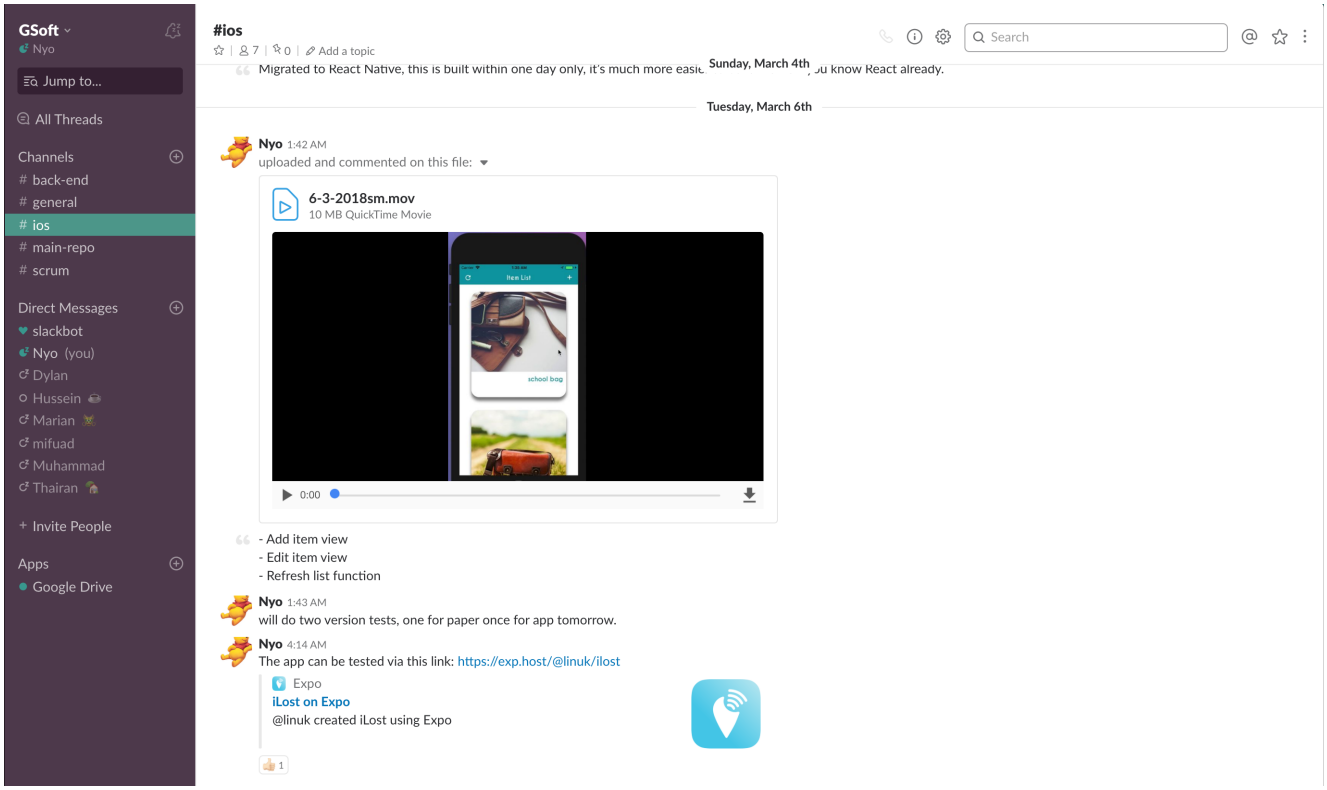


Figure 2: Slack iOS Channel

2.3 Agile Development

Our team tried out two agile development methodologies, **Scrum** and **Kanban**.

2.3.1 Scrum Development

Since we could not actually devote all of our time to develop the application, it would not be reasonable to do the daily scrum and have a meeting day-to-day for all of us. So in the beginning, we tried to set up a routine to simulate the daily scrum with Slack.

We defined 3 days as one sprint and whenever a sprint was finished, the team needed to answer three questions in the Scrum channel on Slack:

- What did I complete last time that contributed to the team meeting our sprint goal?

- What do I plan to complete this time to contribute to the team meeting our sprint goal?
- Do I see any impediment that could prevent the team or me from meeting our sprint goal?

Unfortunately, not every team could follow the sprint process and make any progress every three days. Sometimes a team might work for a sprint then stopped for two sprints since other modules might have a deadline or coursework. Hence, this scrum process was not actually conducted properly and stopped after few weeks after started. The example records show in figure 3.

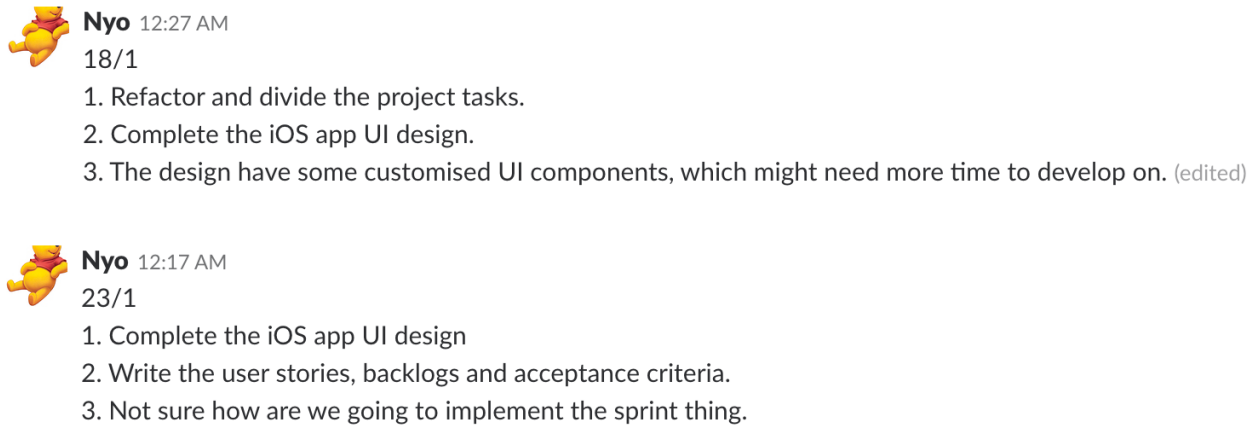


Figure 3: Slack Scrum Channel

At the middle of the development stage, our supervisor suggested us to meet 2 hours a day and 2 to 4 days a week, work and team and engage the team building, since the progress tracking form showed that the working hours were really unbalanced and some people apparently did not put enough effort into the project.

We made a daily sprint schedule, shows in figure 4 and followed the time to do work together. Everyone should follow the schedule and spend at least two days a week to work together as a team. It went well and most of us were able to follow the schedule. We worked in RHB306a, 35 Cafe or the Whitehead building lab. But since the strike started, the daily sprint stopped again because not all of us were coming to the campus due to the long distance between home and the university.

	START	END	Dylan	Marian	Mifuad	Hussein	Thairan	Jheng-Hao
Monday	14:00	16:00	X	X	Δ	X	O	X
Tuesday	17:00	19:00	O	O	O	O	X	O
Wednesday	14:00	16:00	Δ	X	Δ	X	O	Δ
Thursday	11:00	13:00	X	X	X	O	O	O
Friday	17:00	19:00	O	O	O	O	X	Δ

whitehead building lab: 18:00 to 19:00

Legends	Availability
O	Sure
Δ	Either one
X	No

Figure 4: Daily Sprint

2.3.2 Kanban

Apart from the Scrum, Kanban was also implemented with the backlogs in our development to demonstrate the development progress. We tried to keep our project management tool simple and easy to approach, so we not only used GitHub for version control but also for the project feature, which we used it as the Kanban. The iOS development kanban was shown in 5.

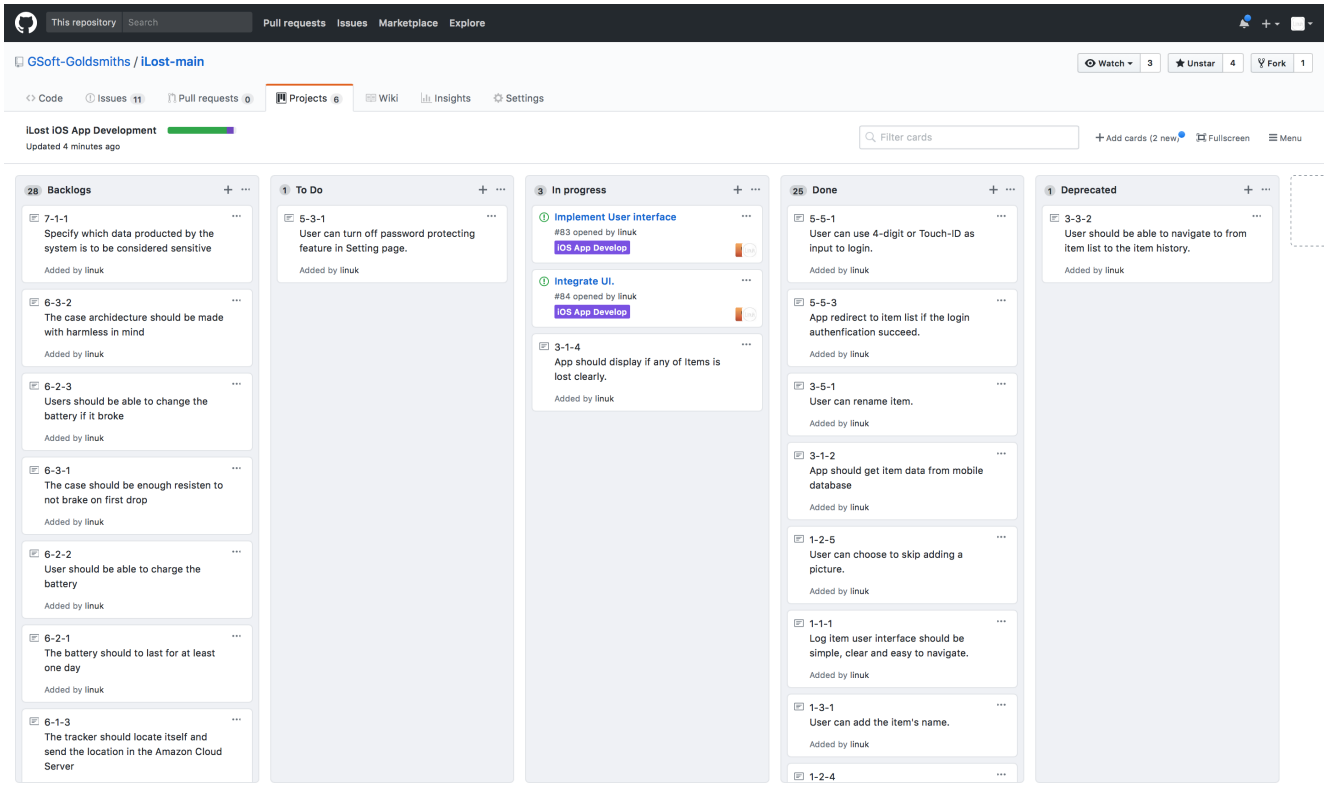


Figure 5: iOS app Kanban

Our kanban contained five columns which shows in Table 2:

Column	Description
Backlogs	The task is scheduled to be develop, but it is possible to be assigned deprecated if it is no longer needed.
To Do	The task is going to be develop.
In Progress	The task is currently developing.
Done	The task has been completed.
Deprecated	The task no longer needs to be developed.

Table 2: Kanban Columns

Apart from the normal columns: To Do, In Progress and Done, we also added **Deprecated** to store the features or tasks did not fit the needs anymore. Kanban gave a nice and clean overview of the current developing process, especially for people from other teams.

2.3.3 Behavior Driven Development

In order to produce the minimum viable product(MVP), we followed the Behavior Driven Development(BDD) to develop our product. To implement this methodology, we wrote the backlogs

of requirements specification at the beginning of the development stage. These requirements provided us with an overview of what was more urgent in what stage.

For example, displaying the item's position and historical locations was one of the most important functionality, so its' priority was **Must** and needed to be finished at the **MVP** stage. On the other hand, navigating to the item by triggering Google Map was helpful but might take longer to develop, so the priority was **Should** and it should be developed at the **Final** stage.

Here was our development process of BDD:

1. Requirements specification Review the backlogs and pick one from them to work on based on the priority and the estimated development time. You can find the details of the backlogs in section 2.4.1, Development Process - Backlogs.

2. Design Review the how was the data storage, data flow and the user interface design, then design how should this requirement be fulfilled. Such as displaying the item list in the item list view, we would need to implement two view components, an item list cell view and an item list table view. The item's properties data, including items id, name and photo, would be stored in the table view's state and the list cell would receive the props from the table view and render the item's properties.

3. Development Put the design into actual codes.

4. Integration Integrate the new view component with previously developed view components. Such as connecting the item list view with the item details view, where users could see its' location with a map.

5. QA reviews Conduct the quality assurance of the functionalities of latest and previously developed components. Please find more details in the chapter 5, Quality Assurance.

6. Usability tests After several developments, we would bundle them as a new version of our mobile application. Then do the usability tests to test if any of the components, user interface or user experience could be improved. Please find more details in the chapter 3, Formative Evaluation.

7. Maintance After the usability tests, we did some improvement based on the usability testing results.

This was a cycle process, after step 7 we went back to step 1 and kept going.

2.4 Development Process

2.4.1 Backlogs

Based on the user stories we made in the proposal, not only did we added more but also wrote sub-user stories and acceptance criteria for each of them. We had a list contain backlogs stored in Google SpreadSheet, which contained these columns:

Column	Description
User Story ID	The user story ID or sub-user story ID, a user story ID should be 1, 2, 3...n, and sub-user story should be 1-1, 1-2, 1-3, ... 1-m where 1 is the parent user story.
User Story	A brief description of the user story, starting with "As a user I want to ..." to describe what is the users' needs. Then followed by "so that ..." to explain why users need it.
Backlog ID	An ID number for the backlog, a sub-user story may contain more than one backlog to satisfy the sub-user story. If a sub-user story ID is 3-5, then the task ID will be 3-5-1, 3-5-2, 3-5-3, ... 3-5-p.
Acceptance Criteria	A description of the backlog which describe how the mobile application or the tracker would satisfy the user stories.
Priority	The priority of this backlog, should be **Muse**, **Should** or **Could**.
Dev Days	The estimated developing times in days, 8 hours counted as one day.
Phase	In which phase should the backlog be finished, in our case was **MVP** or **Final**.
Process	The current developing process, which should be **To Do**, **In Progress**, **Deprecated** or **Done**.

Table 3: Backlogs Columns

For example, the user story 3, **Track Item: User uses App to see the tracking list and track Item.** This section has 7 sub-user stories, which shows in table 4:

User Story ID	User Story
3-1	As a user, I want to view the tracking item list so that I can view my item and look up more detail if I want.
3-2	As a user, I want to activate or deactivate tracker so that I can save my mobile and the tracker's batteries.
3-3	As a user, I want to see the location history in a map format of the item so that I can track/look for it if it's lost.
3-4	As a user, I want to navigate me to my item so that find it quicker.
3-5	As a user, I want to edit my item detail so that I can change the tracking item.
3-6	As a user, I want to delete the item so that I can stop tracking the item for good.

Table 4: Backlog Example: User Story 3

Take sub-user story 3-1 as an example. In order to satisfy this user story, we had 4 backlogs and acceptance criteria to meet the requirements. Each of them had their own priority, estimated development days, phase and process. So this backlog will look like figure 6:

User Story ID	User Story	Backlogs					
		Backog ID	Acceptance Criteria	Priority	Dev Days	Phase	Process
3-1	As a user I want to view the tracking item list, so that I can view my item and look up more detail if I want.	3-1-1	Item list user interface should be simple, clear and easy to navigate.	Must	2	MVP	Done
		3-1-2	App should get item data from mobile database	Must	1	MVP	Done
		3-1-3	App should fetch lastest locations of all the tracking items when User visit item list page.	Should	1	MVP	Todo
		3-1-4	App should display if any of Items is lost clearly.	Must	2	MVP	In Progress

Figure 6: Backlog Example: Sub-user story 3-1

The full backlogs can be found in the appendix A.1 and it was used to create our Kanban showed in figure 5, which kept the developing process recorded easily to follow.

2.4.2 Progress Tracking

To record our progress, we kept using the same progress tracking form as the last term, and this term we did some improvement:

- Lock every week: To prevent any team members from modifying the resource hours dishonestly, a range of the form was locked every week. Only records within recent two weeks were allowed to be added.
- Status documented in more detail: To increase the traceability and convincing evidence, each resource hours commitment was asked to provide a more detailed description. Instead of *reporting writing*, it was improved to be *Final report: Formative Evaluation - mobile application test.*

Please find the full progress tracking form in appendix A.4.

Until March 16, the total working time during these two terms is **581.9 hours**. All these hours contribution includes all lab sessions, supervisor meetings, daily sprint and self-independent working time. 58.7% was contributed by Hussein and Jheng-Hao as shown in figure 7. The top contributor is Jheng-Hao with 222.95 hours contributed, while the last is Chin with 26.5 hours recorded. The line chart displays that the commitment became dramatically different since the start of the second term. Most of the team members stopped to work in the second term, excluding Hussein and Jheng-Hao who still kept working continuously.

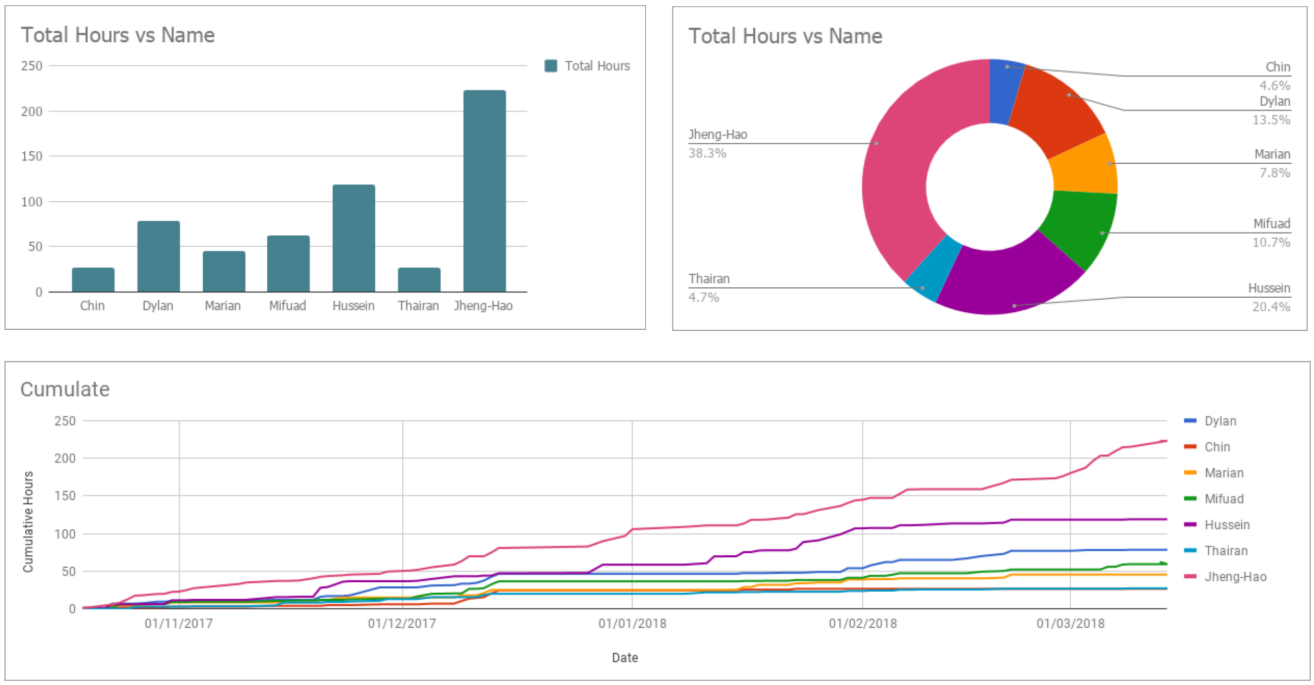


Figure 7: Progress Tracking Diagram (until March 16th)

2.4.3 Evaluation

Scrum This methodology is suitable for a team where they tend to work together in a fixed time, day-to-day. Even though we had scheduled the daily sprint time, not everyone would attend since it was not compulsory and registered. Only few sprint sessions at the beginning went well. In terms of time management, not every one of us were able to contribute fixed time every one or two days, so we could not have a daily sprint or even weekly sprint. Unless we have could work full time for this project and everyone is willing to meet at a place and work together, or it would be better not to implement Scrum in our team.

Kanban Regards of project management, Kanban went well, but it would be better if we could have a workspace where we could combine our backlogs and Kanban. We found that there are two web applications that could provide this kind of services: Clubhouse and Jira They are the web applications where we can add user stories which link to kanbans at the same time. It would be nice to use this kind of tools to save time if the team has enough budget.

Backlogs The backlogs were the most important things we could improve on. We only listed the basic cases while the extreme use stories were not in our backlogs. For example, **"As a user I want my tracker to keep working in the snowing seasons"** could be in our user story as our product should still be functional in the winter season.

Progress Tracking In terms of progressing tracking form, it should added one column which links to the contributions, either their GitHub commit or documentation. It was not convienent to record the hours each individual worked without any evidence but this did happen. Also, it was not really feasible to use working hours to represent the contributions. With the same task, people spend less time to finish the same job should be rewarded higher compared to who spend longer period to do so. But we were not able to do this since everyone worked on different tasks. It

would be fairer if each task could be evaluated by the importance, whoever completes the higher importance means contributing more. We should have a contribution list which records everyone's participation. Then with the list, we could write the peer access form in a more objective point of view.

3 Formative Evaluation

3.1 iOS App Evaluation

3.1.1 Objectives and Questions

We wrote quantitative tasks to test the usability of our mobile application[3]. The purpose of the tests was to test if the participants can actually finish the task with the application, and we could learn from the process to observe how users used our application and improve it if any issue or confusion was raised.

Task ID	User Goal	Task	Scenario	Developing Status	Can be tested		
					v0.10	v0.11	v0.12
1	Log item in the app	Add my bag as a new item and pair a tracker in the mobile app	You are planning to find a solution to prevent losing your bag in public again, so you buy a iLost tracker online now and download the free mobile app in your phone. Go to the app to add your bag in the tracking list.	Done	Y	Y	Y
2	Check items status	See if all my bag item is safe.	You are have successfully log your item and pair the tracker with the mobile app. Go to the app to check if your bag is safe.	Done	Y	Y	Y
3	Check item position	See my bag's current position	You just come back from a chill night in a bar, you realise that you forget your bag at somewhere unitl arriving home. Check where is your bag now.	Done	Y	Y	Y
4	Check item position records	See my bag's 10 minutes before's position	You have checked your bag's current position and feel weird why is it in a place you have never been before, so you deciede to see how it end up there. Check your beg's location ten minutes ago.	Done	Y	Y	Y
5	Trigger native maps app navigation with current position	Navigate me to the my bag's current position in a public tranportation way.	You are really worry about where is your bag , because it seems like someone has taken it to somewhere, and inside your bag are all your important belongings. Go find your bag with the app's navigation feature.	Done	Y	Y	Y
6	Trigger native maps app navigation with history position record	Navigate me to the my bag's 10 minustes before's position on foot.	There is something misiing from your bag, it might be left outsied of the bag, Go and find your bag's previous location 35 minutes ago .	Done	Y	Y	Y
7	Edit item name or photo.	Attach the tracker to other item.	You just bought a new bag and you would like to attach the original tracker to it. Go to one of the tracker view to edit the item's name and photo.	Done	Y	Y	Y
8	Receive notification	Receive notification once my bag is lost.	You receive an notification since the app detecting your item is too far away from you. Check the notification and find where is you lost item.	Done	N	N	Y
9	Setup password	Setup password for the app	Now you know how iLost can help you find your lost precious, but the app need to be kept in private otherwise other people can the app to track you! So you decide to prevent someone else to use the app by setup a password in the app.	Done	N	N	Y

Figure 8: Usability test task list

Column	Description
Task ID	Identify the number of the task.
User Goal	What is the objective we need users to perform.
Task	The process in terms of completing the goal.
Scenario	A setup scenario for engaging testers to use the application in a real-life case.
Developing Status	Current latest developing status of the functionality of this task, it should be **To do**, **In Progress** and **Done**.
Can be tested	Whether the functionality of the application is ready to be tested in each version.

Table 5: Usability test task list columns

Figure 8 demonstrates our tasks and goals. Table 5 shows what the columns stand for.

3.1.2 Participants, Location and Setup

According to Jakob Nielsen, testing 5 users in a usability study could find almost as many usability problems as testing more participants[2]. Following this theory, our application was tested with 5 participants for each version that was created. The study was taken place in the library of the Goldsmiths, University of London, and the participants were the students who used to bring a bag to the campus daily. Totally we have conducted three versions of the application.

3.1.3 Methodology and Measures

Firstly, we explained what was our project about and asked participants to sign up the consent form which can be found in appendices B.1. During the test, the participants were provided an iPhone with the application built-in to test. An observer would guide them through the tasks and the scenarios, then took notes of how the participants reacted to the application. For each task, the observer would record it was successfully completed or failed. The records would help us to build the success rates diagram which helped us to understand which usability needed to be improved[4].

3.1.4 Evaluation

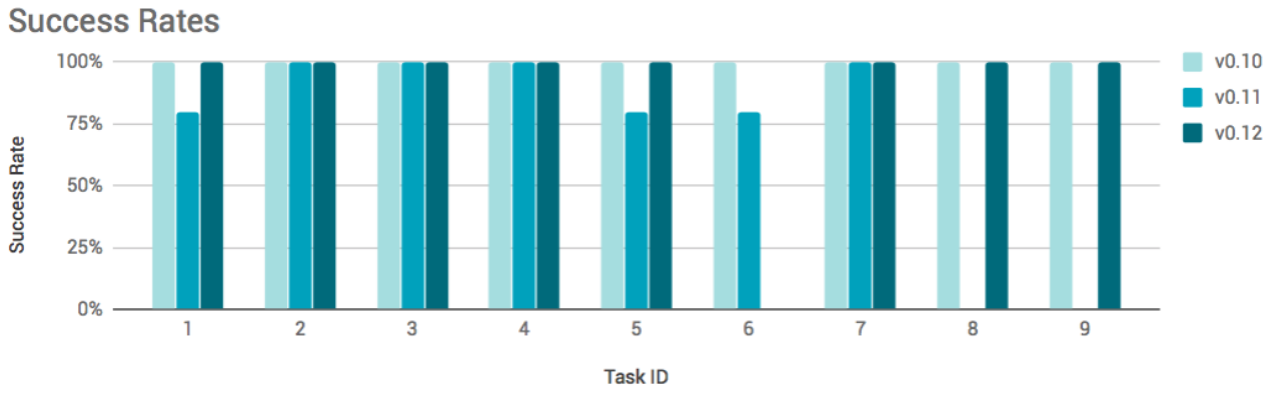


Figure 9: iOS App Success Rates Diagram

v0.10 The tested subject was the application prototype built in **Adobe XD**, which was one of the best design tools to test the prototype. Thanks to the well-designed user interfaces, the goals were easy to achieved and the tasks were all successful. The results only proved that the user interfaces guided the users to the right view, however it could not actually reflect on the usability of the real application. After all, Adobe XD could only let users walk through each view by clicking, while an actual iOS application would support swipe or other gestures. It was more like a paper prototype usability test.

v0.11 We benefited most from this test since this test was the native mobile application we where the participants can actually use it like other application. This version was built in React Native and tested in Expo which a tool and service which we used to build the mobile native application with React Native. During this test, we received several comments towards the **add item view**. For example, the camera icon in that view was originally used as a button. But some of the users could not really regard it as a button but a decoration since it was colourful. Also, the placeholder of the item name field was **Item Name** instead of a prompt message which confused

some of the participants as well. So after this test, we resolved the issues and the difference shows in figure 10. The task 8 and 9 were not developed completely at the moment so there was not any record.

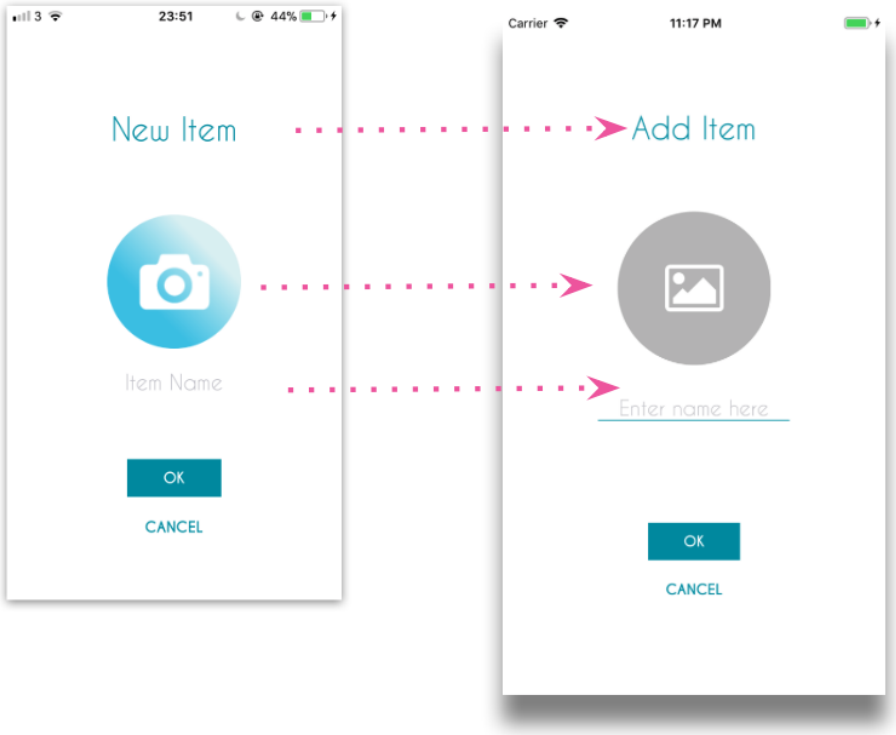


Figure 10: Improvement of the add item view

v0.12 After the previous test, we not only improved some of the user interfaces but also removed some of the features. It is worth notice that there is no record of task 6, which is **navigating to previous locations of the item's location history**. The reason we removed this task was that this goal was not really helpful. The participants commented that it was not beneficial to navigate to the previous locations, they cared about the current location of the item more. So thanks to the feedback, we removed task 6 and eliminated the functionality of navigating through history locations which made the application more simple.

3.2 Tracker Evaluation

total 700 words

3.2.1 Objectives and Questions

100 words

3.2.2 Location, Setup and Participants

100 words

3.2.3 Methodology and Measures

100 words

3.2.4 Tracker v0.10 Evaluation

150 max words

3.2.5 Tracker v0.11 Evaluation

150 max words

3.3 Conclusion

Even though the tasks could be completed, but in terms of the user experience, we still had space to improved.

4 Design and Implementation

4.1 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 (figure 12) is defined as less than 30 metres from the target and long-range (figure 11) is defined as above 30 metres to 400 km and beyond.

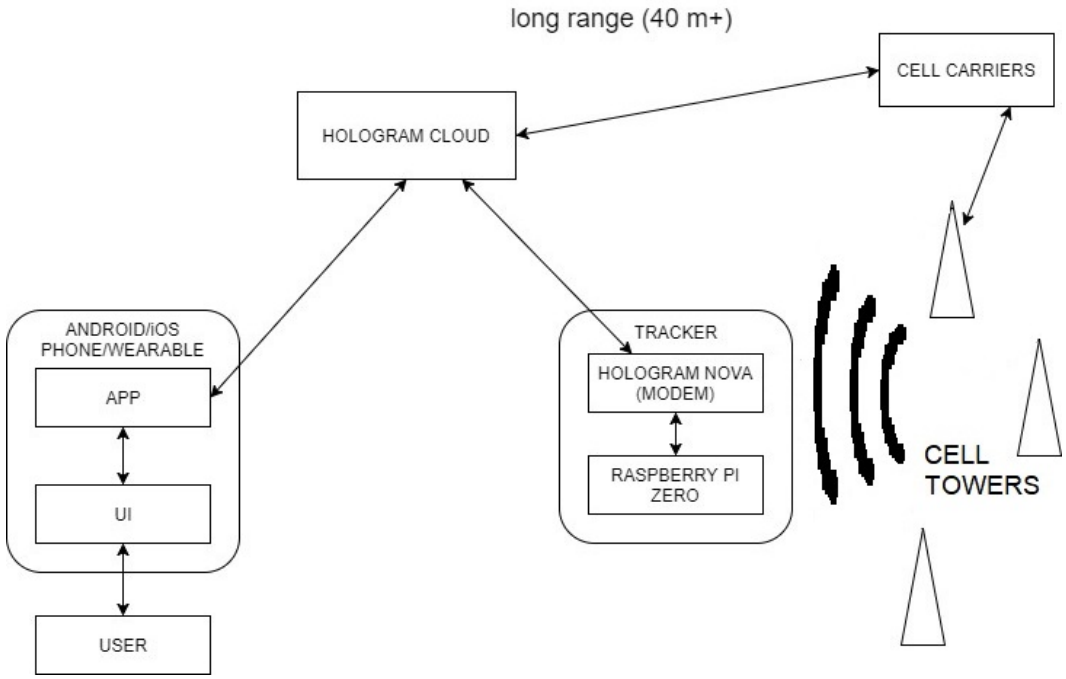


Figure 11: Tracker concept 2.0 at January - long range

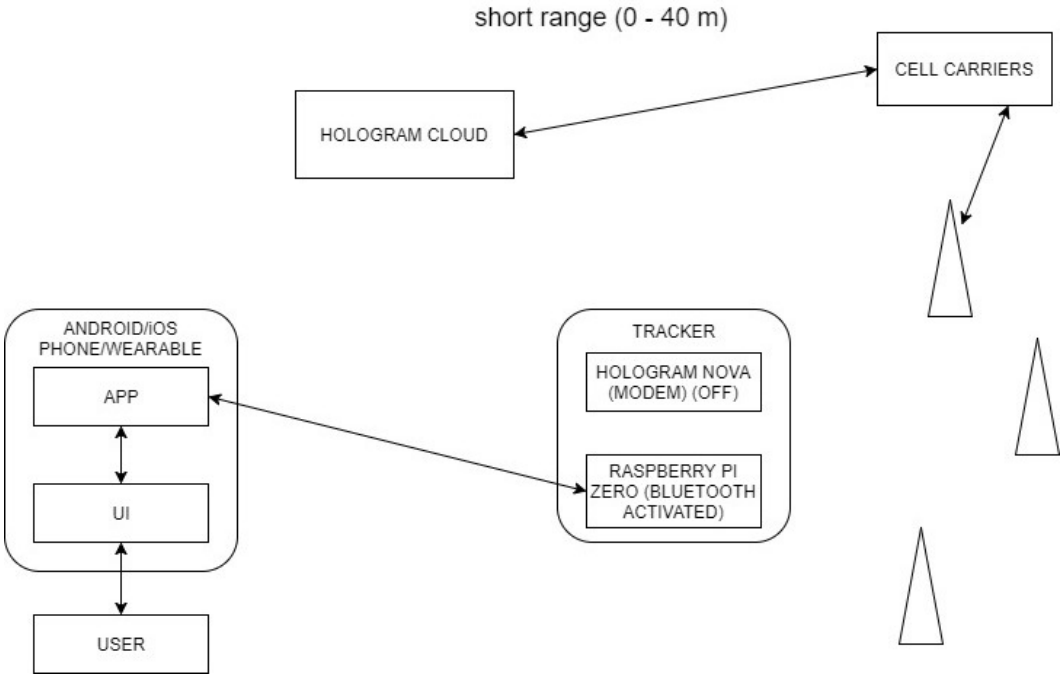


Figure 12: Tracker concept 2.0 at January - short range

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 in figure 13). 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).

Figure 13: 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 users 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 users 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.

4.2 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 shows in figure 14, roughly 53% 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.

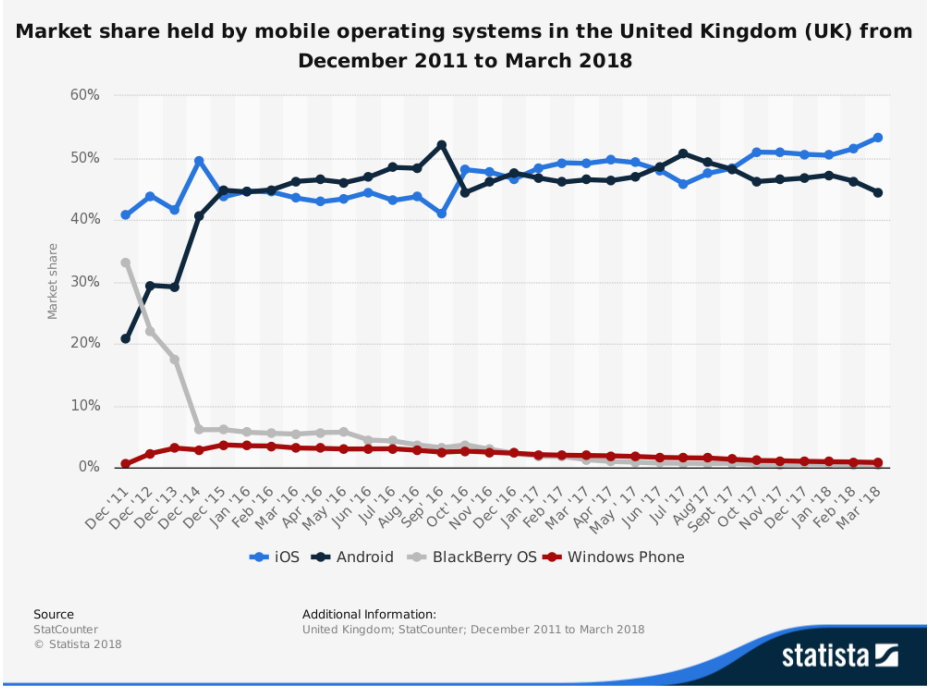


Figure 14: Mobile OS Market Reseatch in UK [5]

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 users smartphone.

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 (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 (its 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.

Moreover, we experimented with other Bluetooth based technologies, to find a replacement. We tested and built mock-ups using the estimote beacon (a Bluetooth tracker, used in upcoming retail and asset tracking), this did not fit our use case, as it increased unnecessary cost and complications to the user. Another novel Bluetooth tracker which failed the use case was the puck.js (a small Bluetooth tracker that can programmed using JavaScript) although a novel concept we did not possess enough technical skill to be able to program the Puck to detect read Bluetooth signals specifically emitting from the raspberry Pi.

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 didnt 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.

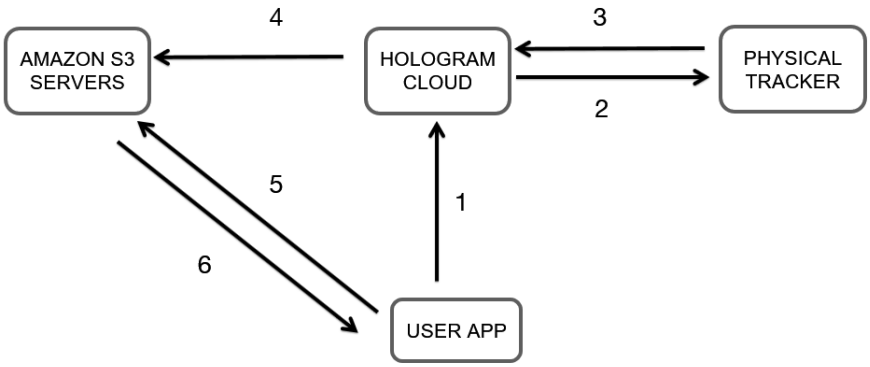


Figure 15: Current Stack

Post-Snowden age, we understand users are concerned about privacy. All data that is stored on Amazon servers are automatically encrypted AES-256 servers side encryption keys. We only store longitude, latitude and time data, for 14 days. Its saves us space and cost, to not store data permanently.

Location data such as longitude and latitude is metadata, its ambiguous not content that can identify the users personal information. The purpose of the Amazon server is only required to provide data back to the user. Also, each tracker has unique ID number, personal information is not identifiable from the data.

We cannot guarantee compliance with UK Data protection act, due to the legal complexity of having the Amazon operating servers globally as well as the Hologram nova having to operate with many network carriers globally. Ideally, with more funding we would be able to consult with legal professionals.

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.

However, we migrated from the iOS app being built in the swift language to React Native. React Native is framework that allows developers to build mobile apps for both the iOS and android platform using JavaScript and the React framework.

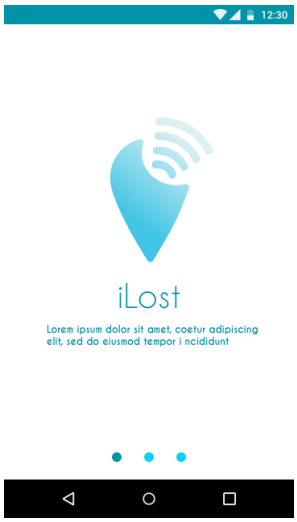
At the beginning of the project, our iOS developer team did not have experience with using the Swift language and associated frameworks. We spent 3 - 4 weeks building a basic application which only had a tab bar view to switch between pages, and only half of the user interface components were functional.

The Android app and iOS app user interface design can be found in figure 16 and figure 17.

Due to the limited development time, we searched for an alternative method that would reduce development time. We found that React Native and Flutter were good alternatives to develop native mobile applications. React Native was chosen because it was more mature than Flutter, which was relatively unstable as it had just been released. Unexpectedly, only three days were required to develop the same application, which the iOS team spent almost a month to build, in React Native. Switching to React Native was a good decision as it significantly shortened the development time providing us more time to do usability tests.

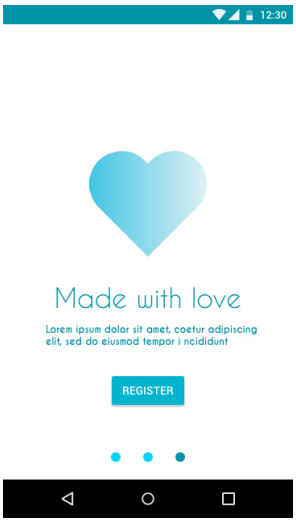
The iOS team also had previous experience in developing React web applications, where React shares the same component-based concept and reactive function programming paradigm with React Native. Also, React Native uses JavaScript which we were more familiar with, so it was relatively easier for us to develop the application.

The Swift API had many noteworthy changes from Swift 2.0 to 4.0 which is the latest version (we were developing with latest version), so whenever we encountered an issue, it was difficult to search for the solution for the latest version of Swift. The official documentation of Swift was also quite ambiguous, it lacked working examples. This led to a steep of learning curve when developing in Swift. On the other hand, React Native had better documentations and community support. The official documentation was explained clearly with good working examples, there were more community-built components can be approached easily as well, this improved the developing experience and made it easier.

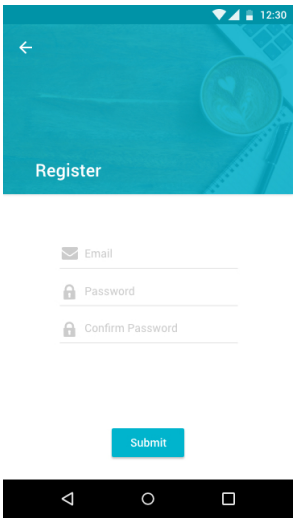


(a) Welcome-1

(b) Welcome-2

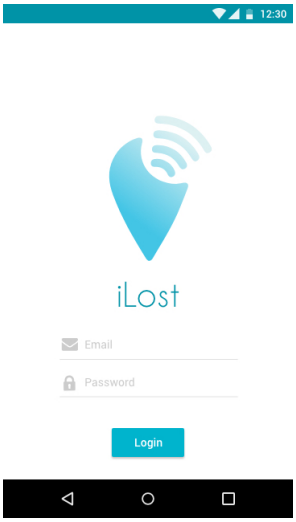


(c) Welcome-3

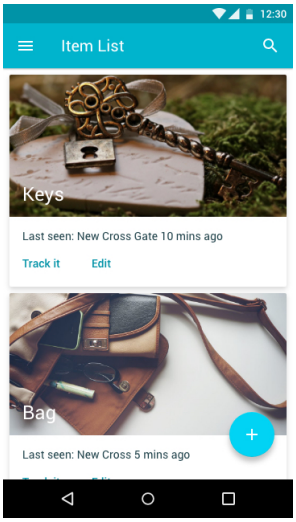


(d) Register

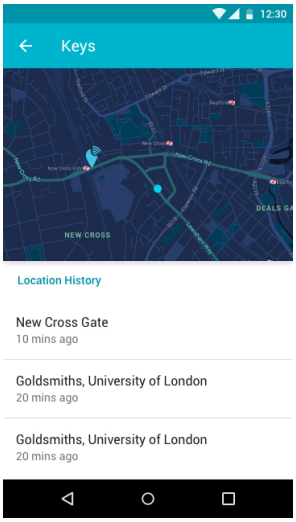
(e) Register Succeed



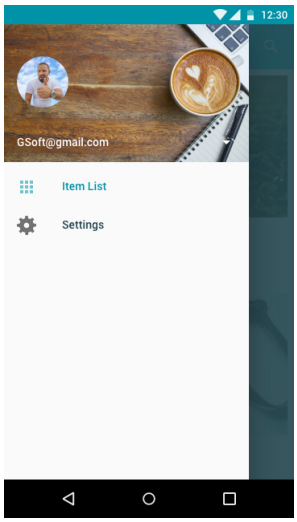
(f) Login



(g) Item List



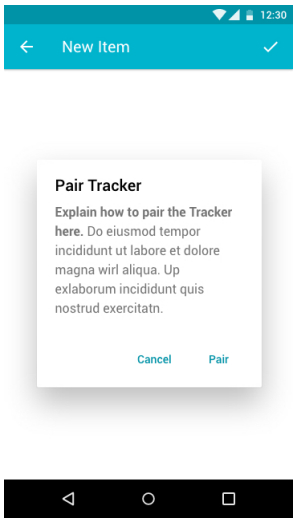
(h) Track Item



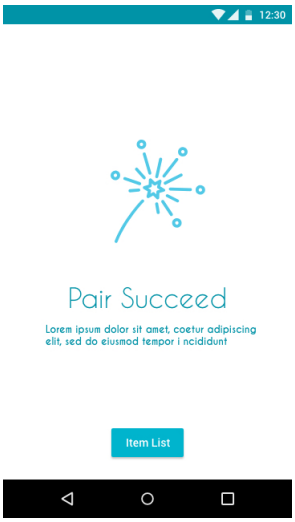
(i) Menu

(j) Add Item Photo

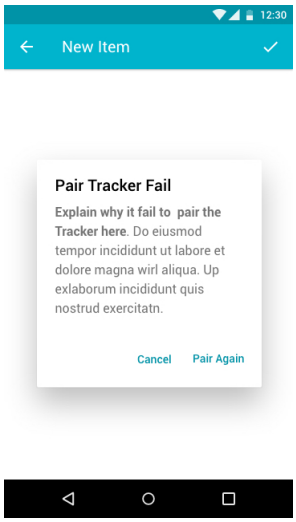
(k) Add Item Name



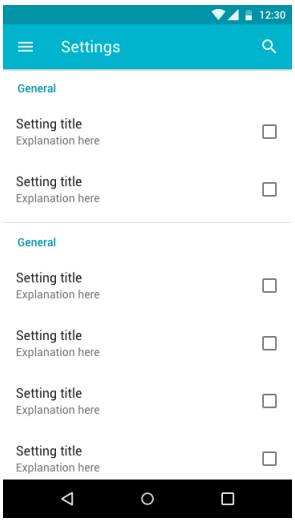
(l) Pair Tracker



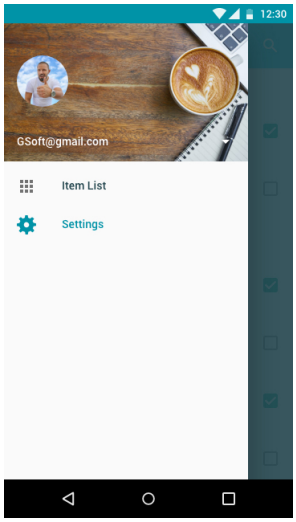
(m) Pair Tracker Succeed



(n) Pair Tracker Fail



(o) Settings

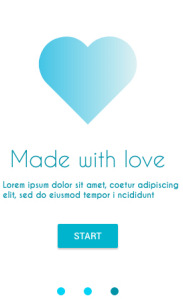


(p) Setting Menu

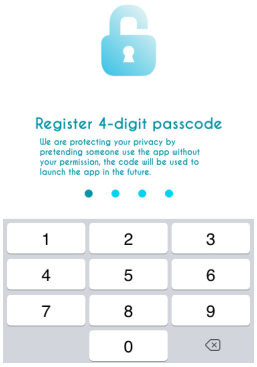
Figure 16: Android App User interface Design

(a) Welcome-1

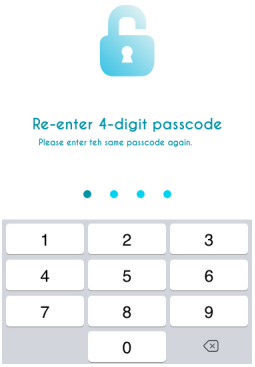
(b) Welcome-2



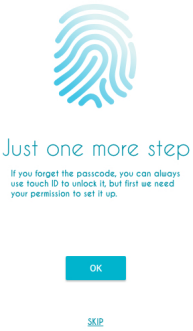
(c) Welcome-3



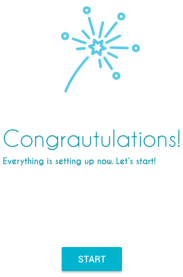
(d) Register



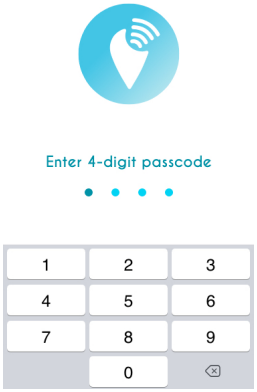
(e) Register Passcode



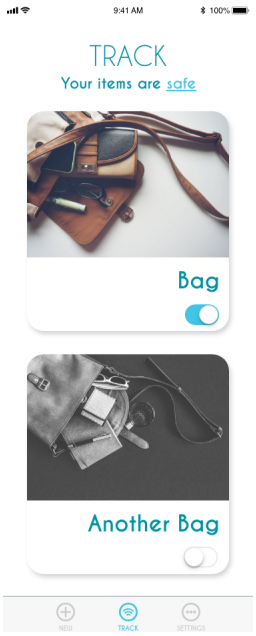
(f) Register Touch ID



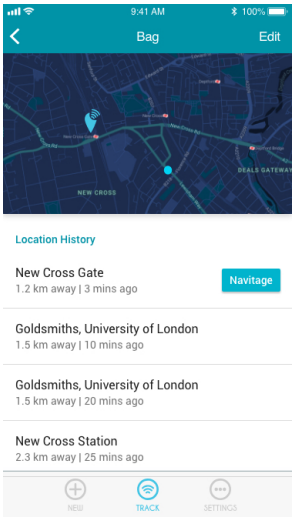
(g) Register Touch ID Succeed



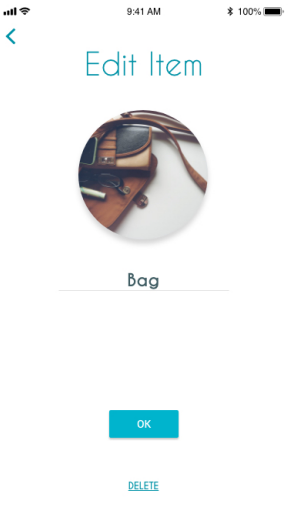
(h) Login



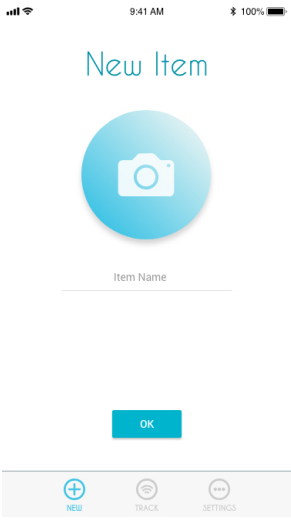
(i) Item List



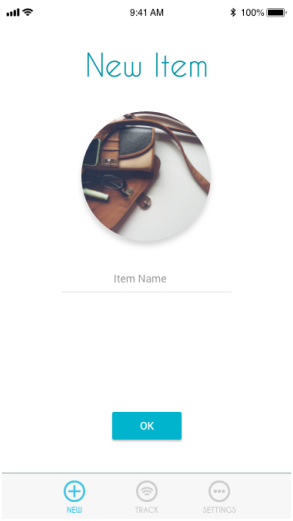
(j) Add a new Item - Track Item



(k) Add a new Item - Edit Item



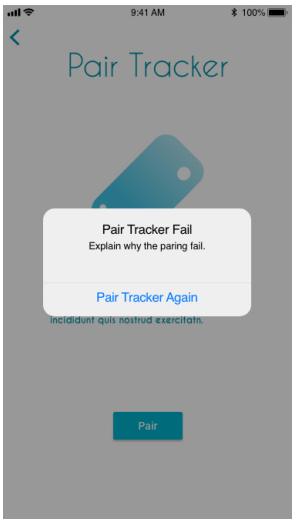
(l) Add a new Item - Photo



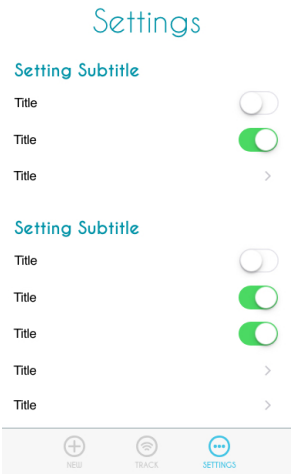
(m) Add a new Item - Name

(n) Add a new Item - Pair Tracker

(o) Add a new Item - Pair Tracker Succeed



(p) Add a new Item - Pair Tracker Fail



(q) Setting Menu

Figure 17: iOS App User interface Design

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.

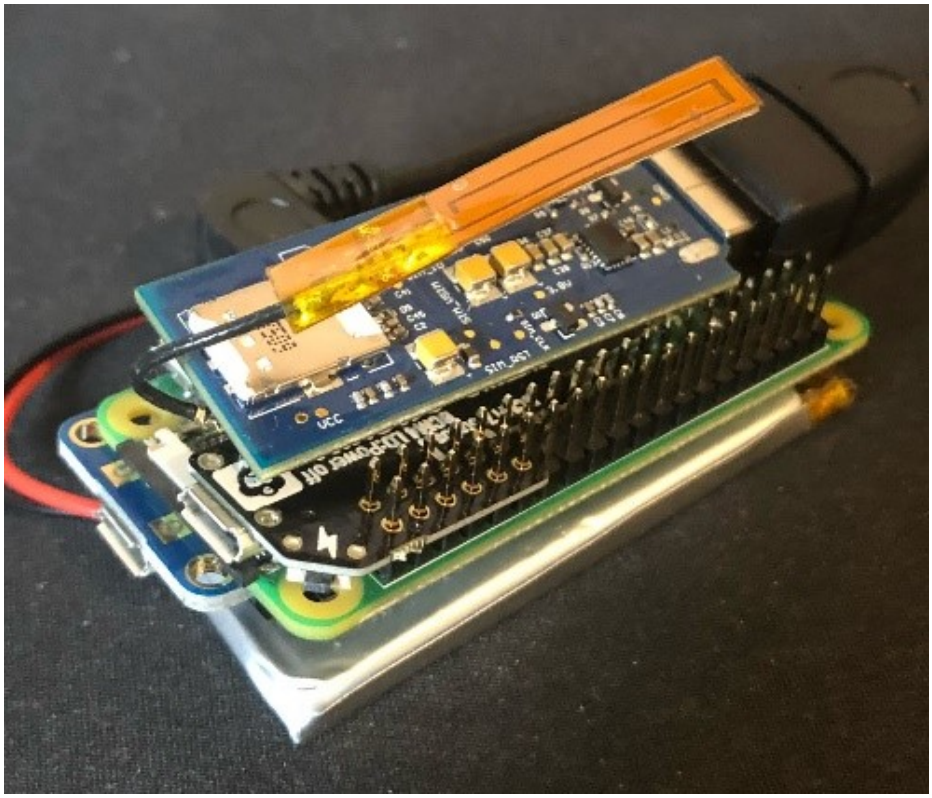


Figure 18: Tracker without case

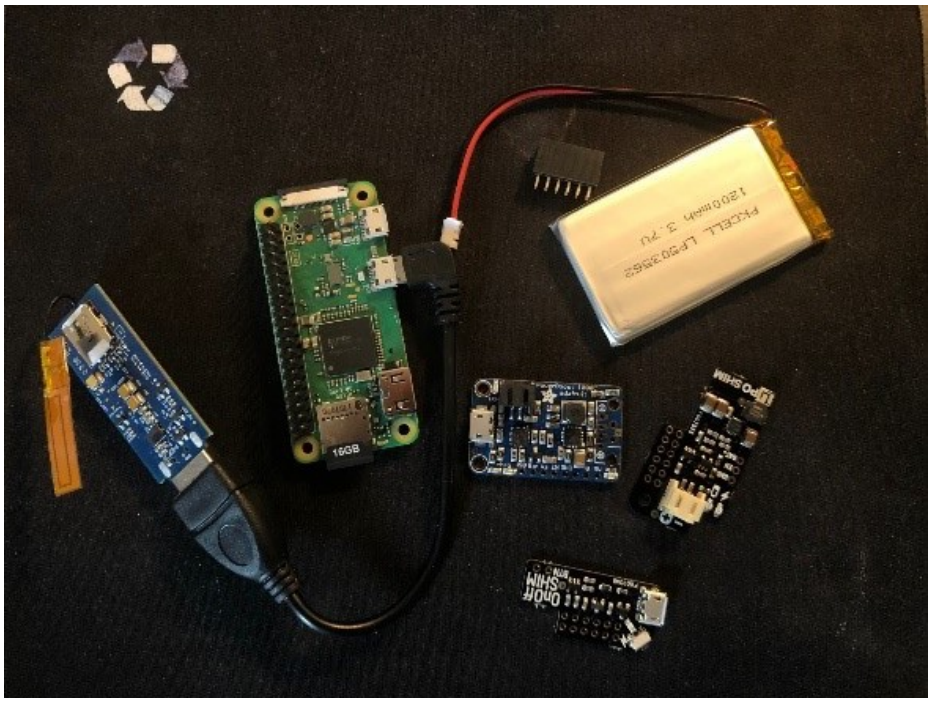


Figure 19: Tracker with all components

Moreover, a physical case which would fulfil the user use cases was needed for the tracker. Namely, it needed to be reasonably drop proof, strong enough to withstand day to day life, durable, simple and functional enough to be attached to a bag. A prototype was created using

laser-cut cardboard, which was interlocking. As a first iteration, it was useful as it allowed us to show to users how our Physical tracker would look, in terms of size (how it compared to other items a user may carry), this iteration was also low cost as it was made from cardboard. By creating the first iteration, we could also see dimensions required for the internal components of the tracker and how these components would behave inside an enclosed space.

Some members of team also undertook 3D printing training, the 3D model design shows in figure 20 and figure 21. Ideally, we wanted the case to be 3D printed, unfortunately we did not have enough time to progress beyond the cardboard prototype, which shows in figure 22.

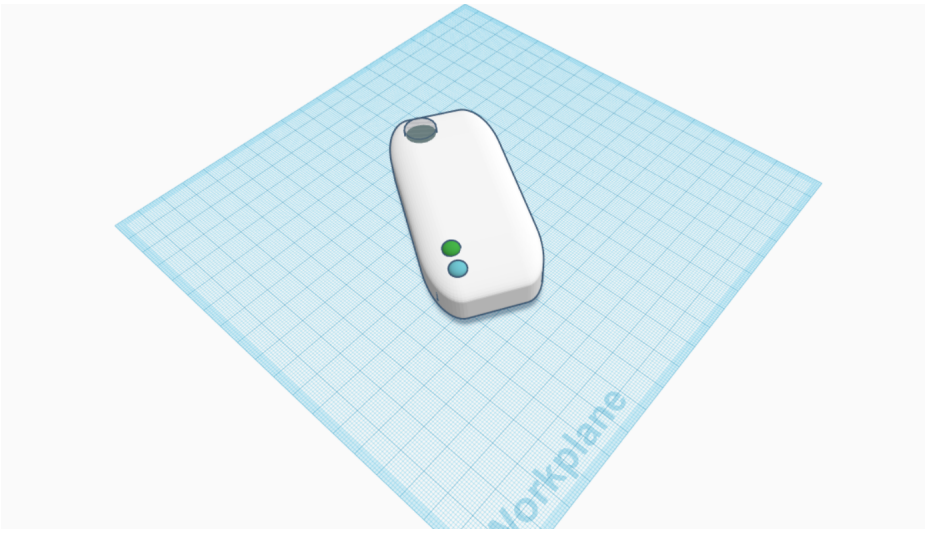


Figure 20: Concept Tracker Case 1

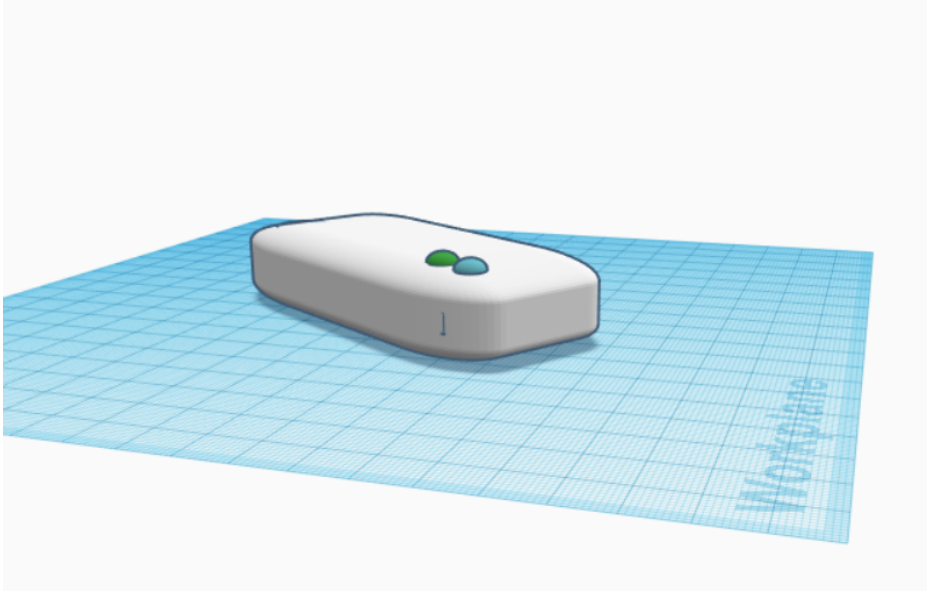


Figure 21: Concept Tracker Case 2

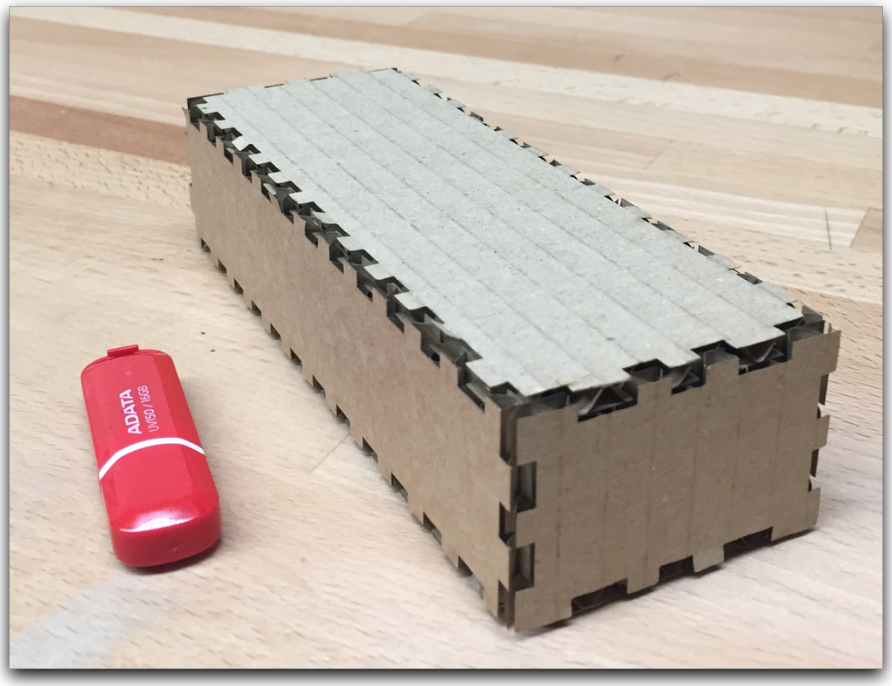


Figure 22: Concept Tracker Demo

5 Quality Assurance

5.1 Process

We conducted two types of quality assurance to ensure our mobile application working as we expected, including black-box testing and white-box testing. While the white-box testing focused on the functionalities, such as how an item list view should react when an add new item button was clicked, the black-box testing covered how a view component or a function should be rendered or implemented, such as a button should render a text on it based on what text state passed to it. Usually, the black-box tests were conducted after the white-box tests, since the functionalities could be functional only if the component worked properly.

Since the time limitation, only the iOS version was fully tested, but the test cases could be implemented in both version of the application.

5.2 Black-box Tests

5.2.1 Method and Environment

The black-box tests covered the functional testing testing of the use scenarios, they were developed by the acceptance criteria of the backlogs. Since our mobile application was not really complicated and complex, we did not implement any automated way to conduct the tests but manually merely. We chose **use case testing** as our approach to conduct the static tests, since it was the quickest way we could conduct the tests with existing resources.

The iOS application was written in React Native, it was imported and tested in the Expo [6] application on an iPhone 6. Expo is a mobile application where everyone can download

others' React Native codes and run on their iPhone or iPad, rather than paying USD 3000/year to subscribe the Apple Developer Programme to use TestFlight [7], which is the official testing application owned by Apple Inc.

5.2.2 Unit Tests

Our use cases table was adapted from the format wrote by a software consultant company IntexSoft [8]. The test cases contained the columns shows in table 6:

Column	Description
ID	Unique ID of the test case
Name	Name of the test case.
Goal	The detail description of the purpose of the test goal.
Precondition	What need to be satisfied before the test case can be tested.
Success End Condition	The definition of how the test case considered as passed
Feiled End Condition	The definition of how the test case considered as failed
Trigger	How this use scenario is triggered.
Normal Flow	The flow of this test case, such as how the stakeholders react with applicatoin.
Alternative Flows	The details of alternative ways to different successful ends.
Frequency of Use	How often has this uses cases been triggered, we use **Rare**, **Normal** and **Often** to represent various level of the frequency since we did not have a proper statics for each use case.
Assumptions	Any assumptions that were made during the analysis of the use case.

Table 6: Black-box Test Cases Columns

So for example, one of the test cases is to trigger the Apple map and navigate the user to the item's location. The fields look like table 7:

Column	Description
ID	30
Name	Item details view: trigger Apple Map.
Goal	The application should be able to use the item's location data, pass it to the Apple Map application, and trigger the navigation function and lead the user to the item's location
Precondition	User is in the item details view.
Success End Condition	The Apple Map application is triggered after the user taps the navigate button. It gets into the navigation mode and leads the user to the location based on the distance. If the user is closed to the item, then it will navigate the user to the item in "walking" mode. If the item is far from the user it will be in the "transit" mode
Failed End Condition	The Apple Map application is not triggered after the navigate button is tapped. Or the Apple Map application is triggered but it does not receive the correct location data of the item and activate the navigate mode either.
Trigger	User taps the navigate button
Normal Flow	1. The user taps the navigate button. 2. The application triggers the Apple Map application and passes the geolocation of the item to it. 3. Apple Map automatically activates navigation mode and lead the user to the item.
Alternative Flows	N/A
Frequency of Use	Often
Assumptions	There are item's location data stored in the application already

Table 7: Black-box Test Case: Trigger Apple Map

Please find more details of the black-box test cases in the appendix D.1

5.3 White-Box Tests

5.3.1 Method and Environment

The iOS application was developed in a component-based way, and we tested most of the elements whenever a new one was built. In React Native, there are two types of components, pure components and containers. Pure components do not store any data but render whatever data are passed to them only, The passed data in React are called "props". On the other hand, containers are the components store the data, which is named "state", and render the pure components. In an MVC model, the pure components are more the **View** and the containers are the **Controller**. We were only able to test the pure components at the time limitation. Testing the pure components was relatively easy and quick, and it helped us to debug and boost the developing efficiency.

The testing framework we chose was Jest[9] and Enzyme[10], a JavaScript testing framework developed by Facebook. The main reason we chose Jest rather than other testing frameworks was that it was maintained by the same company of React Native. There were not only resourceful community supports but also the official documents were really well documents. These reasons made it one of the trendy testing framework in the React ecosystem. Mocha, another popular

JavaScript unit testing framework, was considered too. However, since the iOS team did not have any previous experience with it, so to meet the deadline we chose to use the most familiar tool.

The iOS application was developed in a component-based way, and we tested most of the elements whenever a new one was built. In React Native, there are two types of components, pure components and containers. Pure components do not store any data but render whatever data are passed to them only, The passed data in React are called "props". On the other hand, containers are the components store the data, which is named "state", and render the pure components. In an MVC model, the pure components are more the **View** and the containers are the **Controller**.

We were only able to test the pure components at the time limitation. Testing the pure components was relatively easy and quick, and it helped us to debug and boost the developing efficiency.

5.3.2 Test Cases

The testing framework we chose were Jest[9] and [10], Jest was a JavaScript testing framework developed by Facebook and Enzyme was the React component testing utility. There were several reasons made us choose these tools rather than other testing frameworks. Firstly, Jest and Enzyme were developed and maintained by Facebook and Airbnb developers team. The resourceful community supports and the well-written official documents were all really helpful. These reasons made these tools became the trendy testing frameworks in the React ecosystem. Mocha, another popular JavaScript unit testing framework, was considered too. However, the iOS team did not have any previous experience with it, so we chose to use the most familiar tools in order to meet the deadline.

Jest and Enzyme were combined together into one testing file and test if a component was rendered properly. Button was one of the pure components which has been unit-tested, which contained two props: **text** and **handler**. The **text** was the text rendered on the button, and the **handler** was the function which would be called whenever the button was pressed. The button code shows in Listing 1:

```
import React from 'react';
import { mainFontBold, style } from '../styles/variables';
import { Text, TouchableHighlight } from 'react-native';

export default ({ text, handler }) => (
  <TouchableHighlight
    style={style.button}
    onPress={() => handler()}
  >
    <Text style={{ color: '#fff', fontFamily: mainFontBold }}>{text}</Text>
  </TouchableHighlight>
);
```

Listing 1: Button.jsx

In terms of the button component, we tested it in four aspects:

- **Rendered without crashing**: We used the snapshot API from Jest. A snapshot was JSON string of the component, during the test the newly built component would be converted to

JSON and compared with the previously built snapshot. The test would be passed if both snapshots were the same.

- **Rendered with correct elements**: Text and TouchableHighlight should be rendered in the button component, we tested if they existed.
- **Rendered with correct props**: The text field was rendered with a text prop, which should be rendered in the Text component inside. So we tested it if the Text component was rendered with the correct text.
- **The button handler should work if the button is pressed**: Tested if the handler was functional if the button was pressed simulated.

The actual code shows in Listing 2:

```
import React from 'react';
import { Text, TouchableHighlight } from 'react-native';
import { shallow, configure } from 'enzyme';
import renderer from 'react-test-renderer';
import Adapter from 'enzyme-adapter-react-16';

import Button from '../../src/components/Button';

configure({ adapter: new Adapter() });

describe('<Button /> ', ()=>{
  it('renders without crashing', () => {
    const rendered = renderer.create(<Button text="title" handler={() =>
        {}}/>).
      toJSON();
    expect(rendered).toMatchSnapshot();
  });

  it('renders with correct elements', () => {
    const wrapper = shallow(<Button text="title" handler={() => {}}/>);
    expect(wrapper.find(Text).exists()).toBeTruthy();
    expect(wrapper.find(TouchableHighlight).exists()).toBeTruthy();
  });

  it('renders with correct props', () => {
    const wrapper = shallow(<Button text="title" handler={() => { testData =
        "button is pressed"; }}/>);
    const text = wrapper.find(Text);
    expect(text.props().children).toEqual("title");
  });

  it('the button handler should work if the button is pressed', () => {
    let testData = "before button is pressed";
    const wrapper = shallow(<Button text="title" handler={() => { testData =
        "button is pressed"; }}/>);
    wrapper.find(TouchableHighlight).simulate('press');
    expect(testData).toEqual("button is pressed");
  });
});
```

Listing 2: Button.test.jsx

This is how we unit test the pure components. Our application used most of the native components provided by Expo SDK and React Native, so only 4 components were needed to be tested: **Button**, **FullPageView**, **ItemListCell** and **PasswordInput**. The other 9 container components were tested if they were rendered properly without crashing only. The source code for unit testing could be found in our public Github repository: `https://github.com/GSoft-Goldsmiths/iLost-main/tree/master/src/react-native-app/tests`

5.4 Evaluations

Both types of tests helped us to maintain the quality of our software before actually conducting the usability tests with the participants, and here are some points could be improved in terms of our quality control:

- **Add extreme cases**: The test cases aimed to test the normal use cases since we did not have much time to test the extreme cases. But in real life, the boundary value analysis is relatively important in terms of the software testing as well apart from the normal use cases.
- **Automated the tests**: During the black-box tests we conducted the tests manually, but in fact, we could automate the process and save us a lot of time. In fact, Jest, the tool we used to automated unit testing, supports the automated application testing, it is worthy to do more research on it.
- **More tests related to the tracker**: Since the limited time, we did not have much time left to do the tests for the physical tracker. It is important to test any part of our product, especially the integration between the tracker and the mobile application should be well tested.
- **Follow Test Driven Development(TDD)**: TDD was one of the development methodologies we intended to implement since it controlled the code quality. But the trade off was slowing down the developing speed. For building an MVP, TDD might not be a good option, but in the future development, it would be essential.

6 Summative Evaluation

References

[1] M. Nygard, "Blog — Documenting Architecture Decisions — Relevance", Thinkrelevance.com, 2011. [Online]. Available: http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions. [Accessed: 15- Mar- 2018].

[2] "How Many Test Users in a Usability Study?", Nielsen Norman Group, 2012. [Online]. Available: https://www.nngroup.com/articles/how-many-test-users/. [Accessed: 01- Mar- 2018].

[3] "Writing Tasks for Quantitative and Qualitative Usability Studies", Nielsen Norman Group, 2018. [Online]. Available: https://www.nngroup.com/articles/test-tasks-quant-qualitative/. [Accessed: 14- Mar- 2018].

[4] "Success Rate: The Simplest Usability Metric", Nielsen Norman Group, 2001. [Online]. Available: https://www.nngroup.com/articles/success-rate-the-simplest-usability-metric/. [Accessed: 14- Mar- 2018].

[5] M. 2018, "Mobile OS: market share in the United Kingdom 2011-2018 — Statistic", Statista, 2018. [Online]. Available: https://www.statista.com/statistics/262179/market-share-held-by-mobile-operating-systems-in-the-united-kingdom/. [Accessed: 20- Apr- 2018].

[6] "Expo", Expo, 2018. [Online]. Available: https://expo.io/. [Accessed: 18- Mar- 2018].

[7] "TestFlight - Apple Developer", Developer.apple.com, 2018. [Online]. Available: https://developer.apple.com/testflight/. [Accessed: 18- Mar- 2018].

[8] "use case testing - IntexSoft", Intexsoft.com, 2015. [Online]. Available: http://www.intexsoft.com/blog/item/117-use-case-testing.html. [Accessed: 18- Mar- 2018].

[9] "Jest Delightful JavaScript Testing", Facebook.github.io, 2018. [Online]. Available: https://facebook.github.io/jest/. [Accessed: 18- Mar- 2018].

[10] "Introduction Enzyme", Airbnb.io, 2018. [Online]. Available: http://airbnb.io/enzyme/. [Accessed: 19- Mar- 2018].

Appendices

A Development Records

A.1 Backlogs

User Story ID	User Story	Backlogs					
		Backog ID	Acceptance Criteria	Priority	Dev Days	Phase	Process
1	Log Tracker: User logs the Tracker into App			Total Dev Days	11	Completion Rates	61.54%
1-1	As a user I want to have a nice item logging user interface, so that I can easily log items in App.	1-1-1	Log item user interface should be simple, clear and easy to navigate.	Must	2	MVP	Done
1-2	As a user I want to add a picture of the item as a reminder of what it looks like.	1-2-1	User can take a picture of their item within App.	Must	2	MVP	Todo
		1-2-2	User can decide to use the taken picture or retake the picture.	Must	1	MVP	Todo
		1-2-3	User can decide to retake the picture if the photo is not good enough.	Must	0.5	MVP	Done
		1-2-4	App can store the picture in the mobile.	Must	0.5	MVP	Done
		1-2-5	User can choose to skip adding a picture.	Must	0.5	Final	Done
1-3	As a user I want to name my item, so that I'll know what the item is if I don't want to take a picture of it.	1-3-1	User can add the item's name.	Should	0.5	MVP	Done
		1-3-2	App can store the item's name.	Must	0.5	MVP	Done
1-4	As a user I want the app to tell me if my inputs are correct automatically, so that I don't need to revise them.	1-4-1	App should validate the item inputs.	Should	0.5	Final	Done
1-5	As a user I want to know app can actually detect the tracker, so that I will feel safer that it actually works.	1-5-1	App should send a request to register Tracker.	Must	1	MVP	Todo
		1-5-2	Tracker should receive request and register App.	Must	1	MVP	Todo
		1-5-3	App should be able to store Tracker data.	Must	0.5	MVP	Done
		1-5-4	App should activate Tracker and make it send the location to App.	Must	0.5	MVP	Todo
2	Monitor Tracker: App listens to the signal of Tracker			Total Dev Days	22	Completion Rates	0.00%
2-1	As a user I want the app to track the item's location all the time, so that I have a record of its location once it's lost.	2-1-1	App listens for signal at regular intervals.	Must	3	MVP	Todo
		2-1-2	Tracker send position data to App at regular intervals.	Must	3	MVP	Todo
		2-1-3	App saves the history of the position of Tracker at regular intervals.	Must	1	MVP	Todo
		2-1-4	App should be able to upload the history of the Tracker's position to server at regular intervals.	Must	1	MVP	Todo
2-2	As a user I want the tracker to send the accurate location data to app, so that I can find it easily if it's correct..	2-2-1	Tracker uses bluetooth mode while close to User.	Must	7	Final	Todo
		2-2-2	Tracker uses cellular mode while is far from User.	Must	7	Final	Todo
3	Track Item: User uses App to see the tracking list and track Item.			Total Dev Days	29.5	Completion Rates	50.00%
3-1	As a user I want to view the tracking item list, so that I can view my item and look up more detail if I want.	3-1-1	Item list user interface should be simple, clear and easy to navigate.	Must	2	MVP	Done
		3-1-2	App should get item data from mobile database	Must	1	MVP	Done
		3-1-3	App should fetch lastest locations of all the tracking items when User visit item list page.	Should	1	MVP	Todo
		3-1-4	App should display if any of Items is lost clearly.	Must	2	MVP	In Progress
3-2	As a user I want to activate or deactivate Tracker, so that I can save my mobile and the tracker's batteries.	3-2-1	User should be able to activate/deactivate Tracker to stop tracking.	Should	2	Final	Todo
3-3	As a user I want to see the location history in a map format of the item, so that I can track/look for it if it's lost.	3-3-1	Item history user interface should be simple, clear and easy to navigate.	Must	2	MVP	Done
		3-3-2	User should be able to navigate to from item list to the item history.	Must	0.5	MVP	Deprecated
		3-3-3	App should display the Item and User's location in a map format.	Should	4	Final	Done
		3-3-4	User should be able to see Item's location on the map in at different time.	Should	2	Final	Done
3-4	As a user I want to navigate me to my item, so that find it quicker.	3-4-1	App should link to Apple Map to navigate User to Item's last location.	Should	3	Final	Done
		3-4-2	App should link to Google Map App to navigate User to Item's last location.	Should	1	Final	Todo
		3-4-3	App could point in direction of item using a compass and distance in meters for close range situations in which the item isn't clearly visible.	Could	2	Final	Todo
3-5	As a user I want to edit my item detail, so that I can change the tracking item.	3-5-1	User can rename item.	Should	1	Final	Done
		3-5-2	User can retake the photo of the item.	Should	1	Final	Done
3-6	As a user I want to delete the item, so that I can stop tracking the item for good.	3-6-1	User can delete the item.	Should	1	Final	Todo
		3-6-2	App should delete all the data related to the item.	Should	2	Final	Todo
		3-6-3	App should deactivate the Tracker.	Should	2	Final	Todo

User Story ID	User Story	Backlogs					
		Backog ID	Acceptance Criteria	Priority	Dev Days	Phase	Process
4	Notification: User should receive notification if the item is lost			Total Dev Days	4	Completion Rates	0.00%
4-1	As a user I want to get notification if the item is far from me, so that I can retrieve before it's taken by someone else.	4-1-1	App should activate the OS notification function while it detects Item is too far.	Must	3	MVP	Todo
		4-1-2	Notification should be able to link to App.	Should	1	Final	Todo
5	Onboarding: User registers/logins a passowrd for logining the app			Total Dev Days	14.5	Completion Rates	88.89%
5-1	As a user I want to set password for the app, so that others can't see where's my item.	5-1-1	Password register user interface should be simple, clear and easy to navigate.	Must	2	MVP	Done
		5-1-2	User can register 4-digit password in App	Should	3	Final	Done
		5-1-3	User can use Touch-ID as login method.	Should	1	Final	Done
5-2	As a user I want to reset my password, so that I can prevent if someone has seen my password.	5-2-1	Setting page user interface should be simple, clear and easy to navigate.	Must	1	MVP	Done
		5-2-2	User should be able to reset the password in Setting page.	Should	3	Final	Done
5-3	As a user I want to turn off password, so that I can login quicker.	5-3-1	User can turn off password protecting feature in Setting page.	Should	1	Final	Todo
5-4	As a user I want to access the app, so that I can use it.	5-5-1	User can use 4-digit or Touch-ID as input to login.	Should	2	Final	Done
		5-5-2	App prevents User from accessing App if the login authenfication fail.	Should	0.5	Final	Done
		5-5-3	App redirect to item list if the login authenfication succeed.	Should	1	Final	Done
6	Physical Tracker			Total Dev Days	13	Completion Rates	0.00%
6.1	As a user I want to make sure that the actual hardware is working as expected	6-1-1	User should be able to switch on and off the tracker	Must	1	Final	Todo
		6-1-2	The tracker must always work when is switch on	Must	1	Final	Todo
		6-1-3	The tracker should locate itself and send the location in the Amazon Cloud Server	Must	1	Final	Todo
6.2	As a user I want that the battery to last enough long	6-2-1	The battery should to last for at least one day	Should	1	Final	Todo
		6-2-2	User should be able to charge the battery	Should	2	Final	Todo
		6-2-3	Users should be able to change the battery if it broke	Should	2	Final	Todo
6.3	As a user I want that my tracker does not brake if I drop it.	6-3-1	The case should be enough resisten to not brake on first drop	Should	2	Final	Todo
		6-3-2	The case archidecture should be made with harmless in mind	Should	3	Final	Todo
7	Data Security			Total Dev Days	3	Completion Rates	0.00%
7.1	As a user I want that sensitive data is kept restricted to authorized to access it.	7-1-1	Specify which data producted by the system is to be considered sensitive	Should	3	Final	Todo

A.2 Architecture Decision Records

A.2.1 Mobile Development

Column	Description
Title	Which mobile OS to work on the mobile application.
Context	....
Decision	...
Status	...
Consequences	...

Table 8: ADR Mobile Application Development

A.2.2 iOS Development

Column	Description
Title	Which language was used for developing the iOS mobile application.
Context	....
Decision	...
Status	...
Consequences	...

Table 9: ADR iOS Application Development

A.2.3 Android Application Development

Column	Description
Title	...
Context	....
Decision	...
Status	...
Consequences	...

Table 10: ADR Android Application Development

A.2.4 Tracker Development

Column	Description
Title	...
Context	....
Decision	...
Status	...
Consequences	...

Table 11: ADR Tracker Development

A.2.5 Build a customised server

Column	Description
Title	...
Context	....
Decision	...
Status	...
Consequences	...

Table 12: ADR Customised Server

A.2.6 Transfer iOS Development from Swift to React Native

Column	Description
Title	...
Context	....
Decision	...
Status	...
Consequences	...

Table 13: ADR Transfer iOS Development from Swift to React Native

A.3 Tasks Divided

A.4 Progress Tracking Form

WBS Code	Date	Status	Resource Name (Hours)						
			Chin	Dylan	Marian	Mifuad	Hussein	Thairan	Jheng-Hao
1-1	19/10/2017	Brainstormed and decided the subject topic.	1	1	1	1	1		1
1-2	22/10/2017	Planned and scheduled tasks and redesigned progress tracking form							3.5
1-3	23/10/2017	Built online survey - refined and ready to deploy		3	0.5		4		
1-3	23/10/2017	Deployed online survey, collecting data		3	3	1	1		
1-2	23/10/2017	Read references and drafted first persona							1.5
1-2	24/10/2017	Drafted the second and third of personas							0.5
4-2	24/10/2017	Lab session	1			1			1
2-1	24/10/2017	Drafted the user scenarios							0.75
2-2	24/10/2017	Competing Products (Market Research)				3			
2-1	25/10/2017	Improved the user scenarios and drew the use case diagram.							1.5
2-1	25/10/2017	Drew the diagrams.							2.75
2-4	26/10/2017	Regulation and Standards							2
2-3	26/10/2017	Academic Research							1.5
2-10	26/10/2017	Integrate all the reports together							1.5
1-2	26/10/2017	Stakeholders Report						2.5	
4-3	27/10/2017	Meeting with supervisor		0.5	0.5	0.5	0.5	0.5	0.5
4-4	29/10/2017	Recorded meeting notes							0.25
2-7	29/10/2017	Emailed Pete to ask for a meetup							0.5
2-7	29/10/2017	Research on Hardware to use for the sensors		2	1				
2-7	29/10/2017	RFID and NFC research							1
2-7	30/10/2017	Schedule a meeting with Pete							0.25
4-2	31/10/2017	Lab session	1	1	1	1	1		1
2-7	31/10/2017	Meeting with Pete			1	1			1
2-7	31/10/2017	Estimote research					3		
2-10	31/10/2017	Product name survey					1		
4-1	31/10/2017	Divided and assigned the tasks			0.5	0.5			0.5
4-1	01/11/2017	Tracking tasks progress.							0.5
4-1	02/11/2017	Tracking tasks progress.							0.5
2-10	02/11/2017	Integrate all the reports together							0.5
2-5	02/11/2017	Search and select project management tool							1
4-3	03/11/2017	Meeting with supervisor		0.5		0.5	0.5	0.5	0.5
2-7	03/11/2017	Physical lab induction							2
2-6	09/11/2017	Conceptual Prototype Interviews							4
2-6	09/11/2017	Conceptual Prototype Report							1
4-1	09/11/2017	Set up Github Repository							0.5
4-1	10/11/2017	Move reports to Github Repository							2
4-2	14/11/2017	Lab session	1	1		1	1	1	1
2-7	14/11/2017	Prepare Arduino tutorial kits					2.5		
4-1	14/11/2017	Updated repository and readme							1
2-5	15/11/2017	Planning the project (Sub tasks, Major tasks and Milestones)						3.75	
4-3	17/11/2017	Meeting with supervisor			0.5	0.5	0.5		0.5
2-5	19/11/2017	Drew gantt chart and improved the project plan							3
2-7	20/11/2017	Required Training Project 0 and 1.		2	2.5	1			
2-9	20/11/2017	Project Prototype		2			12		
2-5	20/11/2017	Project Sequence Diagram							2
4-2	21/11/2017	Lab session	1	1	0.5		1	1	1
2-9	23/11/2017	Project Prototype Report			3		7		
2-1	23/11/2017	Technical Architecture Research							1
4-3	24/11/2017	Meeting with supervisor		0.5			0.3		0.5
4-2	28/11/2017	Lab session	1	1		1		1	1
2-10	28/11/2017	User Need Overview & Concept Introduction Draft		9.5					
2-7	29/11/2017	Arduino GPS						2.25	
2-1	29/11/2017	Architecture Diagrams							3
4-3	24/11/2017	Meeting with supervisor		0.5		0.5	0.5	0.5	0.5
4-1	02/12/2017	Updated github work flow & convention							1.25
2-7	03/12/2017	Arduino - RFID				3			
2-10	03/12/2017	Prepare presentation					0.4		1
4-2	05/12/2017	Lab session	1	1		1	1	1	1
-	05/12/2017	Resource request meeting		1.5		1.5	1.5	1.5	
2-10	05/12/2017	Final Proposal				1			2.5
4-3	08/12/2017	Meeting with supervisor		0.5		0.5	0.5		0.5
2-1	08/12/2017	Update use cases					3		3

WBS Code	Date	Status	Resource Name (Hours)						
			Chin	Dylan	Marian	Mifuad	Hussein	Thairan	Jheng-Hao
2-10	09/12/2017	Evaluation Plan - Proposal	3.5						
2-10	09/12/2017	Proposal structural research		2					
2-10	09/12/2017	Final Proposal							4.5
2-7	09/12/2017	Estimote research			3				
2-10	10/12/2017	Final Proposal	3						6.5
2-10	10/12/2017	Final Proposal				6			
2-10	11/12/2017	Proposal reviews	1	1					
2-10	12/12/2017	Final Proposal		2	2			2	
4-2	12/12/2017	Lab session	1	1	1	1	1	1	
2-10	13/12/2017	Final Proposal	5	5	4	5		1.5	5
2-10	14/12/2017	Final Proposal	4	4		4	3		6
4-1	26/12/2017	Project cleanup					0.4		2
3-7	28/12/2017	iLost logo					2		2
-	28/12/2017	Computer fair prep - posters					9		5
3-7	31/12/2017	(ongoing) iLost app user interfaces redesign							7
3-9	01/01/2018	(ongoing) iLost app front-end development							9
-	08/01/2018	Computer fair prep - posters retouching							3
-	11/01/2018	Computer fair (severe travel issues)					2	2	2
3-9	12/01/2018	(ongoing) iLost back-end development					9		
4-1	15/01/2018	Set up back-end repo and team			0.2		0.3		
4-2	16/01/2018	Lab session	1	1	1	0.5	0.5	0.5	1
4-1	16/01/2018	Refactor project tasks and retouch use cases.							1.5
3-7	16/01/2018	(ongoing) iLost back-end development			2.75		5		
3-7	17/01/2018	iLost iOS app user interfaces design							5
3-8	18/01/2018	iLost back-end Set up with Rapsberry and Nova			3		2		
4-3	19/01/2018	Meeting with supervisor				0.3	0.3	0.3	0.3
4-1	22/01/2018	Write the user stories, backlogs and acceptance criteria.		0.5					2.5
-	23/01/2018	3D printing induction							3
4-2	23/01/2018	Lab session	1		1	1	1		
-	23/01/2018	Meeting with tech support			1		1		
4-1	23/01/2018	Setup Sprint structure							1.5
3-8	24/01/2018	Short-range devlopment (puck.js)					9		
4-3	26/01/2018	Meeting with supervisor		1	1		1		1
3-8	26/01/2018	iLost long range					1.2		
3-8	26/01/2018	iOS app develop - add a new Item view							4.5
3-8	29/01/2018	iOS app develop - item list view							5.5
3-8	29/01/2018	iLost long range					8		
4-2	30/01/2018	Lab session		1	1	1	1	1	1
3-8	30/01/2018	Android app development - Welcome screens		4					
	30/01/2018				3				
	30/01/2018					2			
3-8	30/01/2018	app endpoint fixed					3		
3-8	30/01/2018	iOS app develop - item list view							3
3-8	31/01/2018	iOS app develop - item list view							3.5
3-8	31/01/2018	fixed ilost-long range					4		
3-8	01/02/2018	iOS app develop - item detail view							0.75
4-3	02/02/2018	Meeting with supervisor		0.5	0.5	0.5	0.5	0.5	0.5
3-8	02/02/2018	iOS app develop - item detail view & discuss extending user stories							2
3-8	02/02/2018	Android app development - Main activity navigation drawer		3		2			
3-8	04/02/2018	Android app development - Login screen lockpattern		4.5					
3-8	05/02/2018	Android app development - Learning android studio				1.5			
3-8	06/02/2018	Android app development - item list		2		1			
3-8	06/02/2018	update API & back-end script					2.5		
4-2	06/02/2018	Lab session		1	1	1	1	1	1
3-8	06/02/2018	iOS app develop - Amazon S3 API intergrate							4.5
3-8	07/02/2018	iOS app develop - Amazon S3 API intergrate							1
4-3	09/02/2018	Meeting with supervisor					0.5	0.5	0.5
3-8	07/02/2018	iOS app develop - Amazon S3 API intergrate							4.5
5-5	13/02/2018	Final Report ch4 plan					2		
3-8	15/02/2018	Android app - item list		2					
3-8	17/02/2018	Android app - itemlist screen development & testing		3		2			
3-8	20/02/2018	iOS app develop - item detail view, load location							4.5

WBS Code	Date	Status	Resource Name (Hours)						
			Chin	Dylan	Marian	Mifuad	Hussein	Thairan	Jheng-Hao
3-8	20/02/2018	Android app - tracking activity, google maps api implementation		2		1			
3-8	20/02/2018	iOS app develop - map api, data parsing							3
4-2	20/02/2018	Lab session		1	1		1	1	1
3-8	21/02/2018	Android app		4		2			
3-8	21/02/2018	Tracker - Intergrate raspberry pi zero			4				
3-8	21/02/2018	Tracker - Intergrate raspberry pi zero					4		
3-8	21/02/2018	iOS app develop - UI storyboard							4
5-5	27/02/2018	FInal Report ch3 plan							2
5-3	28/02/2018	Make presentation slides							3
3-10	01/03/2018	Usability Tests: research							4
5-5	03/03/2018	FInal Report section 1 plan		1					
3-8	03/03/2018	iOS app develop - Migrated to React Native							7
3-8	04/03/2018	iOS app develop - Authenticate Views							9
3-8	05/03/2018	Make the tracker demo case							2
3-8	05/03/2018	iOS app develop - add item view, edit item view, data storage management							5
5-3	06/03/2018	presentation script pt. 5				1.5			
5-5	06/03/2018	Final report conclusions structure/plan				2			
3-10	07/03/2018	Usability Tests: make cosent form, structure test record table, conduct tests							5
5-5	07/03/2018	Final presentation: update slides - test & evaluation and production pages.							1
3-8	08/03/2018	iOS app develop - bugs fixed, details in commit message							2
3-10	08/03/2018	Usability Tests: conduct tests							1
5-5	08/03/2018	Final presentation: update slides - test & evaluation and production pages.							2.15
5-5	08/03/2018	report structure/writing				3			
4-3	09/03/2018	Meeting with supervisor		0.5		0.5	0.5	0.5	0.5
5-5	14/03/2018	Final report: Formative Evaluation - mobile application test							4
5-5	14/03/2018	Final report: Development Record - Technology selection, agile development							4
5-5	13/03/2018	report writing/reading other points				3			

B Formative Evaluation

B.1 consent Form

Usability Test Consent Form

Please read and sign this form.

In this usability test:

- You will be asked to perform certain tasks on a provided mobile phone.
- We will also conduct an interview with you regarding the tasks you performed.

Participation in this usability study is voluntary. All information will remain strictly confidential. The descriptions and findings may be used to help improve the iLost Project.

However, at no time will your name or any other identification be used. You can withdraw your consent to the experiment and stop participation at any time.

If you have any questions after today, please contact Jheng-Hao Lin at jlin015@gold.ac.uk

I have read and understood the information on this form and had all of my questions answered

_____ _____

Subject's Signature Date

Thank you!

We appreciate your participation.

C Design and Implementation

D Quality Assurance

D.1 Black-box Test Cases

ID	Name	Goal	Preconditions	Success End Condition	Failed End Condition	Trigger	Normal Flow	Alternative Flows	Frequency of Use	Assumptions
1	Add item view: user interface	Display the a view title, camera icon, text field and two buttons in the view.	The user access to the applicaion and is logged in. The user taps the plus button in the item list view.	The correct elements(a view title, camera icon, text field and two buttons) are displayed same as the UI design.	Components are not displayed properly.	Login in the application	1. User is in the item list view 2. User taps the plus button in the navigation bar 3. Application displays the add new item view.	N/A	Rare	N/A
2	Add item view: camera or choose pictures	Display the a view title, camera icon, text field and two buttons in the view.	The user access to the applicaion and is logged in. The user is in the add item view.	After the camera icon is tapped, the application displays two options to select: "Select from photo library" and "take a new photo".	Nothing happens after the camera icon is tapped or nothing triggered after any of the two options is tapped.	The camer icon is tapped.	1. User is in the add item view. 2. User taps camera icon. 3. application trigers a action menu where user can have two option to select.	N/A	Rare	N/A
3	Add item view: trigger camera application	Trigger the native camera application once the camera icon is tapped.	The user access to the applicaion and is logged in. The user is in the add item view.	The camera application is triggered successfully and ready to take a picture of the item.	The camera application is not triggered when the camera icon is tapped.	The camer icon is tapped then the "take a new photo" option is tapped.	1. User is in the add item view. 2. User taps camera icon. 3. application trigers a action menu where user can have two option to select. 4. Uesr taps the "take a new photo" option.	N/A	Rare	N/A
4	Add item view: retake photo	After taking a photo, there should be an option for using it and another option for retaking it.	The user access to the applicaion and is logged in. The user is in the add item view and native camera application is triggered	After a photo is taken, the native camera application displays a button "retake" in the left bottom corner and another button "OK" in the right bottom corner. If the "retake" button is selected, current photo will be erased and enable user to take another photo. If the "OK" button is selected, application will pass the photo to the next view.	The native camera application does not display correct two buttons after a photo is taken.	A photo is taken in the triggered camera application.	1. User is in the native camera application triggered in the add item view. 2. User takes a photo of the item. 3. application displays the taken photo, an button with "Retake" on it and another button with "OK" on it.	1. If the "Retake" button is tapped, the application goes back to previous stage and let the user to take another photo. 2. If the "OK" button is tapped, the application goes to next stage which should be display the photo on the add item view.	Rare	N/A
5	Add item view: store photo	After user decides to use the taken photo, it should store a copy in the photo library.	The user access to the applicaion and is logged in. The user is in the add item view and native camera application is triggered	After a photo is taken and selected to used, the taken photo will be stored in the photo library.	The native camera application does not store the photo in the photo library.	A photo is taken and selected to use in the triggered camera application.	1. User takes a photo and selects it in the native camera application triggered in the add item view. 2. The application stores the taken photo in the photo library.	N/A	Rare	N/A
6	Add item view: choose photo from camera roll	User can browse the native photo application to select an desired photo, rather than taking a new photo.	The user access to the applicaion and is logged in. The user is in the add item view.	The native camera roll API is triggered and user the select any photo from the existing albums. The selected photo will be passed to the next view.	The camera roll API is not triggered or a photo URI can not be passed to the next view.	The camer icon is tapped then the "selecte from photo library" option is tapped.	1. The application trigers a action menu where user can have two option to select in the add item view. 2. Uesr taps the "select from photo library" option.	1. application goes back the add item view if user cancel to select a photo from the photo libarary.	Rare	There are already taken photos in the photo libaray.

ID	Name	Goal	Preconditions	Success End Condition	Failed End Condition	Trigger	Normal Flow	Alternative Flows	Frequency of Use	Assumptions
7	Add item view: not add photo	application should display a placeholder picture if no photo is choosen.	The user access to the applicaion and is logged in. The user is in the add item view.	application will not prevent user to the next stage if the user save a new item without choosing a photo.	User cannot go to the next stage if the user does not use any photo.	User taps save button without taking a photo or selecting a photo from the photo library.	1.User is in the add item view. 2. User taps the save button with the validated item name input but without using any photo.	N/A	Rare	N/A
8	Add item view: add item's name	application should be display display an editable text field in the add item view, which enable user to input the text item's name.	The user access to the applicaion and is logged in. The user is in the add item view.	The text input field in the add item view is editable, user can edit the text value of it.	The text input field in the add item view is not editable, user cannot edit the text value of it.	Uesr taps the text fields in the add item view.	1. User is in the add item view. 2. The text input field displays correct placeholder text: "Enter item's name" 3. User taps the text input field. 4. The application triggers the keyboard. 5. User can input the item's name.	N/A	Rare	N/A
9	Add item view: store item's name	After the save button is tapped, ther application will store the value of the text input field as the item's name in the application.	The user access to the applicaion and is logged in. The user is in the add item view and has typed the item's name in the text field.	After the save button is tapped, the value of the text input field will be stored in the application successfully.	After the save button is tapped, the value of the text input field is not stored in the application/	Uesr taps the text fields in the add item view.	1. The text input field in the add item view is edited by the user with a validated name. 2. User taps the save button. 3. The application saves the value of the text field in the application.	N/A	Rare	The text input value is validated.
10	Add item view: validate inputs	After the save button is tapped, ther application will validate the input's value. The values will only be saved f the inputs are validated.	The user access to the applicaion and is logged in. The user is in the add item view.	After the save button is tapped, the application will validate all the inputs, including the photo and the text field. If the they are all validated, then the data will be saved. Otherwise the application will display the error message to prompt the user to input the validated data.	The application is not able to validate the input, such as the input length of the item's name.	Uesr taps the save button in the add item view.	1. User is in the add item view. 2. User taps the save button. 3. The application validates the input fields.	1. If all the data are validated, the application will save the data in the application, then return to the item list view. 2. If any of the data is invalidated, the application will display an error message to ask user to input correctly.	Rare	N/A

ID	Name	Goal	Preconditions	Success End Condition	Failed End Condition	Trigger	Normal Flow	Alternative Flows	Frequency of Use	Assumptions
11	App registers tracker	System can store a unique id of the tricker and bound it with the item in the application	The user access to the applicaion and is logged in. The user is in the add item view and validated inputs of the item are saved.	The system has saved the validated inputs of the item and the application sends a request to pair the tracker with this item via bluetooth.	The system is unable to send any request to the tracker.	Uesr taps the save button in the add item view with validated inputs , and the system has saved all the input data in the application.	1. The application has saved the item's data . 2. The application send a request to register the tracker.	1. If the bluetooth sensor is turned off, prompty user to turn it on.	Rare	N/A
12	Tracker receives register request from App	The tracker should able to receive and read the request from the application via bluetooth.	The application has sent a request to register the tracker with bluetooth.	The server hosted on the tracker is able to read and interpretate the register request sent from the application	The server hosted on the tracker is not able to read and interpretate the request sent from the application	The application has sent a register request to the nearby tracker.	1. The application send a request to register the tracker. 2. The tracker received the request and read the data in it. 3. The tracker pairs itself with the application via the data in the request.	N/A	Rare	N/A
13	App stores tracker data	The application stores the tracker's special ID and pair it with the item's data as a reference.	The tracker has received a register request from the tracker and already registered itself with the application.	The application stores an unique ID of the tracker and pair it with the item.	The application cannot store any data of the tracker.	The tracker stored the application's registered data and sends back a registering succeed response.	1. The tracker pairs itself with the application via the data in the request. 2. The application stores the tracker's unique ID in the storage.	N/A	Rare	N/A
14	App activates the tracker	After pairing the tracker, the application can activate the locating function via bluetooh.	The tracker is paired with the application.	The application sends a bluetooth request to the paired tracker, which enables the tracker starts the locating service and send data to the S3 database.	The application cannot activate the tracker's locating service at all	For the first time that the tracker has been paired with the application and the application has been paired with the tracker.	1. The tracker has finished the pairing routine with the application. 2. The application has finished the pairing routine with the tracker. 3. The application send a request to the tracker to activate its' locating service. 4. The tracker receives the request and starts the locating services.	N/A	Rare	N/A
15										
16										
17										
18										
19										
20										
21										

ID	Name	Goal	Preconditions	Success End Condition	Failed End Condition	Trigger	Normal Flow	Alternative Flows	Frequency of Use	Assumptions
22										
23										
24										
25										
26										
27										
28										
29										
30	Item details view: trigger Apple Map	The application should be able to use the item's location data, pass it to the Apple Map application, and trigger the nativation function and lead the user to the item's location	User is in the item details view.	The Apple Map application is triggered after the user taps the navigate button. It gets into the navigation mode and lead user to the location based on the distance. If the user is closed to the item, then it will navigate the user to the item in "walking" mode. If the item is far from the user it will be in the "transit" mode	The Apple Map application is not triggered after the navigate button is tapped. Or the Apple Map application is triggered but it does not receive the correct location data of the item and activate the navigate mode either.	User taps the navigate button	1. User taps the navigate button. 2. The application triggers the Apple Map application and passs the geolocation of the item to it. 3. Apple Map automatically activates navigation mode and lead user to the item.	N/A	Often	There are item's location data stored in the application already
31										
32										
33										
34										
35										
36										
37										
38										
39										
40										
41										
42										
43										
44										
45										
46										
47										
48										