World Wide Wonders

Bartsch Liam
Computer Science
Trinity College Dublin
Dublin, Ireland
bartschl@tcd.ie

Farrell Oisin
Computer Science
Trinity College Dublin
Dublin, Ireland
farreloi@tcd.ie

Gupta Sanil
Computer Science
Trinity College Dublin
Dublin, Ireland
guptas1@tcd.ie

Hartshorn Sam
Computer Engineering
Trinity College Dublin)
Dublin, Ireland
hartshos@tcd.ie

Kaushik Tanmay
Computer Science
Trinity College Dublin
Dublin, Ireland
kaushikt@tcd.ie

***Abstract*—This paper focuses on Bicycle security and Internet of Things applications for Bike locks. During this project, we have developed, tested, and implemented a connected Internet of Things Bike lock as well as an application, database, and bike tracker to complete the system. The larger applications of this system can be hugely beneficial for the individual cyclist as well as the city as a whole.**

***Index Terms*—Bicycle, Lock, Security, Internet of Things**

I. Introduction

Bike theft is a rampant problem in large cities like Dublin, Cambridge, Berlin, New York City, and Delhi. As our society becomes more and more technologically advanced and our cities become more interconnected, there's a perfect opportunity to make them safer as well. Our approach to this was implementing an Internet of Things Bike Lock.

There hasn't been a lot of research into the Internet of Things Bike Locks, however, the little research that has been done [1] shows that there is a gap in the security aspect of bike locks. In this instance, they generated authentication between a smart lock and a phone during the travel period. While this isn't the exact problem that our system aims to fix it's in the same domain.

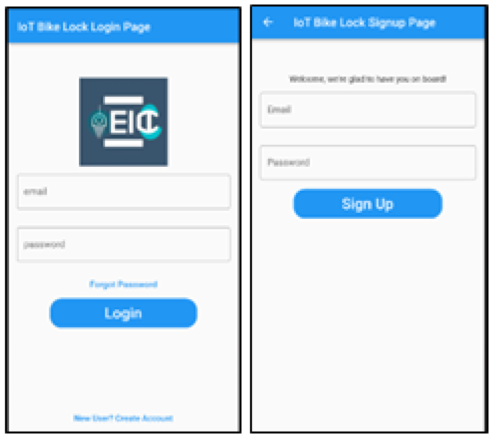
II. Implementation

We started out with a simple model of our bike lock on a breadboard. This basic model consisted of:

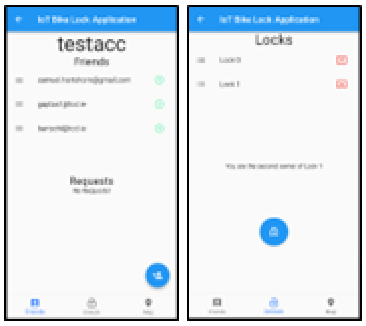
- an LED
- Two ESP32 Micro-controllers
- 9V Battery
- Power Supply MB102
- GT-U7 GPS Module
- Bike lock
- Servo Motor

First, we focused on getting the wiring right so that the LED would light up. In parallel to this we started working on the Flutter Application.

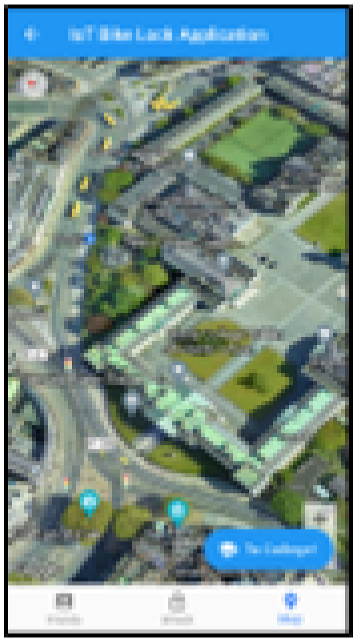
The underlying motivation for incorporating the Flutter Application component into this project was based on the premise that the majority of IoT products are accompanied by an application to enhance the convenience and overall user experience. In light of this, our aim was to design a straightforward yet aesthetically pleasing application to complement our IoT bike lock.



The initial stage of the application involved the authentication process, wherein users were required to either log in or sign up if they lacked an account. The login and sign-up interfaces are depicted in Figure 1. The incorporation of user accounts in the application was necessitated by two primary factors. Firstly, it was implemented as a safety measure, as safeguarding user privacy was deemed to be of paramount importance. By ensuring that only authorized users could access the application, the likelihood of covert activities was greatly reduced. Secondly, the inclusion of this feature lends a sense of authenticity to the application, given that most contemporary applications mandate user accounts as a prerequisite for access.

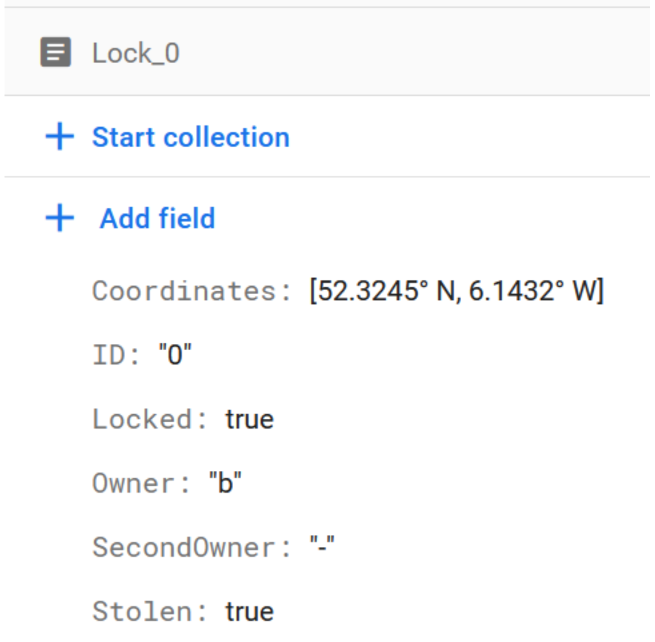


Upon a user's successful sign-in, they are directed to the main page, where an overview of the available locks in there is presented. Subsequently, the user can reserve a lock, thereby obtaining control over it. Once the user has control of the lock they may lock and unlock it at their discretion. Notably, a user may choose to share a reserved lock with a friend to enable the latter's use. To effect this sharing, the user must first make a friend request to that user. Upon the other user accepting the request, they can send the reserved lock to their friend. These pages are depicted in Figure 2.



An additional feature of the application is the map feature. Powered by Google Maps API the map feature allows users to view the locks on the map in real-time. This feature was mainly to increase user satisfaction with the application.

For storage, we decided to use Firebase since it was easily integrated into our flutter application and it allowed us to easily connect to it from the ESP32. Then we focused our efforts on building out each individual component. The first component was getting the ESP32 to turn on the LED if the 'Locked' variable in the database was set to 'true'. Each lock has several properties that can be seen below:



Lock_0

+ Start collection

+ Add field

```
Coordinates: [52.3245° N, 6.1432° W]
ID: "0"
Locked: true
Owner: "b"
SecondOwner: "-"
Stolen: true
```

- The Coordinates are comprised of two data points, one representing the latitude and the second the longitude. This helps the user locate where each lock is on the map and is updated if the lock is moved.
- ID is the identifier for each lock and is unique to each lock.
- Locked is a Boolean state value that determines whether the lock should be in the locked/unlocked position and depending on what the value for this variable is, it will activate the servo motor to lock/unlock the bike lock.
- Owner is a variable to determine who the owner of the lock is, which means that if you're not the owner you can't unlock/lock the bike, to become the owner, there must not be a current owner and you need to be able to reserve the lock.
- SecondOwner is similar to Owner where it allows you to lock/unlock the lock, but it's meant to be used by friends of the user only with the owner's permission.
- Stolen is a variable that determines whether or not the bike has been stolen from that lock and is activated as soon as the bike is removed from the lock while the lock is still in the locked position.

Once we had the ESP32 connected to Firebase and read the state of the locked variable and turned the LED on/off accordingly, we then focused on the next task which was having a Servo motor lock/unlock.

Once we had the servo motor working we then focused our efforts on the Bluetooth aspect of our lock. This meant that we would need to communicate from the bike lock to the bike (both represented as two ESP32s). After several trial and error attempts, we settled on using Bluetooth Low Energy as

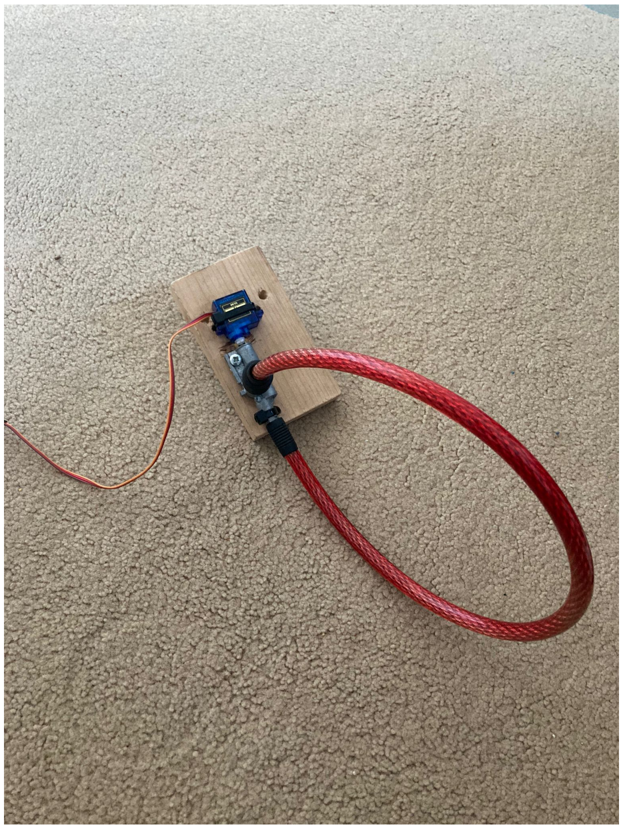


Fig. 1. Bike Lock

any other attempt crashed the memory. As we weren't getting very far with the Bluetooth, we decided to focus on the GPS module until we could get the rest working. Getting the GPS to work was also difficult and we didn't manage to successfully implement this part at the time. So we then decided to hard-code the coordinates and go back to work on the Bluetooth part of the project. After numerous failed attempts we successfully managed to connect the two ESP32s together. This was great news as it was another step in the right direction after our failed attempts with the GPS. Now that we had a working state-getter and Bluetooth connection we proceeded to focus on the security element of the project. Say a bike thief arrives and steals the bike. We needed a way to trigger an alarm and be able to detect that use case. For this, we planned on detecting when the ESP32 Bike was disconnected from the lock meaning the thief had stolen the bike and the lock was still locked. This would ideally change a Boolean state in the database which would then let the user know. Getting the ESP32 to connect to the database and read a value was pretty straightforward, however, getting it to modify a state was a more complicated task than expected. After some tweaking, we managed to get the ESP32 to update the database state if the Bluetooth connection ended and the lock was in the locked state. Once we had this success we decided to revisit the GPS problem. Thankfully this time we had a new GPS module that worked and actually gave us good data. Now all that was in place the only thing left to do was to integrate it all together and film the demonstration. There were a few more issues that occurred when trying to integrate each component together but eventually we managed to successfully implement the demo and record it. Once we had the demo finished we then started writing the report as well as trying to fix small bugs that occurred during the project that we didn't have time to fix at the time.

III. Challenges

1) One challenge was learning basic hardware, wiring, and Arduino coding concepts since these are not areas in which we had lots of experience. To work through this challenge we had to watch hours of tutorials on YouTube and then try to implement similar components with the hardware we were given. This was a steep learning curve but the more familiar we became with the hardware the easier it got. Looking back now, it seems incredible that we were able to achieve what we set out to achieve as well as able to fix problems as they arose without having been taught about these specific problems in class.
2) One issue we had was with the GET request to the Firebase database. The issue we were having was that the ESP32 didn't seem to be able to make the request despite the help of the HTTP library. So to solve this challenge we had to first write a Python script that made the request to make sure we were setting the right headers as well as the right URL. Once we confirmed that it was working with Python we then double-checked to make sure it worked with ReqBin an online tool that helps make HTTP requests. It was a bit tricky to figure out the correct URL and headers especially since we weren't familiar with the correct authentication methods to read database values from FireStore. Eventually, we did succeed in making the requests from both Python and ReqBin and then implemented it in Arduino so the ESP32 could do it as well. It was such a joyful moment when we finally saw it working on the ESP32 and this was a huge win that helped propel us forward to the next goal.
3) Our next challenge was choosing the appropriate database. At the start, one of us had started developing the code for the Real-time database as that was what most tutorials used. However, the app used the Firestore (the other database) which meant that we had to update one of the implementations to switch and use the other database. After some deliberation we decided to switch over to the Firestore database, however, at a later date, we then realized it would have been easier to use the Real-time database. But at the time we realized that it was too late to switch everything so we ended up just using Firestore. To prevent this from happening in the future we would communicate in advance about which database to use as well as make it easily changeable.
4) The next issue is a known problem with ESP32s. Because they only use a single-core CPU, it's unable to

perform simultaneous actions such as Bluetooth Low Energy connection and HTTPS requests. This was a problem because we needed to connect to the bike and if disconnected we needed to update the Stolen variable in the database. To get around this issue we ended up pausing the Bluetooth connection if it was disconnected, then having a continuous while loop that tries to make the PATH request until it succeeds, and only when it succeeds then restart the Bluetooth server connection. This sometimes works and sometimes doesn't, and we didn't fully solve this issue, given more time we could have potentially investigated memory allocation and gotten a working solution.

5) The next issue was a simple space issue where we needed to connect wires on both sides of the ESP32 but the board only had 5 rows of pins and we needed 6 since the distance of the ESP32 pins was 4 rows of holes. To get around this issue we added another board and had to slightly bend the ESP32 pins so that they fit on both boards.
6) Another issue we encountered was the small amount of memory storage that the ESP32 contained. For some reason, the whole program would crash when a Bluetooth connection was made, and we assume the issue was because it couldn't handle all the simultaneous operations all at once however we were never able to confirm this with 100% certainty. To fix this we just need to restart the ESP32 by pressing the reset button once it's started, sometimes it works sometimes it doesn't and we weren't able to fully understand why.
7) The next biggest issue we had was with the GPS. At first, we had three different GPS modules. When connecting them we were able to get them to power on and read the data but for some reason none of them connected to satellites meaning we couldn't get any GPS data at first. We tried all sorts of different potential solutions such as putting it on the roof outside, going to different locations, and reviewing each line of code to understand why it wasn't working. Looking back now it could have been a simple wiring issue. Thankfully we received a new GPS module near the end of the project and we were successfully able to connect it and use it.
8) The next issue was with authentication to the database. Because Firebase is from Google we needed a way to authenticate the users and the ESP32 to access the database. However, this proved difficult as getting access to the database meant having to send a request to get a token which could then be used to verify the device and then we would be able to get the API authentication. However, it always expired after a while and we weren't able to open a web page and verify the device on the ESP32 so we had to find a different solution. To implement our system in the real world we would have to assign the ESP32 a user and use that for each lock. But we didn't have enough time to figure out the Google authentication needed for that so we had to leave the database open for the demonstration.
9) The last issue we encountered was that the App stopped working on real and simulated iOS devices. This meant that for some of the features we had to revert back to a previous commit where we knew it worked. We believe this issue was linked to the use of the Firestore database but because of the lack of time, we weren't able to fix it in time.

IV. Conclusion / Results

Overall the implementation of our smart bike lock was a success as our prototype was functional and the proof of concept could be used to develop an existing system in the real world. The results from this prototype show that some improvements can be made to the modular components such as the ESP32, database authentication, and the servo motor. If these challenges are solved then our system is a good candidate to be scaled up for a whole city. That being said, realistically because of budget constraints it doesn't seem likely that people will be ready to invest in upgrading their bike lock security since most people already have a lock. So the way to convince them would be to show much more security and ease of mind our solution offers.

All in all the whole project was a very great introduction to the world of the Internet of Things and it helped us reinforce problem-solving skills we already had and teach us new skills such as hardware development, communication skills, and integration management.

The final project looked like this:

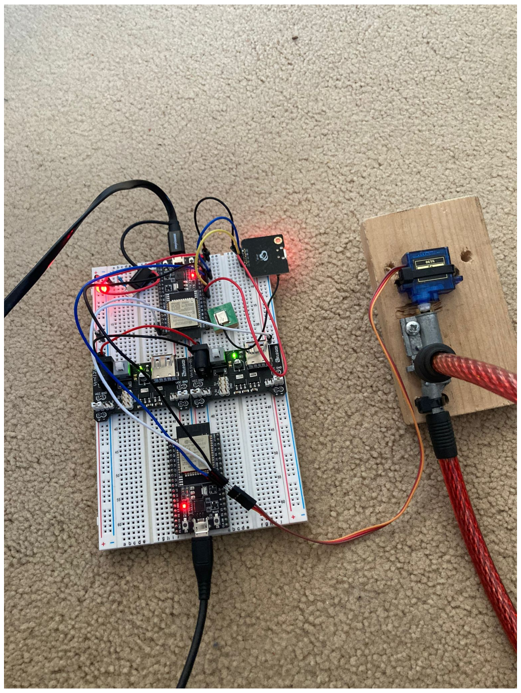


Fig. 2. Final Lock System

References

[1] A. Arno, K. Toyoda and I. Sasase, ”Accelerometer assisted authentication scheme for smart bicycle lock,” 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), Milan, Italy, 2015, pp. 520-523, doi: 10.1109/WF-IoT.2015.7389108.