Smart Offices and Event Participation Tracking with Internet of Things

Justin Alisauskas

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2018

Abstract

Internet of Things is one of the fastest growing markets, with more than 20 billion devices shipped over 2017, and a number expected to grow to around 75 billion by 2025.

The IoT solutions are used everywhere - health trackers (Fitbit), vehicle components status logging, parcel tracking or smart thermostat solutions. These small, low-power connected devices enable Internet-connected data monitoring solutions while still being efficient enough to function on battery power for days, months and in some cases - years.

In this project, we implement IoT solutions tailored for the use in the University of Edinburgh. We identify two areas within School of Informatics where IoT would bring the most benefits in terms of increased convenience or savings, and then create a working "Proof of concept" solution.

A Smart Office application is aimed at improving the experience for the professionals, by providing Twitter-enabled notes and SMS notifications at their office door. The Event Participation Monitoring solution is meant to replace the current tutorial presence marking at the university, by introducing RFID scanning based automated alternative.

Along with the explanation of final solution, we also present the IoT development process, pointing out the issues, limitations and challenges. We show the results of extensive evaluation done throughout the prototyping process, showing the gradual change of participants feedback as we progress with the IoT solution.

Acknowledgements

I cannot thank enough for my supervisor, Dr Paul Patras, for incredible support, encouragement and guidance that motivated to go an extra mile at every task I approached.

I would also like to thank my family, for giving me that additional push when it is needed the most, and being a voice of reason, when it is necessary.

Lastly, I want to thank my friends in Lithuania, for not forgetting me during these four years abroad; and my friends in Scotland, for the fantastic undergraduate experience in Appleton and Forrest Hill.

Table of Contents

1 Introduction **3**
- 1.1 Project Aims . . . 4
- 1.2 Project Outcomes . . . 6

2 Background **9**
- 2.1 Technical developments that allow IoT . . . 9
- 2.2 IoT in the University of Edinburgh . . . 13
- 2.3 Economical considerations . . . 14

3 Hardware Research **15**
- 3.1 Final hardware design . . . 15
- 3.2 Main IoT device board . . . 16
- 3.3 Internet connectivity . . . 23
- 3.4 RFID cards and tags scanning system . . . 26
- 3.5 Other peripherals . . . 30

4 Software Implementation **33**
- 4.1 Programming the Embedded System . . . 34
- 4.2 Cloud setup . . . 44

5 Evaluation **55**
- 5.1 System Performance Evaluation . . . 56
  - 5.1.1 IoT Device . . . 56
  - 5.1.2 Cloud API . . . 56
- 5.2 User and Market Study . . . 67
  - 5.2.1 Material for client studies . . . 67
  - 5.2.2 Interviews . . . 68
  - 5.2.3 Live presentations . . . 74
  - 5.2.4 Surveys . . . 78

6 Future work and Conclusions **87**

Bibliography **89**

Appendices **95**

Chapter 1

Introduction

IoT(Internet of Things) and IoE(Internet of Everything) applications are at the edge of innovation in the 21st century. The releases of the first smartphone for the masses (Apple iPhone, 2007 [20]), the tablet PC (Apple iPad, 2010 [19]), as well as the Internet-connected smart assistant (Amazon Echo [10], 2014), pushed for a breakthrough in our day to day life.

IoE devices - smartphones and tablets, gave the instant on-the-go availability of content such as emails, news, video calling and various other applications that require a data connection.

IoT applications (Amazon Echo [10] home assistant, Nest [40] thermostat, Fitbit health trackers [29]) enabled owners to improve their home environment(voice controlled lights, music, appliances), reduce expenses (smart electricity meters) and take better care of their health (Heart-rate monitoring, sleep tracking).

The changes are not restricted to the home environment and free-time activities. The biggest changes are in Education and Enterprise sectors - the places, where people spend most of their time.

In a business environment, the always available Internet connectivity enabled secure access to work from any place, at any time, making the physical presence in the office an option, not a necessity. At the same time, with every employee becoming reachable at all times, a demand of immediacy has become a norm, shifting the work schedule from 9-to-5 Monday-Friday, to a more scattered pattern of work with 24/7 availability. Adjusting to the environment, enterprises started trialling out IoT applications which would enable office adjustments for better performance. For example, Bank of America used employee badges that record posture and movement. They analysed collected data, connected it to business performance and modified the office space accordingly, yielding 10% productivity increase [45].

Significant changes are taking place in the education sector as well. Pupils and students have a chance to embrace the technological advancements in a much earlier stage of life (around 97% of people aged between 16-24 own a smartphone, compared to 85% of those aged 45-54 and 59% of individuals between 55-64 [65]). As a result, their expectation of immediacy is even more apparent.

In response to such cultural shift, Universities are already trialling IoT applications that aim to make campus environment more pleasant for students [53]. One of the first IoT studies in the University of Edinburgh was Main Library desks availability monitoring. The project was using an array of sensors to detect unused study spaces. Collected data was used to notify students on free space count reducing the amount of time that individual spent looking for a vacant table.

Even though enterprises and universities are showing the curiosity in IoT, they are trialling applications which would require large investments or a change of infrastructure, but do not commit to putting enough funding in place. As a result, investors do not even see the savings and added convenience which comes with large-scale IoT implementation. The IoT applications either get cancelled right after the pilot study, as it was the case with Bank of America [45]), or the initiators of the IoT solution do not invest enough to apply the product on a scale that would be large enough to make an impact, as it happened with the University of Edinburgh (Main Library desks monitoring is implemented only on one out of seven floors. Due to this, students do not even check the provided readings, as the amount of supplied information is not valuable enough to make it worthwhile).

1.1 Project Aims

In our project, we identify several areas within the University, which could benefit from IoT and then create a working 'proof of concept' solution. We are taking such initiative to further motivate large organisations (universities and enterprises) to take major steps in IoT innovation and build solutions that are cheap to implement, yet offer great convenience and/or financial benefits by targeting the most problematic areas in the company or education institution.

Our Smart Office IoT application is aimed at enhancing the experience for the professionals in the office, by providing a Twitter-enabled notes [69] and SMS notifications via Twilio [68]. This additional option for communication and planning is meant to benefit the employees who have ever increasing pace of work.

The Event Participation Monitoring solution aims to replace the existing way of presence marking with an automated alternative, by introducing RFID card scanning system integrated with an online database.

Our emphasis on development of functional and robust implementation is based on online sources analysis. A survey made by Cisco in 2017 states that "60 percent of IoT initiatives stall at the Proof of Concept (PoC) stage and only 26 percent of companies have had an IoT initiative that they considered a complete success. Even worse - a third of all completed projects were not considered a success." **??**. The mentioned statistics are applicable both to IoT hobbyist community and to professional IoT companies with sizable knowledge on software and hardware. The findings of Cisco survey motivated us to further create a better understanding of the proof of concept development phase - a main point of failure step, and identify the key issues by trying out the prototyping first-hand.

Our general aims are to:

1. Show the potential IoT applications have in improving environment and reducing expenditure around University campus and enterprise companies.
2. By creating a functional IoT implementation, provide an insight to the trends, problems and evolution of an IoT development.
3. Prove the IoT applications can have a higher success rate if the developer does in depth market analysis at the start, and listens to opinions of potential clients throughout the development process
4. Show that development of quality IoT prototype can be done with a low budget.

Following the whole prototyping process structure, our technical aims are set to:

1. Create a working IoT solution for Smart Office and Event Participation Monitoring applications.
2. Develop a Cloud setup which is able to handle multiple devices and provides scalable functionality for further improvements.
3. Research and apply the hardware optimization features (performance and power saving).
4. Design and implement a user friendly device packaging. (As this is one of prototyping process steps as well)
5. Evaluate the end result of hardware implementation and server side.

1.2 Project Outcomes

At the end, we have created a complete end-to-end IoT infrastructure which provides Smart Office and Event Participation Monitoring applications. Our development approach closely represented the process that IoT companies go through in their R&D departments during the prototyping stage for their own solutions. We faced many roadblocks, witnessed the fragmentation in the market of hardware modules, and in the technology standards that are available. As a result, around 70% of the time was spent on the hardware compatibility research, as well as debugging and fixing of code libraries that are described by manufacturers as completely plug-and-play solutions.

We succeeded in achieving our general aims, as we:

1. Showed that IoT solution would reduce the University expenditure, saving at least around £802.4 per single semester of one course by automating tutorial presence marking.
2. Presented two functional IoT solutions meant to improve the student and staff satisfaction. Furthermore, the data collected during the interviews, presentations and surveys showed public has a positive reaction to a prototype, and there is a sizable potential clientèle for a full-scale implementation of our prototype.
3. We developed two IoT applications that are considered successful both from technical and business perspectives. The result proved that analyzing the market, listening to feedback from potential customers and addressing expressed concerns in the early stages of prototyping leads to well received quality end product.

Just as in full scale IoT projects meant for mass production, our prototype development required to develop and apply knowledge from multiple disciplines [57]:

- For embedded device:
    1. Embedded Systems Programming - We did low level programming on a low-power board, to provided encrypted WiFi communication, handled the peripherals (showing information on lcd display, reading cards with RFID scanner, handling buttons input) and increased power efficiency via ESP8266 sleep mode [34].
    2. Electrical Engineering - We set up the wiring, soldered the modules, tested and chose power supply, enabled connections between different voltage devices, performed power consumption measurements.
    3. 3D Design and Fabrication - We created a 3D model of device enclosure and then printed it out with 3D printer.
    4. Human-Computer Interaction - We performed user studies, identified most important features and added design, functionality improvements to the prototype.

- For server side:
    1. Cloud server design - We set up AWC EC2 instance [4], installed and configured Apache HTTP Server [48], set up hosting, added HTTPS certificate [24].
    2. API development - We created custom Cloud API, enabling encrypted way of communication between a HTTP-only embedded device and the server. In addition, the API integrates all of the services provided by Twitter [69], Twilio [68], OpenWeather APIs [55](requiring HTTPS and OAuth), as well as our local ArrestDB database API [14].
    3. Database development - We created an SQLite [63] database and provided an access to it via an ArrestDB RESTful PHP API.
    4. Web development - We reacted to feedback given by University community and design a web interface to provide a live view of database entries.

With all of the work put into an embedded device and server side development, we managed to design and build a functional, scalable IoT infrastructure from the ground up. Our solution is capable of event participation monitoring - the IoT device is scanning student cards and communicating with a Cloud API to check the presence. For this feature we also added additional functionality - if a person arrives to a wrong event, device display shows location and details of the event to which an individual was supposed to come. We also have come up with a working Smart Office solution, with a Twitter feed and additional capabilities - SMS notifications and weather forecasts.

All of the mentioned features are handled by our Cloud server implementation, which contains a Cloud API, SQLite database and handles a encrypted communication with our embedded devices. Our IoT device achieved power efficiency via power saving mode, while, reacting to concerns of potential clients, we also added encryption for data transfer between the prototype and the server side.

Chapter 2

Background

In this chapter, we discuss current issues that IoT developers face, such as hardware interoperability, longevity, security, and privacy in a relatively young market of Internet-connected embedded devices. We also cover the topic of Public Cloud services - a technological advancement which enabled the growth of IoT solutions. Other important factors, such as economic benefits and current large-scale IoT projects are also explored.

2.1 Technical developments that allow IoT

Cloud services

The development of Cloud-based services such as AWS [6], which including storage (S3), computation (EC2) and other parts of infrastructure (PAAS services), played a crucial role in IoT expansion.

Before the introduction of IaaS, an Internet-connected device, such as a Nest Thermostat [40] or FitBit smart band [29], would have to store all the collected data locally, process it and have some user interface to show the results to an owner. Such IoT device was not feasible with power limitations, lack of storage and packaging issues. The only option before IaaS would be to rely on private server setup at home, adding additional complexity as well as maintenance issues and costs. All the problems combined meant that IoT existed only as a theoretical concept until 2006.

The landscape of Internet-connected devices development changed with the introduction of the Amazon Web Services in 2006. The launch of EC2 cloud computing [4] and S3 [5] storage services created a low cost, available around the clock online option for data collection and analysis. The always available cloud infrastructure enabled low-power IoT solutions, as the devices now can send the data to the server, without storing anything on site. The collected results then can be processed without any limitations of power resources, and displayed through a web interface or on a dedicated smartphone app, as it is done in many consumer centred IoT applications.

Furthermore, the Cloud providers (AWS[6], Google Cloud[30], Microsoft Azure[50]) now offer PaaS solutions, which give the IoT developers the already set-up cloud infrastructure. The products such as Google Cloud IoT core [33], provide ad-hoc queries via Google BigQuery, machine learning and data visualisations - the features which are at a centre of simple IoT solutions, focusing on data collection and analysis (E.g. sleep tracking, temperature monitoring).

IoT devices

One of the goals we set at the start of our project was to provide an insight into the situations that arise when developing IoT solutions. Since we did not have any experience of IoT prototyping, we had to do background research[51][52] and identify an Internet-connected applications development pattern that we could follow throughout the process.

At the start, a choice had to be made, opting for one of the two prototyping paths:

- Create a prototype based on single-board computer, which would be able to showcase the intended functionality. Due to the lack of focus on power efficiency, the option would not share anything with a mass production setup, except the general idea of implementation and user interaction patterns.
- Focus on building an MVP prototype (Minimum Viable Product), which closely represents a final, mass production device, with similar power, energy consumption and network communication setup.

The strive to gain in-depth knowledge about as many different aspects of IoT industry as possible meant that a decision to work on MVP prototype was made.

Bespoke solution results in limits extendibility - Development of hardware solution tailored to a specific set of tasks limits the choices for further extensibility, for example - Arduino Uno supports most of the peripherals that NodeMCU does, yet it has to forgo a robust WiFi solution. A NodeMCU, on the other hand, would have compatibility issues with Arduino Shields(add-ons explicitly built for Arduino).

Tailored application increases efficiency - A decision to develop a more optimised device provides options to take away unnecessary peripherals, ports, LEDs, chips. Such measures result in a solution that is more compact and power efficient than fast prototyping alternatives implemented with more universal components.

Inspirational IoT projects - at the start we developed the partly functioning IoT device implementation with cues taken from 3 other projects done by hobbyist embedded devices developers:

1. Twitter with Arduino and Ethernet shield [35]- a Twitter reader implementation using wired Internet access. The only take away from this project was an initial hardware packaging idea, along with LCD control. The Twitter functionality was implemented through a no longer available Twitter RSS feed feature.

2. RFID scanner and Arduino for smart door lock [15]- a project which helped to choose a widely used RC522 RFID scanner, supported by most of the low-power devices. Our card reading functionality is based on this implementation as well since the codebase is relatively new and still up to date.
3. Arduino Tweet Reader [41] - a wireless solution for Twitter feed reading. Even though it is able to achieve functionality similar to one of our features, it is limited to reading only the feed of a whole Twitter audience, instead of querying specific account. The project though helped us better understand the security requirements of different Twitter services, as well as security limitations of the low power boards.
4. Arduino Uno with ESP8266 integration [12]- one of many projects, showing an Arduino and ESP8266 working in tandem. We based one of the prototype versions on this example, yet the limited communication between the Arduino and ESP8266 proved that it is not a viable solution.

Our determination to create a power efficient, scalable, secure wireless system though meant that the hardware assembly, along with the whole software codebase (embedded system and the Cloud), had to be done from the ground up. The result thus greatly differs from these example projects in almost every aspect.

Security and Privacy

IoT devices collect sensitive data - Ensuring security and privacy in IoT infrastructure is one of the fundamental challenges of this quickly expanding market. IoT systems are dealing with private data - heart rate, location and sleep tracking (FitBit smart band), Voice detection and communication (Amazon Echo [10] or Google Home [31]), real-time video transmission on smart home security cameras (Canary View camera). And even though they have access to the sensitive information of the customer, they are still expected to be power efficient, compact and responsive.

The low-power devices have secure communication problems - The low-level code does not need an operating system, running instructions on the "bare metal" instead. This relative simplicity of IoT devices ensures that there are fewer opportunities to exploit the software running on a SoC itself. At the same time, the low-power architecture constraints open up security loopholes during the data transfer. To provide privacy and security to the customers, the IoT developers have to tailor fit encryption algorithms that are simple enough not to cause extreme overhead, yet are still robust enough to mitigate security concerns that come with HTTP protocol, BLE connection [47] or RFID transmission [67].

Even some IoT devices currently on sale are not secure - A company developing an IoT solution wants to provide a device with the most functionality, while still using a low-power solution. Sometimes, because an IoT application is meant to be located in a physically inaccessible location, wireless security issues are forgotten, thinking that a device just is not a potential hacker target. At other times, due to the low concern about security from a customer point of view, the developers sometimes choose to trade

a resource-heavy layer of security to an additional hour of battery life, or to another piece of functionality. The result of a negligent view of security and privacy, attacks targeting IoT devices are becoming more prominent, and with the growth of IoT market - more potent and costly.

- Hackable St. Jude Medical Cardiac devices [54]- In 2017, the FDA (Food and Drug Agency, USA) [28], confirmed that implantable cardiac devices have vulnerabilities allowing unwanted access to the device [22]. Once a hacker is connected to a medical device, he/she can adjust the parameters of pacemaker or defibrillator to deplete a battery, introduce incorrect pacing or shocks.

  This IoT device was using a Radio Frequency based data transmitter to provide physicians with device data over Merlin@home product[62]. A link between a low-power device and the central hub, which, just as in other hacks mentioned below, is the reported weak point of a system.

- Insecure hotel key card system [46] - on the same year (2017), the hotel in Austria suffered from a breach of its card system. As a result, guests were unable to enter the rooms, and the staff was unable to reprogram the cards. During the security patching, the loopholes were fixed by replacing compromised systems and isolating service - critical parts of the application from other parts of hotel network.

  The hack of this IoT infrastructure shows the importance of well-secured server - the central hub all connected devices.

- Fitbit security issues [38] - Even a leading producer of wearables (14% wearables market share in 2017 [64]), was unable to provide a robust enough communication between the end node (the Fitbit smart band) and the server (the Fitbit cloud). A research done by Dr Paul Patras (University of Edinburgh) shows that by using a virtualised instance of a real smartphone and reverse-engineering the data transmission, Fitbit traffic can be used to gain complete control over the user data. The data packets sent from an IoT device to the smartphone over BLE (Bluetooth Low Energy) [47] connection get sniffed and altered before reaching the Internet connection layer, thus compromising user privacy, allowing firmware changes or traffic redirection to a different Cloud implementation.

Longevity

IoE devices, such as smart phones, are typically developed with an expected life cycle around the 2-3 year term and the consumer buying it also has similar assumption taken into account. Even if some security issue arises, the software can be easily upgraded over the air both for the newer, and the older models (even if the fix slows a smartphone down, it still keeps it functional and safe).

On the other hand, most of the IoT solutions are expected to have a life cycle between 5-7 years for consumer embedded electronics and 10-20 years for enterprise use [42]. The thinking behind such expectations is - IoT hardware is usually a relatively simple

devices that is good at some specific task (E.g. monitoring temperature with sensors scattered around the city and sending the data to a server), and there is no need for improvements. Also, the potential savings of IoT applications would be greatly diminished by high support cost.

A prolonged lifespan of the device introduces many problems to a whole network connected to IoT:

1. Due to low-level hardware, any update process of the embedded software would either require a physical presence near the device to push a new firmware over the cable connection, or there might be no way to change the software at all since it is installed permanently during the manufacturing process. Even if the updating process is possible, the task of going around every IoT device and manually updating the board will result in high labour and transport costs.
2. The technologies are developing in a fast pace, thus in 10-20 years, the IoT sensors using 2G, 3G, WiFi, Lora [61] or any other current standard of communication, might not even have infrastructure around itself to function properly.
3. If API used by an IoT device undergoes a change, the functionality is likely to be lost. (If the communication protocol and other parts of API implementation were upgraded without backwards compatibility).
4. An IoT device is useful only while the Cloud service is available. If the developer of IoT hardware goes bankrupt or decides to terminate the support due to a significant drop in users, all dependent IoT device become worthless.

2.2 IoT in the University of Edinburgh

Currently, the University of Edinburgh is performing several pilot studies to test potential use cases IoT applications [53]:
1. **Library occupancy monitoring** - With the introduction of OccupEye sensors (monitoring temperature, light level, pressure and movement) which are used to detect if a desk is currently in use, the University aims to make study environment in the Main Library better cater to the needs of students. The project was initiated due to the lack of available study space available for students in the central area. By providing live data indicating the number of vacant desks on one of the library floors, the time spent on the search for free space is reduced, positively impacting the student satisfaction.

The IoT solution is implemented only in one of the floors. Thus, by not providing all of the necessary knowledge (available desks on every level), such change is still not sufficient for daily use.

2. **Meeting rooms occupancy monitoring** - A pilot study took place in a small meeting room located in Argyle House - refurbished University of Edinburgh office. The IoT solution was implemented to provide building managers with information on oc-

cupancy and environmental conditions and trial IoT infrastructure application in workplaces.

3. **Biodiversity sensors in urban green spaces** - The "CitySounds" project is using IoT sensors to capture and analyse the sounds in the Meadows urban greenspace. By gathering sounds from audible to ultrasonic range, the created IoT infrastructure is capable of evaluating the biodiversity and well-being and sharing the results with the public.

IoT Research and Innovation Service
Apart from testing the possible applications on current infrastructure, University of Edinburgh is also aiming to provide a suitable network for a broader range of connected applications. The IoT Research and Innovation Service are already assembling a Low Power Wide Area Network based on LoRaWAN [61], a secure long range and low power radio communication technology, with a current coverage shown in the Figure below.

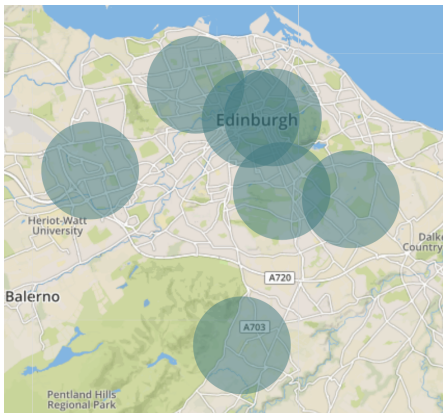


Figure 2.1: LoRaWAN gateways with 2km connection range

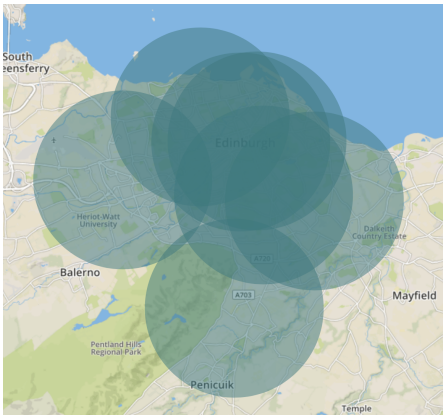


Figure 2.2: LoRaWAN gateways with 5km connection range

2.3 Economical considerations

1. **Data** - The continuous data collection and monitoring enables companies to process collected information and introduce added benefit to the consumer.

For example, the statistics gathered from UPS vehicles Internet-connected sensors [49] allow to measure the wear of components and perform preventative maintenance, reducing the downtime of an asset.

2. **Automation** - IoT solutions also enable people to automate processes. For example, the smart thermostats[40] can reduce heating and electricity bills in the office or at home. By combining the sensors temperature data with movement sensors, a smart thermostat could automatically adjust the heating to better suit people while they are present, and reduce it when no one is around.

Chapter 3

Hardware Research

The crucial part of our project was to identify the hardware components that adhered to our power consumption and performance requirements. We started our search for the suitable options thinking that since IoT devices are becoming more and more popular, we can as well expect the market to be sufficiently mature with plug-and-play modules available for newcomers. In reality, though, competition between different standards and technologies, in fields such as Internet connection (ZigBee [2] vs WiFi vs LoRa), serial communication protocols (I2C vs SPI) or supported programming languages (C vs Lua [60]) results in a market suffering from fragmentation. Manufacturers are not trying to limit this problem either, adding their modifications to the setup of a SoC, sensor, LCD display or other prototyping parts sold to the public.

Taking the compatibility problem into account, we readjusted our priorities. We decided to, first of all, find a main IoT prototype board that has a proven record of stable WiFi connectivity, a well balanced low power SoC with a good performance and support for most of the peripherals in the market. Even then, there were many problems with compatibility of peripherals, let it be a hardware related issue or a problem with lack of code libraries that support the interaction between two parts. With extensive testing and development, we had to go through four different versions of our hardware implementation to come up with well functioning and efficient final design.

3.1 Final hardware design

For the last iteration of our prototype, we tried to reduce the number of parts to the bare minimum, mainly because using fewer parts results in lower power draw, reduction in cost and smaller overall packaging. After in-depth comparison between 3 options for our final prototype iteration (Wemos , Huzzah, NodeMCU)3.7, NodeMCU fit our power, efficiency and compatibility criteria the best, becoming the heart of our setup. NodeMCU is a small main board with ESP-12 SoC 3.6 (System on a chip) that has embedded WiFi module for Internet connection. All other parts of the system are I/O devices. For Input, we are using RFID sensor 3.13 (To scan Student cards) and two buttons (To trigger preset action, such as sending a request to the Cloud or wake the

device up from Sleep mode). Our output device is an LCD screen, able to show four horizontal lines of 20 characters. The problem though was that some of our parts were running on of 3.3V while others required 5V. The solution for this issue was to use a Bi-Directional Logic Level Converter that shifts the data between the different voltage channels. Lastly, a 3350 MAh power bank with added wireless charging functionality is used to power the setup.

Figure 3.1: Final Hardware

3.2 Main IoT device board

Choosing the board for IoT project is one of the more difficult parts. IoT software greatly depends on the limits enforced by choice of main hardware.

For example, our NodeMCU board can only run either C++ based Arduino IDE code, or code written in the more abstract Lua programming language. Arduino Uno 3.2 and other of Arduino systems are even more constrained, dropping Lua support.

On the other hand, if a choice is made to develop on more powerful systems, like Raspberry Pi, you get a full-fledged OS running on a device, with support for most of the popular languages such as Java, Python, C++ or JavaScript. This choice is not without compromises though - more than ten times higher power consumption, as well as lack of software support for peripherals, are main issues restricting the application of OS-running systems from the IoT applications.

Since the hardware choice enforces a software environment, we had to do extensive research, making sure that a most optimal micro-controller and board package have been selected at the very start of the development process. Due to the versatile nature of our project, the main requirements were:

1. Power efficiency - The device is meant to be portable and function wirelessly. It has to last at least a day on its own power for event participation monitoring tasks. In smart office scenarios the requirements are even higher, with expected battery life of around one week.
2. Easy recovery from power loss and ease of use - the device boots up automatically when the power is restored, continuing to function without any additional setup.
3. Main board compatibility with multiple additional sensors - RFID scanning for UoE student cards, LCD display 3.19 for Tweets.
4. Reliable wireless connectivity - Used for transmitting data back and forth between IoT device and a Cloud Server, a vital part of the device to be functional.

In addition, the popularity and price of a product were important as well. The project was our first step into IoT solutions, with a helpful community, IoT projects, and guides available online being of utmost importance.

We set Arduino Uno R2 as the starting point, due to it being a well documented and widely popular option for IoT solutions. After facing numerous Arduino issues with Wireless connectivity, we transitioned to a board explicitly tailored for IoT applications - NodeMCU development board with ESP8266 SoC (System on a Chip). This change of setup, highly recommended by the IoT hobbyist community[43], allowed us to solve Internet access issues and, after flashing the board with firmware for Arduino code, focus on developing the embedded and server-side software. Furthermore, the NodeMCU offered more processing power and memory, while still managing to provide a great power efficiency, 75% lower price and 60% smaller footprint.

Apart from Arduino UNO and ESP8266 3.3, did search for other alternatives in the market. Boards such as Raspberry Pi, Intel Edison and various other options were deemed either lacking a sizable developers community, targeting a more professional developer audience, not having sufficient compatibility with other I/O devices, or just too expensive.

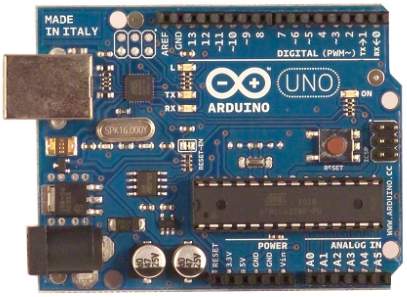


Figure 3.2: Arduino UNO R2 development board

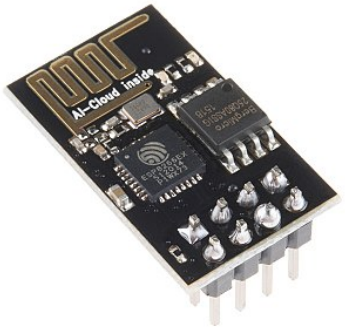


Figure 3.3: ESP8266 System on a Chip made specifically for IoT solutions

Arduino

The Arduino Uno R2 3.2, together with Ethernet Shield 3.12, was the first version of our prototype development. As Arduino is the most popular hardware prototyping device, a variety of examples for Twitter integration, RFID sensors and other projects exist online, along with setup guides for wired and wireless internet connections or other hardware peripherals. Besides, Arduino boards allow an in-depth control of energy consumption, by using multiple levels of device sleep states, enabling the board to function from smartphone size battery (2500mAh) for more than a month.

Arduino, similarly to other embedded devices, is low maintenance and not sensitive to power loss, as right after the power is restored, it simply boots up and starts looping the instructions. For comparison, single board computers can suffer from file corruption and require user help to go back up online after the power outage. Arduino has well-implemented error recovery as well - if board runs out of memory, it restarts instead of becoming completely stuck.

It does not come without flaws, with the key one being a lack of at least moderately up-to-date wireless connection set up. With Arduino-made WiFi Shield discontinued 3.10, as well as Arduino Yun, the WiFi enabled Arduino board version, losing support, the only reasonable option left is an integration of ESP8266, a system capable of running Arduino code on its own, as a WiFi card for Arduino Uno. As this integration is not off-shelf, but community driven, it lacks proper documentation and guides, as well as adding limitations due to the usage of Serial Bus to communicate between two systems. Last but not least, Arduino Uno runs on 5V, unlike the ESP8266-suitable 3.3V. Therefore a logic level shifter has to be used, in addition to 5V to 3.3V Step-Down regulator for supplying power to ESP8266 from the same 5V power supply.

Arduino Uno R2	
Pros	**Cons**
Power consumption	Internet connectivity issues
Recovery from power loss (Power loss does not corrupt data)	Discontinued peripherals (E.g.WiFi Shield)
Compatibility with peripherals (RFID card reader, LCD screen, etc.)	Lack of up to date guides for some modules (Ethernet Shield, WiFi Shield)
Error recovery - If device runs out of memory or encounters other type of error, instead of getting stuck, it reboots automatically	Limited processing power (16 mHz on Arduino UNO in comparison to 80 mHz CPU speed on ESP8266)
Low cost of unnamed versions made according to freely distributed Arduino board design (£6.99)	Steep price of Arduino-made boards (£22.20)
Support for Analog and Digital sensors.	

Raspberry PI

Several versions of Raspberry Pi were considered for the project, mainly due to the ease of programming for such device. All of Raspberry boards, from smallest and least powerful Raspberry Pi Zero to Raspberry Pi 3B+, are powerful enough to run stripped down Debian Linux distribution. As a result, development on Raspberry Pi does not depend that much on the choice of programming language. This family of SoCs is also fully capable of supporting additional digital sensors. All of Raspberry Pi form factors have easy integration of wireless Internet connection (WiFi is available either with an inbuilt module or via USB WiFi dongle explicitly made for Raspberry boards), and Bluetooth connectivity also comes embedded or can be added with an additional dongle.

Even though Raspberry Pi has a simple WiFi connection set up and supports a variety of peripherals, several design choices of this board make it unsuitable for IoT applications. First of all, IoT devices are required to be low maintenance, and since Raspberry Pi is running a full-fledged Linux distribution, in case of a program error or a power loss, it would require a user to start up the device and programs once again. Sometimes unexpected power loss can lead to corrupted memory blocks as well. Apart from the power loss issues mitigation, another feature required for low maintenance devices is power efficiency. Raspberry Pi being a mini-computer, some amount of resources are always necessary to keep the OS running; thus it cannot compete with an OS-less microcontroller system. As measured by Raspberry PI forum users 3.4, the board draws between 100mA and 350mA on idle, and even if the single board PC is entirely powered off, it would still use 20-30 mA if a physical wire to the power source remains connected. For contrast, while idling Arduino Uno R2 and NodeMCU boards draw 53mA and 70mA respectively and both can lower power consumption state when in light/deep sleep mode.

Raspberry Pi boards (Zero - Pi3B+)	
Pros	**Cons**
About 10 times more powerful than Arduino IDE running boards. Raspberry Pi boards run on CPU clockspeed between 700MHz and 1400MHZ.	User dependent error recovery (If running process gets stuck, a program will not restart on its own, requiring user to manually reset a device).
Full Debian based OS enabling multiple programming languages support.	Risk of memory corruption in case of power loss. (Running out of battery charge / disconnecting from power socket without shutting down OS first)
Inbuilt Bluetooth, LAN and/or WiFi connectivity.	High power draw during an idle state.
Large IoT hobbyists community	No on-board support for analog I/O.
	Even the cheapest option is equal to the price of three NodeMCUs. (Pi Zero - £9.2, NodeMCU - £2.92).

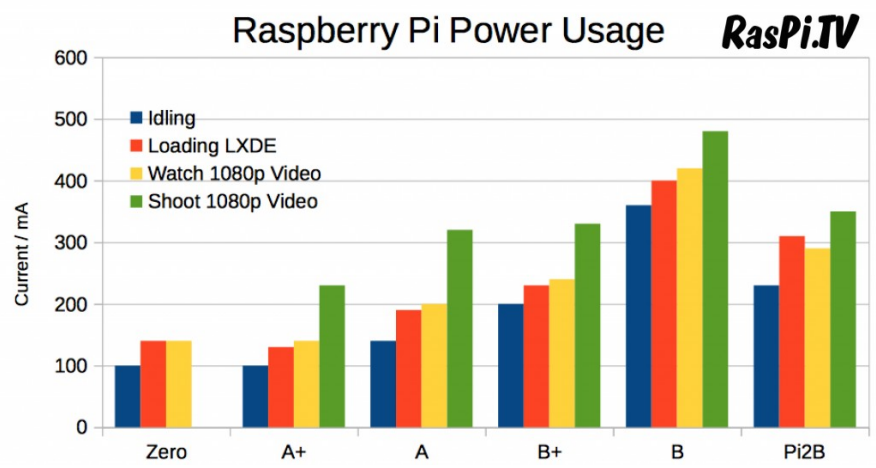


Figure 3.4: Power consumption of Raspberry PI models[58]

ESP8266 SoC family

ESP8266 is a backbone of most popular IoT development boards with embedded Wifi module, all using the base ESP8266 or improved versions of it. With it originating from China, no English documentation was available until it was introduced to the western market in 2014. Due to inbuilt WiFi module though, it quickly became a preferred over Arduino Uno systems for IoT applications. Since the ESP8266 SoC design is free to use available, a variety of manufacturers are making their versions of these chips, without adhering to any specific standard, changing the amount of available flash memory (512KB to 4MB) or the number of available pins (8 to 20).

Even though Arduino Uno and a Raspberry Pi are described as the main options for IoT applications in most of the online publications, increasing ESP8266 user base pushed a shift towards this new device. ESP8266 is capable of running Arduino IDE code, which enables the use of low-level, more optimised code. Some models (NodeMCU, Huzzah, Wemos D1 Mini) have direct micro-USB connection with 5V to 3.3V shifter, and other, smaller versions (ESP-01, ESP-12) require an FTDI (semiconductor device company, Future Technology Devices International) breakout board to establish a wired connection with the computer.

In comparison with the Arduino Uno board, ESP8266 is capable of establishing a WiFi connection on its own, using an inbuilt transceiver. As a result of optimised and compact chip design, ESP8266 requires less power in applications where internet connectivity is necessary. It also allows processor sleep state manipulation in the same way as Arduino Uno, since the code-base of these two boards can be used interchangeably. ESP8266 is a smaller device than Arduino Uno, with more versions of it available, allowing to prototype using a board with more ports, such as NodeMCU, and then choosing a smaller, more single-purpose MCU like ESP-01 3.5 for the final iteration.

The problematic part of ESP8266 though is the variety of board setups, which meant that most of the versions available in EU had to be researched in depth, to choose the most reliable and well-supported version. Even then, general development queries are handled by other hobbyists on the ESP8266 forum [26], and initial setup is still not very straightforward, due to modest manufacturer support.

ESP-01

Raw module Designed to function either as a standalone device, or a WiFi module for Arduino microcontroller. It has limited applicability as a standalone IoT device due to having only 2 GPIO (General - purpose input/output) pins that are used for connecting additional sensors. This board is especially sensitive to varying voltage and requires 3.3V to be used for powering it and for passing the signals on any of the ports.

ESP-12

Raw module

A SoC, with 11 GPIO pins available, as well as 1 ADC pin, allowing to have an interface for digital communication with peripherals such as a display, as well as an analogue connection for simple sensors such as an accelerometer. Just as ESP-01, it requires 3.3V power.

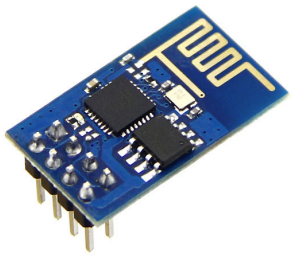


Figure 3.5: ESP-01

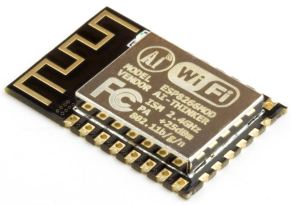


Figure 3.6: ESP-12

Adafruit HUZZAH ESP8266 Breakout
Development module
A standalone board, comprising all features of ESP-12, just with additional electronics soldered to the board for more I/O options. The main advantages are breadboard - friendly design, allowing for easier prototyping, as well as power regulator, allowing the ESP8266 to tolerate 5V power sources as well. As most of the ESP8266 Development Modules, it comes with FTDI pinout for software uploading onto the chip, which still requires FTDI cable to allow uploading code from a computer through USB.

NodeMCU
Development module
NodeMCU is an improved version of Huzzah ESP8266 Breakout. The main difference is that FTDI pin-out is replaced with a micro USB port. The change allows direct connection to a computer by using micro-USB to USB cable type. A micro-USB connection enables a direct upload of Arduino code to the board, without pressing reset buttons, which would be required in the previous board design.

Wemos D1 Mini
Development module
A cheaper version of ESP8266 breakout, the main alternative to NodeMCU development module. It shares most of the parameters with the NodeMCU design but offers a smaller form factor. On the other hand, it is less popular, and as a result support for it is not as easily accessible.

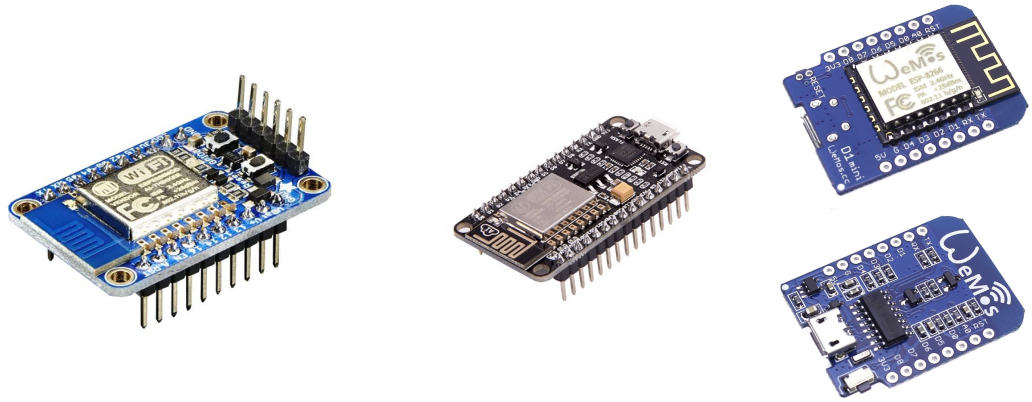


Figure 3.7: Development boards:
HUZZAH ESP8266 Breakout, NodeMCU and Wemos D1 Mini

Main board choice - NodeMCU
The project hardware design went through several iterations, and for the final version, the ESP8266 breakout board was selected as the most suitable option. Specifically, it is a NodeMCU standalone development module as we require a higher amount of pins for screen connection, RFID scanner and buttons, thus smaller board options were not sufficient. Other iterations were based on Arduino Uno combination with an additional board for Internet access, which is a topic covered in "Internet Connection" section.

3.3 Internet connectivity

The development of our IoT device started out by using an Arduino UNO board, which did not have WiFi and Ethernet connections. Since prototyping started from a completely blank page, we at first were trying to use different versions of shields to create Internet access (Shield - a board that stacks on top of Arduino, adding some additional functionality). The first shield was for Ethernet connection, and later, after a decision was made to make a portable solution, Wifi Shields. Though, due to lack of up to date libraries and dramatically diminished community support, none of these Shields was functioning as desired for our application. As an alternative, ESP8266 SOC, which is the basis for our final NodeMCU, was tested in a role of WiFi Module, but the communication on a serial bus between Arduino and ESP8266 was unstable as well. For our final version, we completely rethought our approach from the ground up and researched the alternatives to Arduino in more depth. The findings of our research resulted in a transition to NodeMCU - a much smaller device which has embedded WiFi module, solving our problems with software compatibility, power consumption and resulting in a compact package.

NodeMCU embedded WiFi module 3.7
The NodeMCU itself is an ESP8266 SoC with WiFi functionality, running an Arduino code. As a result, we do not have any problems associated with communication between the main board and a separate module/device over the Serial interface. The WiFi functionality is set up via the ESP8266WiFi and ESP8266HTTPClient libraries**??**, removing the necessity of writing AT commands, improving the robustness of the communication.

Furthermore, as both Arduino code and WiFi communication is handled by the same SoC, the power efficiency increases and a whole system footprint gets smaller (NodeMCU is 1/3 of Arduino Uno size)

ESP8266 as a WiFi Module 3.8
An ESP8266 functioning as a WiFi module is a cheap, yet complicated way of adding a WiFi connection to the Arduino UNO. The ESP8266 has an integrated TCP/IP protocol stack and is capable to either function as a WiFi shield, or run Arduino IDE code by itself (The latter option isn't suitable for us. The Low amount of pins means that we would not be able to connect all of our peripherals).

Using ESP8266 as a WiFi module requires additional components to provide communication and shared power supply between two devices. ESP8266 runs on 3.3V whereas Arduino needs 5V at supply pin. As a result, we add a logic-level switcher (which functions as a gateway for communication between devices using different voltage) and voltage step-down chip (creates 3.3V power access reducing the incoming 5V voltage from the power supply).

The pairing of ESP8266 and Arduino Uno has one limiting factor though - the pair can work together only by using AT (ATtention) commands over the Serial Bus. AT commands are not as reliable as solutions with embedded WiFi connectivity, thus even after lengthy testing of ESP8266 as WiFi shield, the setup was replaced by NodeMCU board with integrated ESP12 WiFi chip.

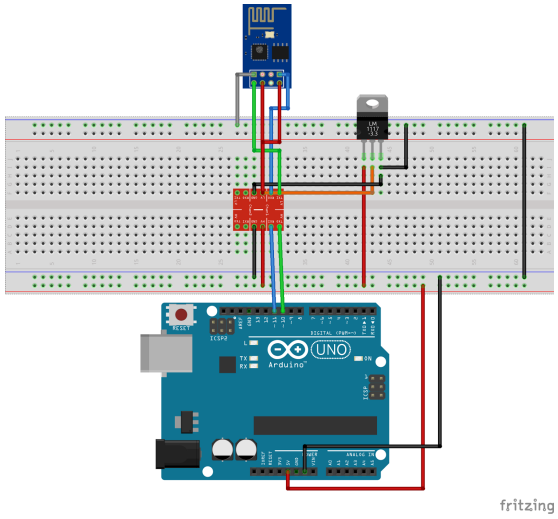


Figure 3.8: Arduino UNO with ESP8266 as a WiFi shield

ESP-12E UART WiFi Shield 3.9

The ESP-12E UART shield is meant to function as a plug and play solution enabling WiFi connection to Arduino, while still keeping all the ports free. This ease of setup was the main reason why we bought and tried it. In reality, though, this module does not communicate with an Arduino board through the mount pins. UART module requires additional wiring to be added between the pins located in the middle of a shield (where ESP-12E WiFi SoC is located) and the Arduino pins extensions on the sides of a shield. This results in a same problematic communication just like with the Arduino and ESP8266 3.8.

Arduino WiFi Shield 3.10

Arduino WiFi Shield is also not supported by the manufacturer anymore, and as a result, libraries are out of date. In addition to this, it does not have a low power consumption mode (Deep Sleep) like an alternative ESP8266 option, it is a more expensive than ESP8266 and it also does not support TLS/SSL required for most API's.

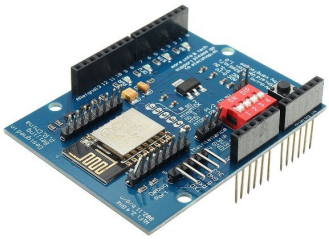


Figure 3.9: ESP-12E UART WIFI Shield

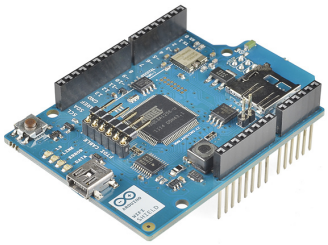


Figure 3.10: Arduino WiFi shield

Arduino Ethernet Shield with Power over Ethernet 3.11
Another inherent flaw of Ethernet connection is lack of mobility since it is still a cable-dependent connection. This could be turned to a positive aspect, by using PoE (Power over Ethernet), to power device with the same cable which is used for data transfer. It would still mean that a device always has to be attached to a wire, and such solution is a too big drawback for a portable application part of our project - the event participation monitor. Even for a smart office device application, a wire would have to be brought right through the front wall or somewhere near the door of an office room, making installation process overly complicated and expensive.

Arduino Ethernet Shield 3.12
Internet connectivity over Ethernet Shield has several limitations, with the main one shared between most of Arduino peripherals - discontinued manufacturing and support. Consequently, libraries are not being kept up to date with newest standards and community support is diminishing. Just as the other Internet connectivity solutions, Ethernet Shield is unable to work with TLS/SSL authentication required by most popular API's available online. As a result, data from OpenWeatherMap API [55], Twitter API [69] and other popular services is not available.

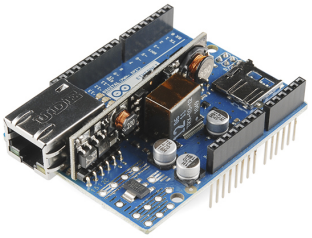


Figure 3.11: Arduino Ethernet shield with Power over Ethernet(PoE)

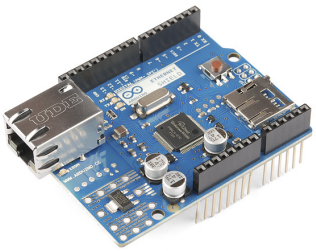


Figure 3.12: Arduino Ethernet shield

3.4 RFID cards and tags scanning system

RFID
RFID (radio frequency identification) - wireless communication incorporating electromagnetic spectrum radio frequency to uniquely identify an object. An RFID system combines three components: scanning antenna and transceiver (a reader), as well as RFID tag, acting as a transponder. The whole system works in a pattern of transceiver transmitting a signal over radio frequency waves, activating tag within a range. Activated RFID tag then responds by sending back a wave containing data, which is translated by the antenna of RFID reader.

For the second IoT application (event participation monitoring), an RFID scanner was required, as by using embedded NFC scanner on a smartphone we found that UoE Student ID card uses RFID system for authentication. To do further research, as well as an implementation of IoT system itself, we decided to work with an RC522 Sensor 3.13, along with RFID enabled tags and cards.

Common applications:

- Access Control
- Item Tracking
- Inventory Management

NFC
NFC (Near Field Communication) is a subset of RFID technology. It operates at the 13.56 MHz frequency, just like High-Frequency version of RFID (Radio Frequency Identification). NFC differentiates[67] itself from other RFID applications by providing a secure data transfer (High-Frequency 13.56 MHz NFC signal functions only to around 3 cm distance, adding a physical distance barrier as one of the security measures). In addition to this, NFC enabled device can function as a reader and a tag, allowing a peer to peer communication.

In our IoT solution, the use of NFC was not researched in depth, as the application relies on RFID communication with University of Edinburgh staff and student cards. After tests with RFID reader were done, we were sure that student cards are only passive RFID enabled, without NFC chips in them.

Common applications:

- Information sharing between two NFC enabled devices (Pairing devices, sending contact details)
- Contactless payment services (Apple Pay [11], Google Pay[32], Contactless bank cards)
- Smart Posters (Posters with NFC tags, enabling embedded information to be shared with other devices)

RFID Reader

RFID-RC522 (MFRC522)
RC522 tags reader is using a 13.56 MHz electromagnetic field. This spectrum allows it to interact with RFID tags in a measured distance up to around 5-20 mm from the centre of a sensor. It is supported both by Arduino Uno and ESP8266, with community-driven codebase support available on GitHub repository.

We chose to use this version of RFID reader because of its widespread popularity, a high count of available example projects online and incredibly low price of around £1.5.

Figure 3.13: RC522

Adafruit PN532 NFC/RFID breakout board
This version of a PN532 sensor is primarily focused on NFC applications. It is able to read/write the NFC tags or function like an NFC card itself. Furthermore, Adafruit board is highly versatile, supporting RFID tags as well. On the other hand, there are two main issues with this peripheral - the price and the size. It costs £40 - 25 times more than RC522, and the dimensions of this device are 54mm by 120mm, in comparison to 39mm by 59.5mm of RC522. As we did not need the extra functionality that PN532 offers, this sensor did not fit our criteria of a compact and cost-effective device.

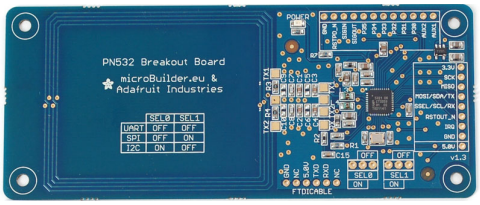


Figure 3.14: RC522

RFID Tags

RFID Card Tag Types [1]	
Type	**Frequency**
LF (Low-Frequency)	125 kHz / 134 kHz
HF (High-Frequency)	13.56 MHz
UHF (Ultra-High-Frequency)	865-868 MHz / 902-928 MHz
Active (Battery-Powered)	2.45 GHz / 433 MHz

Figure 3.15: RFID tag types

Active RFID tag
A tag with its own power source for a longer read range and larger memory storage. Active RFID tag continuously broadcasts its signal to achieve the capability of being detected and read from a further distance. On average, active version of RFID tag lasts 3 to 5 years, and most commonly the battery is not replaceable; thus the whole device has to be swapped.

Active RFID tag can be used as:
1. Beacon - tag does not wait for readers signal. It sends out information every 3-5 seconds instead.
2. Transponder - Functions quite similarly to passive systems. Active Tag waits for a signal from the reader, and then sends relevant information back to the reader. Because a system functions only when it is in range of a reader, battery life is better than in beacon application.

Active tags are mostly used in applications that require ruggedness and resistance to varying temperatures and moisture. As a result, they are bulky, heavy and expensive (prices range from £20 to £100). None of the Active RFID features is adding benefit to our solution, and University has already implemented passive RFID cards system that is fully compatible with our event participation monitoring. Thus we quickly dismissed Active RFID tags as an option.

Use Cases:
Active Tags, in general, are tailored for environments requiring higher tag reading range (up to 100m). The transponder is used for secure access control, toll booth payment systems. The beacon is for accurate real-time tracking of high-value assets.

Figure 3.16: Active RFID tag

Passive RFID tag
A tag receives power from the antenna of RFID reader using electromagnetic wave induction to provide current to RFID tag. The passive RFID tags last more than twenty years and cost less than active RFID solutions while keeping a smaller footprint as well.

The passive cards function in a high-frequency range (13.45 MHz) and include fixed or rewritable (programmable) memory for processing the communication data 3.15.

Use Cases:
Access control, file tracking, smart labels.

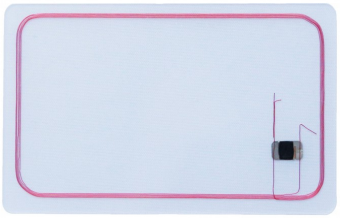


Figure 3.17: Passive RFID tag

University of Edinburgh Student RFID card
Every University of Edinburgh Student card contains a MIFARE 4K RFID chip with an antenna. The tag has 4KB of EEPROM memory, with 40 sectors where encoded information is held. The only part that RC522 can read without information supplied by the university is the unique ID of the card, as the data sectors are encrypted with specific keys.

For our solution, knowledge of ID itself is enough, if we can assume that in a release version of a system, a connection with main UoE database and the access to encrypted card data would exist. In this case, access to the UoE database would allow identifying the students without relying on the RFID ID which can be copied into another card.

Use Cases:
Access control, Personal Identification, Payment for services within University.

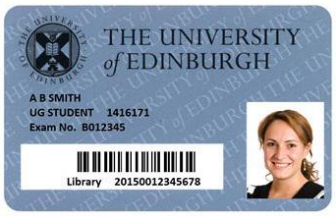


Figure 3.18: Student RFID card

3.5 Other peripherals

QC2004A Display with LCD2004 Chip

As both Arduino Uno and NodeMCU have a limited amount of pins, the QC2004A LCD display using LCD2004 Chip was chosen. The screen features four rows with 20 characters on every line, allowing for sufficient amount of information from Twitter or general use case, to be displayed. The main benefit of using LCD2004 is the reduced amount of connections required to power and interact with a display as this serial controller enables an I2C pins connection.

We researched other display options, but E-Ink (Electronic Ink) and Matrix displays are either too small[8], or cost three times more[9] than QC2004A[7].

Figure 3.19: QC2004A Display, Top - LCD2004 chip, Bottom - QC2004A display.

Buttons
The buttons are a quite straightforward hardware piece of our project. They are used to launch actions such as a refresh of Twitter note or trigger an SMS sending action. One additional part of the functionality that a button attached to reset pin on the board could do though is the on-click interrupt of a deep-sleep power saving state. The click of a button wakes the NodeMCU up, does the programmed task and goes back to sleep until a new reset event is triggered.

Enclosure

The main aim of our project was to build a fully functioning proof-of-concept, going through all of the prototyping steps that any company developing IoT solution has to overcome as well. This goal meant that an enclosure for the device had to be assembled. We looked into several different options, starting with just modifications of universal plastic boxes available for sale on hardware prototyping shops. The available models were either too bulky or too small and without required fixation points for our hardware 3.20.

A decision was made to design a 3D model of enclosure that would fit our requirements perfectly, while also giving us a chance to try out 3D printing, which is commonly used

for proof-of-concept designs. We, first of all, drew the preliminary design on paper and took dimensions measurements of all the components of our prototype. As the next step of a process, Autodesk Inventor, a 3D CAD design software, was used to create a case with holding pins and slots for every part.

Our use of 3D modelling and printing allowed to have a lighter, more compact and solid structure. Due to a choice of using a custom structure for the box walls, the RFID scanner was still able to scan the cards through the plastic casing (the insides of the walls were printed in the porous honeycomb pattern instead of solid plastic, to maximise scanner reading distance).

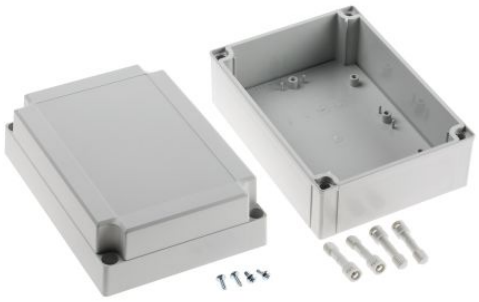


Figure 3.20: Universal prototyping box

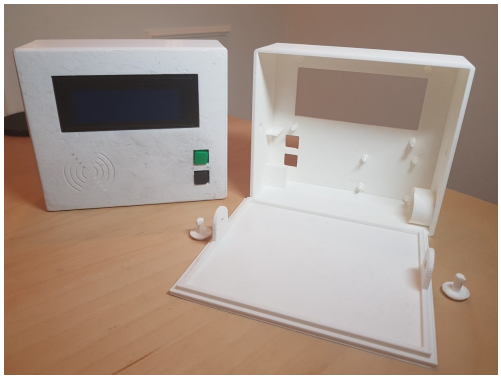


Figure 3.21: 3D modelled enclosure

Power supply

For the device to be portable, it has to have a battery, and we decided to use a widely available 3350 mAh Anker Powerbank 3.22[3]. The main reason for choosing it was safety - these portable battery chargers come with anti-surge protection, as well as short circuit protection. Having such security measures in place is especially important in prototypes, where a loosened wire would otherwise damage a board or some of the peripherals.

We have tested these safety systems, comparing the behaviour of a shortened device when powered from a powerbank, and from an unprotected power supply. While our portable battery pack cut off a power supply and restored it only when short-circuit was removed, the unprotected supply did not react to the situation, and a board was damaged beyond repair.

Figure 3.22: Anker Powerbank

Chapter 4

Software Implementation

Our final codebase consists of two main pieces - software embedded on an IoT device and a system running on a back-end Server, similar to a majority of IoT projects. IoT device is running on a C++ based code, while our server side is a Cloud system set up on an Amazon Web Services EC2 (Elastic Compute Generation 2) virtual machine, running Ubuntu 16.04 operating system.

The brain of IoT device is a NodeMCU Wifi 3.7 development board. It handles communications with embedded WiFi module, RFID scanner, LCD screen and awaits for clicks on either of the two buttons to initiate associated actions. The main board reads, filters and encrypts the data gathered from RFID 3.13 scanner. NodeMCU then sends encoded information to the server through a WiFi connection. When a response is received, our board decrypts received information, processes it, and displays received a message on the screen. By design, our prototype does not do any heavier computation. It is vital to ensure the best power efficiency of this battery-powered device without sacrifice of fast response. With the aim of achieving these targets, the IoT device only communicates with the PHP script on an AWS server. All of the complex tasks are done by a virtual machine on the server-side.

Our Cloud server is a virtual machine running on the Amazon Elastic Compute Cloud, otherwise known as EC2 [4]. It is the brain of a whole IoT system we have built. As mentioned before, most of the processing and functionality is on the cloud side, as unlike the IoT device that we created, the server does not rely on battery power and runs a full-fledged operating system. Even then, our tasks are not requiring an extraordinary amount of processing power and a single core, 1 GB ram setup is just right (or even too powerful) for the application. The service is handling API requests coming from the IoT device, talking with Twitter, Twilio and OpenWeather API's. It also interacts with SQLite database via ArrestDB - RESTful PHP API. The whole system is very cost effective, taking less than £6 per month to run 24/7.

4.1 Programming the Embedded System

Choice of programming language

The implementation of our prototype required to be portable, energy efficient, easy to use and support peripherals with a variety of different connections (E.g. I2C, SPI). As a result, the final software setup was created for a device best fit for a purpose - a NodeMCU, low-powered SoC, using C++ based Arduino code. Since the project combines both hardware and software, we could not avoid the trade-offs that come with a choice of boards using low clock-speed processors and peripherals having limited manufacturer support. With these constraints in mind, we chose C++ as our language of choice. The decision was based on the fact that C++ IoT prototyping community is the most active one. Thus there is a sizable amount of example projects, guides, custom libraries and helpful forum threads online.

C++
Over the time we worked on implementation, we were always thinking of ways to extend device functionality and make it more efficient. The potential changes in hardware peripherals meant that the codebase, just like our hardware, had to be compatible with a high amount of potentially useful sensors and peripherals available. Most of the manufacturers of these sensors, supply libraries for C++ solutions, due to the high popularity of Arduino Uno and similar prototyping boards. If the manufacturer support is lacking, or a library is not functioning, a strong IoT Arduino community is quick to come up with a working solution. Just as mentioned in Power Efficiency section, there is more freedom to optimise device to great detail, as low-level language has better access to some specific controls such as Analog to Digital Signal Conversion.

Lua
NodeMCU is only sold with firmware supporting Lua programming language. Nonetheless, there is a flasher supplied by the NodeMCU creators, allowing it to be used with Arduino IDE environment. Since we still did not have big codebase at the point of NodeMCU arriving at our door, we still investigated the possibility of using Lua as well. After a brief search online, it was clear that it will not be our language of choice. As Lua is a higher-level, more abstract language, a code performance suffers from a drop, the important access to hardware level for energy efficiency, is lost. A chain reaction of problems with this language follows from that step. Due to lower performance and higher power consumption, fewer people are joining Lua community and code is not being as proactively developed. Lua also suffers from random freezes and crashed often enough, that it would be unstable to run for longer than 3 hours. In the end, we think it is quite clear why Lua was not our choice.

Single board PC alternative
While our prototype is using Arduino code running on a low-power NodeMCU board, there was alternative option to achieve similar functionality - a single board PC implementation. An example of such device is a Raspberry PI board. It is powerful enough to run a stripped down version of Linux from SD card. An operating system allows single board PC to run programs written with Java, Python, PHP or any other high-

level language, and use a wider selection of encryption libraries. As a result, it would be able to handle simple IoT device tasks (reading sensors data, communicating with Cloud API) and have the ability to do most of the Cloud tasks as well (sending requests to Twitter[69] / Twilio[68] / OpenWeather[55] APIs, communicating with a database via ArrestDB API).

As it was mentioned in section 3.2 though, using single-board PC has implications on power consumption and device ability to recover from power loss. The power issues, combined with lack of libraries support for many peripherals, restricted us from using a more complex software on the IoT side.

Integration of the peripherals

Handling communication with peripherals is another task where our choice of hardware and software base made a lot of sense. As the great majority of IoT projects use either Arduino boards or ESP8266 running Arduino code, it was quite easy to set up a connection with our peripherals.

Displaying text on a screen
The IoT prototype has an LCD screen, used to show tweets, SMS notification status when in use or weather information when other functionality is not required. Our screen is a character display, that is, it can show 20 characters per line, with four lines in total. Sometimes text from Cloud responses might not fit into the 80-character limit. One of our functions takes the text length into account, and if the text is longer, after a pause of 10 seconds, the first block of printed out text is wiped, and a second block is then shown.

The LCD is using the I2C connection via an attached interface (backpack), to reduce the number of wires required for communication with main board from 15 to only 4. The I2C connection requires specific library provided by a maker of I2C interface Sainsmart, and though this vendor supplies a faulty C++ library, online we found an improved library, "LiquidCrystal_I2C", made by a member of IoT prototyping community.

RFID communication

RC522 RFID reader 3.13 is used for event participation monitoring. For our proof of concept, we are using it to mark students' attendance in tutorials. Connecting an RC522 RFID reader software-wise was quite straightforward, as a library worked immediately and with supplied example projects. The supplied libraries are lacking I2C protocol support, forcing to use SPI protocol and restricting us from sharing the same pins for LCD screen and RC522 module at once.

For the participation monitoring, the scanner is set to be always enabled, reading the data, specifically the ID of a card when it touches the sensor. The initial format of data is saved as a 4-byte array and matched with the last read UID. If they match, the whole

iteration of a loop gets terminated, and a new iteration starts. This action is done to prevent spamming of our Cloud API with the same readings that a sensor would get if a person holds the card for more than a second. If a new 4-byte array is different from previous one, it then gets converted to hexadecimal representation and saved as a string.

On the next step of a process, for secure data transfer, hexadecimal UID is with Base64, encrypted with AES128. It is then sent via POST request to the Cloud API. Server-side processes the data sends a response payload, which is then decrypted and displayed on a screen.

Scanned identification data

Why is UID of a card used instead of a student number?

We were curious to find out how well-secured data on the University cards is. To research this, we used one of supplied RC522 functions, dump_info, which, just as the name implies, tries to access all the data inside a card. When tested with University student card, it showed the RFID ID of a card (UID), yet all of 40 memory sectors were encoded and required University supplied access codes to be read. These access codes are inbuilt in every university card reader, and same ones are used to decoding the card information of every user. Naturally, we could not get a piece of software with such high-security risk. As a result, instead of accessing specific memory sections of a card, we are using the RFID UID, which is an id of a card itself. In the proof of concept implementation, UID is the only piece of data required to be on an RFID card, as it is associated with student ID on the Cloud server. All student information is available for access only on a database.

Security

From a first look, our implementation might seem not that intrusive - the only data read from a user is a card ID, allowing to identify a person. On, the other hand, when this piece of data is associated with other parameters, it allows identifying where the specific user was at the point of card scan. It is done by:

- Name and Surname - read ID is matched with list of names and surnames data available on database.
- Location - when ID is read, it is sent to the Cloud together with an unique ID of IoT device (location of IoT device is known from event location specified on database.
- Time - on information arrival to the Cloud, a timestamp is taken, allowing to associate it with student and location.

As our device is meant for enterprise use, processing sensitive data, we had to ensure secure communication between IoT devices and a Cloud server. Since IoT market is still emerging and growing rapidly, there are no regulations enforced on the manufacturer that we could follow during the development. In Europe, for example, there is no

basic level defined for the security and privacy of connected and smart devices (ENISA - European Union Agency for Network and Information Security). In addition, no legal guidelines or precautionary requirements for security of IoT devices and services are in place.

Following the proposed ENISA recommendations, we decided to work on the security of the networked architecture as a whole. It included:

- Encrypting the data sent though HTTP connection between IoT devices and the Cloud server.
- Allowing access to database on the Cloud only for our custom API located in the same server.
- Adding log-in for database Web Interface.
- Enforcing HTTPS protocol on devices accessing database Web Interface.

Adding a security layer to the IoT device proved to be more difficult than expected, as low powered board restrains from many options widely available for OS running systems. The main issues are:

- Low-powered boards are not powerful enough for secure communication over HTTPS. Key negotiation (handshaking) procedure used by HTTPS requires a high amount of CPU and memory capacity.
- A lack of HTTPS connection limits the selection of viable options, as the key renegotiation is automatically compromised.
- A fragmentation of IoT boards specifications results in lack of well developed and supported security libraries tailored for specific memory and CPU parameters.
- The memory and CPU overhead added by security layer barely fits for device processing capabilities. This can cause device stability issues (E.g.Random restart caused by the lack of available memory).

To overcome the lack of security even with these issues in place, we had to look for alternative solutions that ensure safe data transfer from our IoT device to Cloud server via API and back. As NodeMCU is a low-powered device, there were not many options to choose from, and we decided to implement a two-level encryption.

We did in-depth research on the AES encryption, and performed the tests on our embedded device, using AES and Base64 libraries, as well as CryptoJS and Crypto on the Cloud implementation. At the end though, we found that when a CryptoJS encrypts the text with Key and a random IV, instead of a user-specified static IV, then the AES algorithm works differently than the Arduino implementation. Due to such compatibility issue, we decided to leave encryption for future work.

Before we identified an issue, the AES setup functioned as follows:

Before the message is sent through HTTP, our device generates an Initialization vector(IV) - an arbitrary number which prevents repetition in the next step of encryption, reducing chances of finding a pattern and breaking the cypher. The IV is used in combination with AES key (hardcoded in IoT device and on Server) to encrypt the message with AES algorithm using CBC mode. Finally, an IV vector is converted to decimal notation and added to the start of an encrypted message, and the resulting cypher is sent to the server via POST request.

AES encryption

IoT device runs a 128-bit version of AES, as the more complex AES-192 and AES-256 not only add bigger overhead, but are susceptible to (theoretical, yet not currently computationally feasible) related key attack as well [16]. AES is block-based encryption, taking a plaintext message, breaking it up into blocks of data and then using a static AES key (saved on device and server) alone (ECB) or together with IV and previous blocks (CBC) to encrypt the text.

AES encryption with ECB

At first, we implemented a base version of AES - ECB, as shown in the example of ESP8266-specific AES library page. The solution did not require the generation of IV keys and shared the same AES key for every block. The extreme simplicity of this option means that due to shared key, blocks have similar patterns even after encryption, thus making data still susceptible to eavesdropping4.1. As a result, we decided to use more complicated, yet a lot more secure AES mode - CBC.

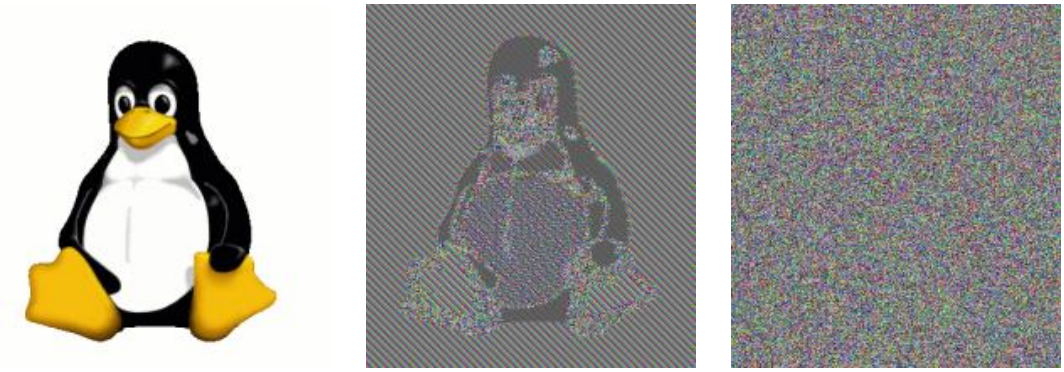


Figure 4.1: Image encryption: Unencrypted, AES with ECB, AES with CBC [21]

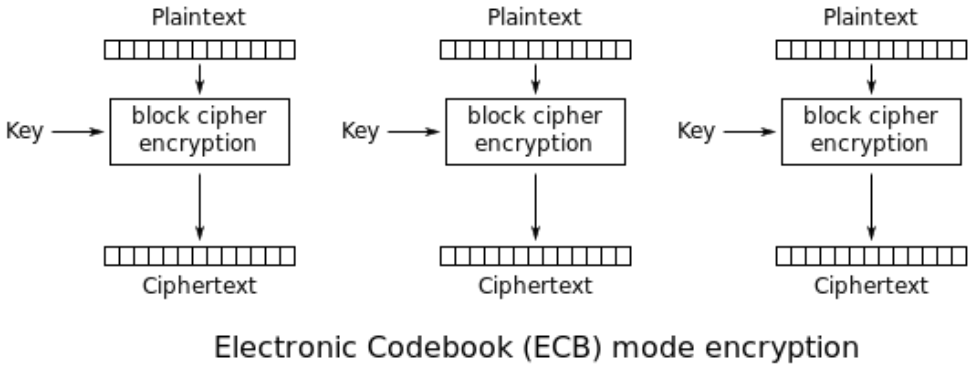


Figure 4.2: ECB encryption diagram.[21]

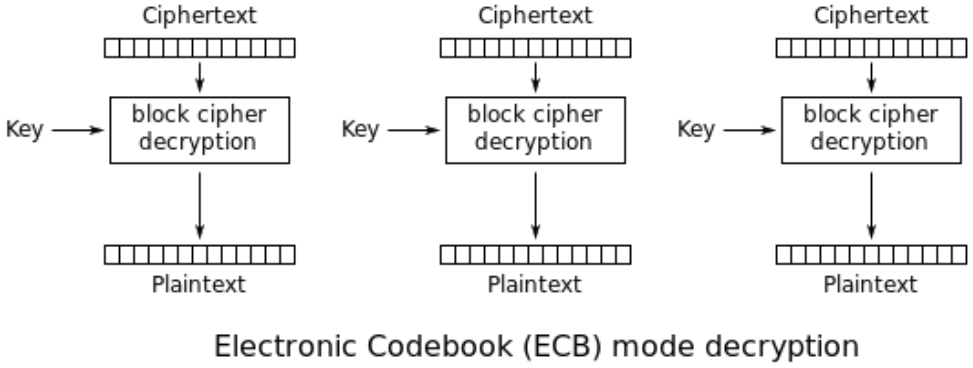


Figure 4.3: ECB decryption diagram.[21]

AES encryption with CBC

CBC version is also running a 128-bit AES version, yet it introduces the initialisation vector into the equation, as well as some additional measures. In CBC, a plaintext message is once again broken into blocks of data. The first block is then encrypted using randomly generated IV together with static AES key. For the following blocks, it takes a part of the previous block and uses it as IV, combined with AES key, to encrypt a current key. The result is a message which has no pattern similarity with initial data, ensuring privacy.

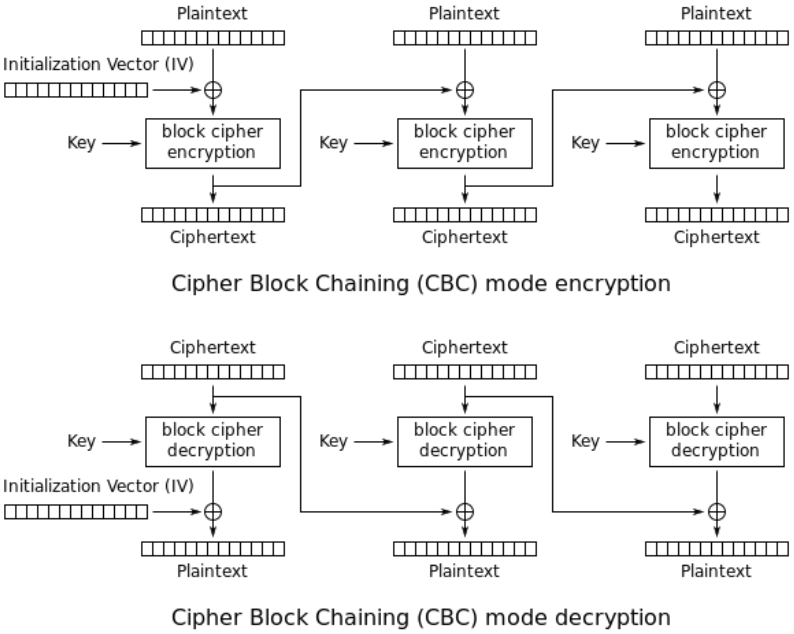


Figure 4.4: CBC encryption and decryption diagram.[21]

Initialization vector
We are using the IV key to mitigate the problems that arise from the repetitive use of the same AES key. By generating and using a new initialisation vector every time when AES encryption is used, we ensure that resulting ciphertext has no similarities to previous encrypted messages.

Communication with a Cloud

Our setup relies on an HTTP connection to send the data from NodeMCU to the API on our Cloud server. We are aware of the security problems that arise with the usage of this protocol; thus a lot of time was put into the research of possible HTTPS connection implementations. Using HTTPS on a device with limited processing power and memory resources creates many stability issues though. The overhead, added by HTTPS, makes the SoC run at the edge of its limits, making the risk of device restart or crash extremely probable. Even the developers of ESP8266 libraries such as ESP8266WiFi, offering HTTPS/TLS support for ESP8266 chip found inside the NodeMCU board, are stating that it is not a good choice for a device that is meant to run reliably for extended periods of time [59]. Whenever IoT device communicates with a Cloud API, it makes a POST request to the server. Depending on parameters passed to the Cloud, a response is then sent back and NodeMCU shows it on a display. There are four situations in which our prototype queries the API:

1. Reading Twitter feed - a query is sent to the API, specifying 'twitter' as a required service. Together with a service name, a device ID is sent (for device owner identification and parsing of different Twitter feed). Cloud then sends back a response with a most recent tweet. After NodeMCU receives a response, it performs a usual set of actions - decrypting a message and showing it on LCD.

2. Sending Twilio SMS notification - a query similar to the first one is sent, with service name 'Twilio' and device ID for owner identification (for database search of assigned phone number). Correct query initiates Twilio service to send an SMS notification to a pre-set number. Server response contains a message which states if an SMS was sent successfully.

3. Marking event participation - a query is sent to the Cloud, specifying UID of RFID Card that was scanned and a unique ID of an IoT device. Server response contains a data notifying the student of the participation status. The possible messages sent back are:

    - Message stating that users presence was marked successfully.
    - Directions to another tutorial room if a participant mixed up locations.
    - Response that student has no assigned events taking place at that moment
    - Statement saying that no student is associated with the received card details.

4. OpenWeather weather forecast querying - a query specifying 'weather' service and device ID is sent, asking for a weather forecast. The returned response contains text describing weather conditions for device location during the upcoming 3 hours.

Power Management

NodeMCU power efficiency solutions - The developed prototype is meant to be used on battery power, requiring charge as rarely as possible, with around a week-long battery life. To reach this goal, we worked on optimizing many parts of the software, as well as hardware. On a software side, we chose to use C++, a low-level programming language, which allows the use of extensive power saving modes. These settings are available only on low-power boards such as NodeMCU, used as our final implementation main board, and for Arduino Uno, the first iteration of the main board. NodeMCU has three applicable sleep modes shared between devices with ESP8266 SoC:

Item		Modem-sleep	Light-sleep	Deep-sleep
Wi-Fi		OFF	OFF	OFF
System clock		ON	OFF	OFF
RTC		ON	ON	ON
CPU		ON	Pending	OFF
Substrate current		15 mA	0.4 mA	~ 20 µA
Average current	DTIM = 1	16.2 mA	1.8 mA	-
	DTIM = 3	15.4 mA	0.9 mA	
	DTIM = 10	15.2 mA	0.55 mA	

Figure 4.5: Differences of ESP8266 sleep modes [27]

- Modem-sleep

  ```
  wifi_set_sleep_type(MODEM_SLEEP_T);
  ```

  Modem-sleep mode is automatically enabled (or can be enforced with the line of code above) when NodeMCU connects to the router in station mode. The board stays connected to the router through the DTIM beacon mechanism. A power saving is a result of WiFi module turning off between the intervals of Beacon arrivals from the router.

- Light-sleep [34]

  ```
  WiFi.mode(WIFI_STA); //Enable WiFi station mode.
  wifi_set_sleep_type(LIGHT_SLEEP_T); //Light Sleep on
  delay(100); //sleeping for 100ms
  ```

  Light-sleep is similar to Modem-sleep, only with the addition of powering off the CPU clock and suspending the ESP8266 CPU for extra power saving. When a mode is enabled, a board changes WiFi mode to STA(station / standard WiFi client) goes to light sleep on a delay() function call.

- Deep-sleep [27]

  ```
  ESP.deepSleep(30e6); //Sleep for 30s and then wake up.
  ```

  A most intensive software-only power saving mode. The chip will turn off Wi-Fi connectivity and data connection, with only the RTC (Real Time Clock) continuing to work (and the RTC memory). RTC is thus, responsible for periodic wake-ups between deep-sleep episodes. A line of code above, puts NodeMCU into a deep sleep for 30 seconds.

Our current software implementation is using a Light-Sleep power saving option. Since Deep-Sleep mode introduces inconveniences such as longer wake-up times and slower response to user input, the 27% difference in energy efficiency (19 mA vs 14 ma) was not seen as sufficient motivation.

If we would be using an Arduino board (which does not have a well supported WiFi implementation) - The available power saving modes (From lightest sleep "Idle" to most low power "Power down") would be:

1. SLEEP_MODE_IDLE - stops the CPU while allowing the SRAM, Timer/Counters, USART, 2-wire Serial Interface, SPI port, and interrupt system to continue functioning.
2. SLEEP_MODE_ADC - stops the CPU and all I/O modules except asynchronous timer and ADC (Analog to Digital Signal Conversion), to minimize switching noise during ADC conversions.
3. SLEEP_MORE_PWR_SAVE - the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping.
4. SLEEP_MODE_STANDBY - the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption.
5. SLEEP_MODE_PWR_DOWN - saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset.

	Active Clock Domains					Oscillators		Wake-up Sources							
	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other I/O	Software BOD Disable
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X(2)	X	X	X		
Power-down								X(3)	X				X		X
Power-save					X		X(2)	X(3)	X	X			X		X
Standby(1)						X		X(3)	X				X		X
Extended Standby					X(2)	X	X(2)	X(3)	X	X			X		X

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

Figure 4.6:
Active Clock Domains and Wake-up Sources in the Different Sleep Modes[13]

During the initial prototyping stages, using Arduino Uno, several solutions of power consumption saving were tested out:

- Power-down sleep mode is used during the time when the device is in use for the Smart Office environment. It is using watchdog timer, a clock separate from CPU, to set a device to sleep for 8 seconds after every loop. If a person tries to interact with the device by pressing either of two inbuilt buttons, an interrupt will trigger a new loop cycle, immediately getting the CPU and other parts out of the sleep state.

- Setting unused Digital pins to IO LOW reduces consumption on Arduino board by around 2 mA (The board consumes around 15 mA on Power-save and 52mA on Idle).

Throughout the development time, we made a great effort to stick to C++. It was an obvious choice for IoT prototyping, as most of the low powered boards support this low-level language, thus if a decision to change the main board is made, just like we did when switching from Arduino Uno to NodeMCU, the codebase support, along with options for power saving features, remained.

4.2 Cloud setup

The Cloud is a most vital part of our project. It is a virtual machine on AWS (Amazon Web Service) EC2 service, functioning as the main hub for all IoT boards. For the Cloud setup, we chose one of the cheapest-tier virtual machine options on Amazon, running Ubuntu 16.04 LTS Linux distribution with 30GB storage, 1GB of RAM and one virtual CPU. Even then, it is fully capable of quickly handling IoT devices requests via Custom API, performing the tasks that low-powered boards are unable to achieve due to hardware limitations and managing an SQLite database. Our current server setup is responsible for:

- Hosting an SQLite database.
- Providing database management layer via PHP RESTful API (ArrestDB).
- Communicating with Twitter, Twilio, OpenWeather API's.
- Providing Custom API for communication with IoT devices.
- Hosting web interface for the database.

At the start of software development, the server was meant only for the SQLite database, and API (handling SQL queries and communication with the device) used event participation monitoring task. With every iteration of prototype embedded code, we offloaded more and more of the device functionality onto a server. Moving most of the codebase away from the board resulted in:

- Lower NodeMCu memory usage (leaving more space for overhead added by encryption of outgoing messages and decryption for incoming messages)
- Secure communication with API's, avoiding the HTTP protocol used by the boards, and using HTTPS with OAuth for Twitter.
- More future proof implementation. In case any of the used API's (Twilio / Twitter / OpenWeather) get updated without backwards compatibility (stopping to use the same format of communication with the users), breaking our current implementation, it can be fixed by making relevant code changes for a Custom API on a Cloud. If a direct communication with these API was set up on IoT device,

any issue with software support would require all IoT devices to be updated on site, with a cable acces to the main board.

- Adjustable implementation. On click, the buttons of an IoT device are sending a command with encrypted trigger word that initiates an action on a server-side. As a result, the cloud action assigned to a command can be reprogrammed to perform different task while maintaining the same trigger word. This approach allows to change the functionality without reprogramming the embedded systems.

Cloud Service [56]

Our server is running on an IaaS product, the Amazon Web Services (AWS) EC2 cloud platform. Before settling on this cloud computing service, we had to evaluate other available options against a list of software that we had to develop (Shown at the start of 4.2). All possible choices can be split into three categories: IaaS (Infrastructure as a Service), PaaS (Platform as a Service), Dedicated Server.

IaaS (Infrastructure as a Service)[71]
Google Compute Engine, Amazon EC2, Microsoft Virtual Machine
IaaS infrastructure provides on-demand access to virtual machines / data storage / databases or scalable and automated compute resources. The provider is responsible for networking, storage, servers, virtualization. All of the software of a virtual machine is controlled by the user, from a choice of OS to installed packages, running programs, databases and more. As costs scale depending on used resources, IaaS offers scalability and flexibility, without and commitments to purchase of specific server hardware.

PaaS (Platform as a Service)[71]
Google App Engine, Google IoT Core, Amazon IoT Core, Amazon IoT Analytics
PaaS is a more closed down service than an IaaS, as the provider is responsible not only for networking, storage, servers, virtualization but the choice of OS and available software(code compilers, packages, etc.). With all the setup in place, the user only has to develop an app and manage the incoming and/or outcoming data.

Dedicated Server[71]
Providers: GoDaddy. 1&1, etc.
Renting a dedicated server is an option if a complete control over the server is required. This choice is less flexible, yet more configurable alternative to IaaS. While IaaS offers a virtualised setup, a dedicated server allows the user to select a specific hardware setup, and pay a flat rent fee assigned to it, independent of the received demand for resources. Due to the smaller level of abstraction, a server software setup can be optimised to a higher level, taking the hardware specifications into account.

The decision to use a virtual machine setup, an IaaS, was made due to the issues we had with other options:

1. The initial idea of using Google App Engine (PaaS) was not successful because

of limitations inherited from provider setup. We were hoping to use PHP for API development, yet Google App Engine did not have PHP modules for HTTP connection. Furthermore, a lack of modules for SQLite integration made editing SQLite database via PHP script impossible. A more restricted nature of PaaS meant that every single one of required software pieces (database, IoT communication API, database management API, database Web interface) had to be launched as separate instances, using not only App Engine service, but Cloud SQL (database service) as well.

2. IoT tailored PaaS solutions, such as Amazon IoT Core, were another option. We wanted a whole IoT setup including not only API's, but also database and web interface running on a same virtual machine (for easier migration between providers and simplicity). The IoT Core could not satisfy this requirement and was dismissed.

3. A choice of dedicated server was too limited and financially unfeasible for a proof of concept. A monthly fee of at least £40 offered by 1&1, and £135 by GoDaddy, meant that our monthly expenses would be at least five times higher than on AWS EC2 (costing £7.2 per month). The huge price difference is a result of the only available setups being high-performance servers (4 Cores, 16 GB RAM), which are unnecessary for an application that uses at most 10% of processing power available on a single core, 1GB RAM machine.

In the end, AWS EC2 was a most suitable choice, due to the simple setup process (launching a virtual machine instance in 5 minutes), quality tutorials by both Amazon and other programmers, and an ability to pause an instance when it is not required (saving money in the process). The most important feature of EC2 IaaS though is freedom of software setup, allowing us to have a database, APIS and Web interface running on the same machine, using the programming languages their modules and database setups that we wanted to try out.

Virtual machine configuration

Since we were developing a proof of concept solution, our goal was to keep the Cloud implementation as simple and light as possible, without a compromise on security or performance. The final solution represents an IoT application prototype similar to the solutions full-time IoT developers create, meaning that easy integration into an existing software infrastructure of an enterprise or university had to be taken into account.

The configuration includes:

- ArrestDB RESTful database API, adding a level of abstraction for simpler communication with other services used within organization.
- Relational database (SQLite) used due to wide popularity of SQL databases in enterprise, allowing easier project integration with existing DBMS'es (Database Management Systems).
- Modified Web Server configuration, redirecting all HTTP connections to HTTPS. (Except the Custom IoT API that runs HTTP due to IoT board limitations).

Apache Web Server [48][23]
Our virtual machine is running Apache Web Server software, which is using TCP (for establishing and maintaining the connection) / IP (for data transfer via small packets) protocols to handle network communications, listening to the ports to which requests are sent. It then analyzes the received headers to find out a type of request method (GET / POST), a specified location of a required page/file (address of Custom API / Web Interface address / ArrestDB API) and a version of HTTP/HTTPS (HTTP for Custom API, redirect to HTTPS for other requests).
If the request is successful (address exists), response with status code 200 is sent, together with response headers and requested data.
Otherwise, for example, when a file is not found, a response with a header containing error message is sent.

HTTP vs HTTPS
Safe exchange of information between the client and the server was ensured with the use HTTPS(Encrypts data with SSL before transmission) instead of HTTP(Sends data over the network as plain text) in every possible area. Our Cloud system automatically fetches and deploys SSL/TLS certificates from "Let's Encrypt" CA (Certificate Authority) via a CertBot ACME Client, allowing us to have a working HTTPS protocol.

Cloud API

We developed the Cloud API to be the main link between IoT devices and system on a server. The programmed Cloud API is:

- Handling requests made by IoT devices.
- Encrypting the incoming and decrypting the outcoming messages between IoT devices and the API (With NodeJS script using CryptoJS module).
- Interacting with Twitter/Twilio/OpenWeather APIs in response to IoT devices requests.
- Modifying/Reading data on SQLite DB via ArrestDB API for Event participation monitoring functionality.

The API is used to overcome limitations of our low power IoT devices. These small boards are unable to support HTTPS and OAuth, thus due to lack of security neither Twitter, Twilio APIs, nor our SQLite database should be directly accessible to them. The Cloud API functions as a secure gateway for both of the sides, communicating with NodeMCU boards via HTTP combined with AES encryption, and interacting with all the other services through HTTPS (and OAuth where necessary).

Processing IoT device request

- Every request arriving from IoT device to the Cloud API is combined out of Initialization Vector (IV) and encrypted message.
- Upon receiving the data, Cloud API passes it to NodeJS script with CryptoJS module, which decrypts the message using IV and a static key (AES key is saved

both on server as well as on IoT device). Decrypted data is then sent returned to PHP script.

- API reads the decrypted data containing a name of requested service, along with necessary parameters(RFID ID / Device ID) and initiates actions associated with the service.
- After a requested service is finished, the answer containing response from a service is generated. The message is then encrypted with CryptoJS using new random IV, along with static AES key and then sent back to IoT device.

Twitter
Using an OAuth access token, OAuth access token secret, consumer key and consumer secret generated by Twitter developer toolkit, a get request is initiated to Twitter API through HTTPS connection. Along with these parameters, a username of required feed is sent. A response is retrieved in a format of JSON object, containing tweets and other information about the of an IoT devices owner Twitter account activity. The data is then processed, a most recent tweet is extracted, encrypted and sent back to IoT device.

Twilio
The SMS sending via Twilio is done by using a dedicated Twilio REST API Client, using Account SID and Token acquired from Twilio developer console. A message is sent from a number we purchased from Twilio developer dashboard to a number of device owner saved in the API. Currently, for proof of concept, our an implementation uses a free credit allocated at the time of registration, binding us to send notifications to one specific number. If a paid account would be used, SMS could be sent to multiple numbers, without requiring many alterations in the codebase (adding a query to DB, requesting numbers of different PTs).

OpenWeather
Weather data pulling is the least complicated task of our API since it relies on a method that has no security measures apart from HTTPS added to supplied data. Weather data is widely available and public; thus a JSON with all weather information is easy to obtain. A retrieved dataset is then formatted, keeping only the forecast of next 3 hours, encrypted it and sent back to an IoT device.

Event Participation Monitoring Tutorial checking is also handled by the same API. On arrival of the request, a timestamp is taken and used to parse database tables via ArrestDB RESTful API along with device ID and student RFID card ID.

- ('Tutorial name' + tutorial presence marked) message is sent as a response to IoT device if the student is at the right tutorial room and within required timeslot.
- (Wrong room, you have 'Tutorial name', go to 'Tutorial location') is a response if a student has a tutorial at that specific timeslot, yet appears in the wrong room.
- (You have no tutorial at this time, go home.) response is sent if the details match to a student in a database, yet there are no events assigned to the person at that time.

- (No student associated with this card) message appears if RFID ID does not match to any student on a Students database table.

With the shown differences in requirements of clients, APIs and authentication for every service access, it would be an incredibly complex task to have an IoT device do all of these actions on its own. Also, updating the software on IoT device is more difficult than on a cloud. If any of APIs would over-go a change in the way of access or data retrieval, all of the devices would need to be brought back to the workshop and updated via a USB wire. Instead, a change on an API script is all that would be necessary in case of our Cloud solution.)

ArrestDB [14]
ArrestDB is a RESTful API mapping directly to an SQLite database. The ArrestDB supports not only SQLite but MySQL and PostgreSQL, thus adding a layer of flexibility to our IoT project implementation. We are using this API to enable database querying and editing from our own PHP script. With ArrestDB there are four main actions, all initiated with a different type of request being sent over HTTPS and specifying table and list in the URL:

- GET - acquires data from the database.
- POST - creates a new row containing data sent with the request.
- PUT - updates the database fields that correspond with PUT data.
- DELETE - deletes a row of a table which matches sent parameters.

CryptoJS [70]
The Cloud API is using the same type of encryption as the IoT device, with the main difference being the libraries, modules and programming languages used to achieve the same result. The NodeMCU board is using an ESP8266-specific AES library with an Arduino code. For the server side, we created a NodeJS script which encrypts and decrypts via functions from a CryptoJS JavaScript library. As mentioned in the IoT device section on encryption, a 128-bit version of AES in CBC mode is used together with a static AES key and a random Initialization Vector. The NodeJS script is called from a PHP script, whenever encryption or decryption is required.

Database Web Interface

As an addition to the whole proof of concept IoT system, we developed a simple web interface. It is based on a PHP script handling the Login page4.7, processing JSON data from ArrestDB API and loading HTML, CSS and Bootstrap for the front-end.

The simple interface is meant to be used by the staff of an enterprise or university, who want to look at the database, yet have no programming knowledge. Currently, after logging in, the interface displays a structure of our database tables4.8 and does real-time querying of SQLite tables to show the existing data4.9.

We did not focus on adding a full database management functionality to the website and allocated most of the time on the IoT device an server development. Even then, the web interface solution was enabled a more interactive experience for people attending Honours Project Feedback Presentations Session, allowing to better understand the structure of a server.

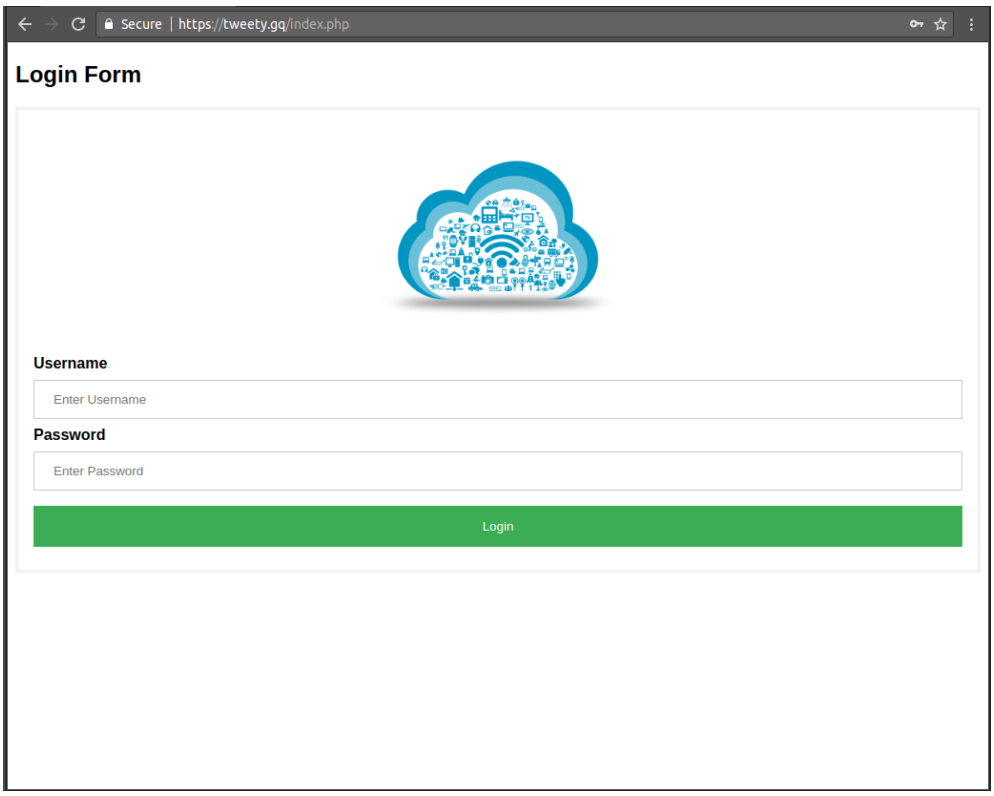


Figure 4.7: Web Interface: Login Screen

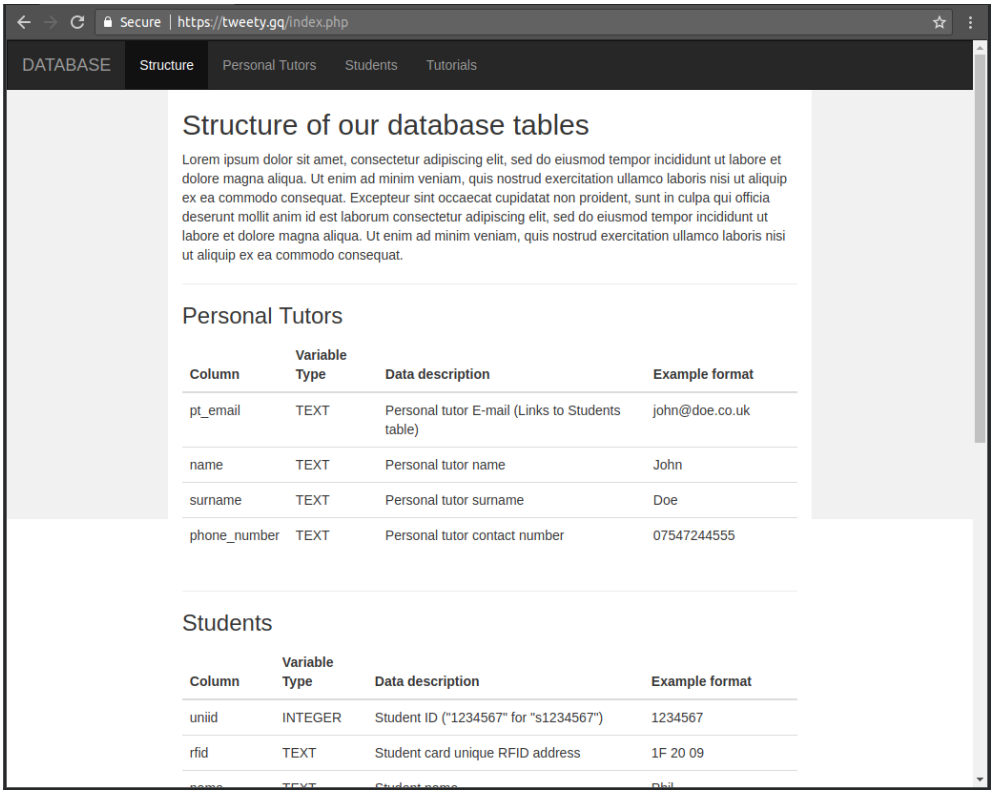


Figure 4.8: Web Interface: DB structure view

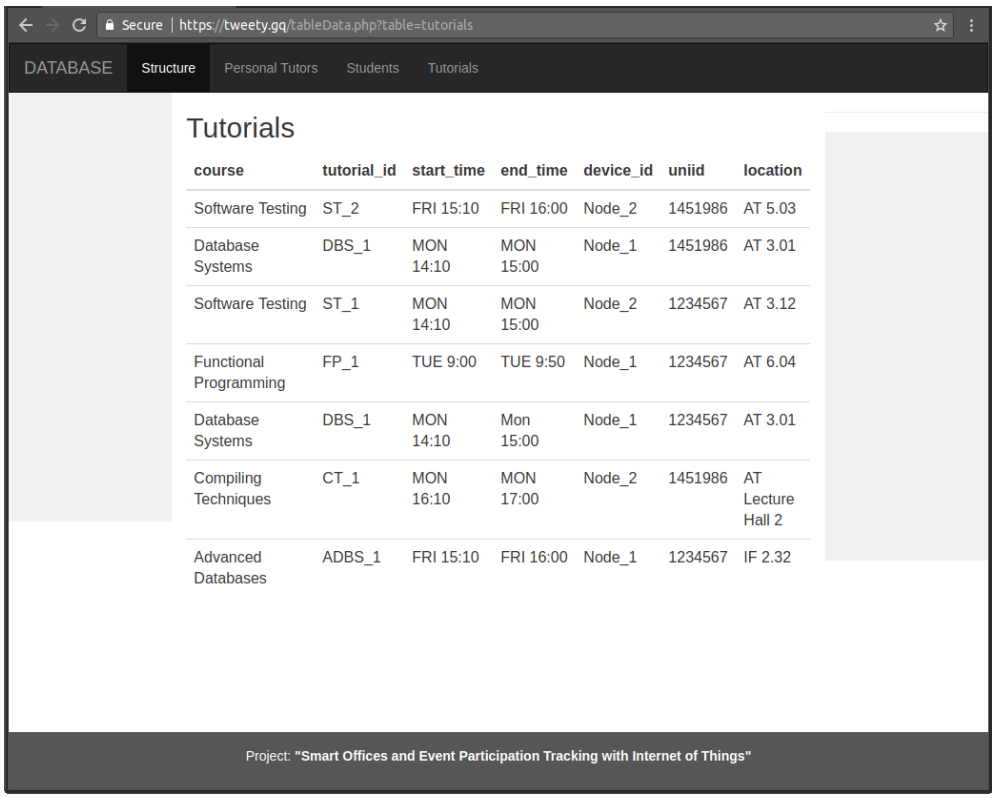


Figure 4.9: Web Interface: Live Tutorials table view

SQLite database [63]

When chose a relational, SQL based solution, SQLite, for our IoT infrastructure. The decision was based on the fact that RDBMS (Relational Database Management System) databases are the most popular among programmers and companies[**?**], ensuring our project would be easily applicable to existing infrastructure of the enterprises.

Even though SQLite is not as widely used as MySQL, SQL Server or PostgreSQL relational databases[66], it is a lot simpler to set up and run than the alternatives. SQLite is based on a single cross-platform file, containing a whole database (tables, indices, data, etc.), while MySQL or PostgreSQL function as stand-alone database servers. The single-file based SQLite system thus offers better performance than more complex alternatives, mainly because SQLite integration works with directs calls to file holding the data, unlike the solutions like MySQL, communicating through MySQL daemon process(a process that runs in the background, serving DB queries).

SQLite lacks the support for concurrent writing to a database, relying on a writer queue instead. For a prototype application or a small/medium business, it is still completely sufficient, as a possible delay of a few milliseconds for participation marking functionality does not impact overall performance.

If there still is a need for more complex solutions enabling concurrent queries and data recovery, our software is compatible with other SQL alternatives. ArrestDB RESTful

API, currently used to interact with SQLite DB, supports more complicated MySQL, PostgreSQL relational databases as well. As a result, with minimal ArrestDB settings adjustment, we can cover around 3 out of 5 most used DBMS'es by professional developers (according to 2018 StackOverflow survey ranking databases popularity [37].

Database structure

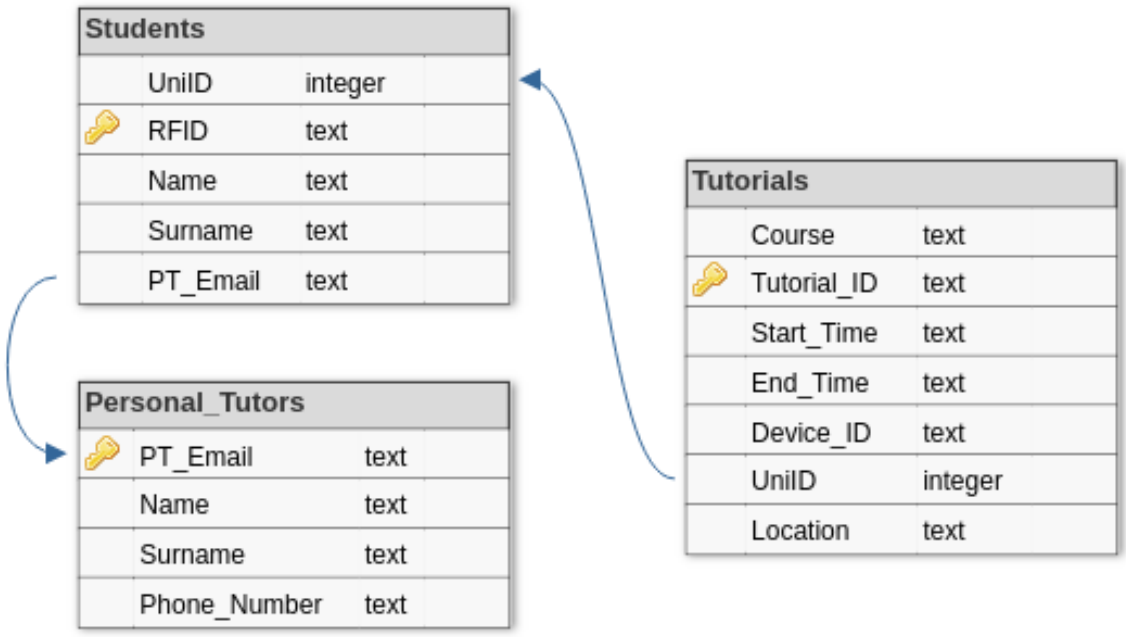


Figure 4.10: Database structure. Keys indicate Primary keys of tables, Arrows indicate parameters that link the tables.

Chapter 5

Evaluation

We evaluated the project from two different standpoints, namely regarding technical performance of the implementation and usability via a user study. The split approach is motivated by the fact that even the best execution of a consumer-oriented IoT hardware and software is worth nothing if demand for the product simply does not exist.

According to statements made by companies participating in one of the biggest IoT events - IoT World Forum (May 2017)[44], a collaboration between IT and business sides is the key factor behind a successful IoT project. Even when 35% of IT technical leaders consider their IoT solution a success, only 15% of the business leaders are satisfied with the end product. The 20% difference is due to disproportional allocation of resources right from the start of IoT solution development, immediately focusing on technical implementation, leaving out the research on market demand, use-cases and identification of target audiences [39].

Our evaluation of the system's performance, measuring scalability, security, speed and reliability, represents a critical technical review. The feedback surveys, interviews and project presentations approached evaluation from a business standpoint. For the market and user study, we analysed patterns of interaction with the device, identified missed out features. Most importantly, we assessed the level of interest that the public has for our IoT solution before and after the changes made according to user feedback.

5.1 System Performance Evaluation

5.1.1 IoT Device

Power consumption
The power savings of our device are reduced to the lowest amount available with existing hardware. By taking additional steps, such as converting a whole system to 3.3V (which would allow to solder-out a power-hungry 5V to the 3.3V converter on the NodeMCU board) and removing the status LEDs on all devices. A custom alteration of the boards does not represent a prototyping process. Thus we leave this step for future improvements (when the device is being tailored for mass production).

Power measurements	
Measured state	**Consumption**
Full Setup (NodeMCU, RFID, QC2004LCD). No WiFi Connectivity	$\sim 152-158$ mA
Full Setup (NodeMCU, RFID, QC2004LCD). Searching for WiFi	$\sim 137-140$ mA
Full Setup (NodeMCU, RFID, QC2004LCD). Connected to WiFi	$\sim 128-131$ mA
NodeMCU without power saving	$\sim 84-92$ mA
NodeMCU with light sleep	$\sim 16-27$ mA
NodeMCU with deep sleep	$\sim 12-14$ mA
QC2004 LCD (White on Blue)	$\sim 23-36$ mA
QC2004 LCD (Black on Green)	$\sim 26-38$ mA

RFID scanning

The iterations of our 3D printed enclosures had different settings of the filament (amount of plastic in the inside layer of the enclosure walls) as well as varying temperature and other 3D printer settings. We optimized the plastic structure design to provide the least amount of RFID signal suppression, while maintaining an enclosure rigid.

RC522 RFID Card Scanning tests	
Placement settings	**Measured range**
Without enclosure	34mm
Enclosure V1	24mm
Enclosure V2	23mm
Enclosure V3	28mm

5.1.2 Cloud API

API's Response speed - We tested the Cloud performance by creating a script that sent requests to Cloud API, measuring the response speed of Smart Office functionality (Twitter feed, SMS notifications, weather forecasts) and Event Participation Monitoring application (database querying via ArrestDB).

Cloud API serving Event Participation Monitoring requests

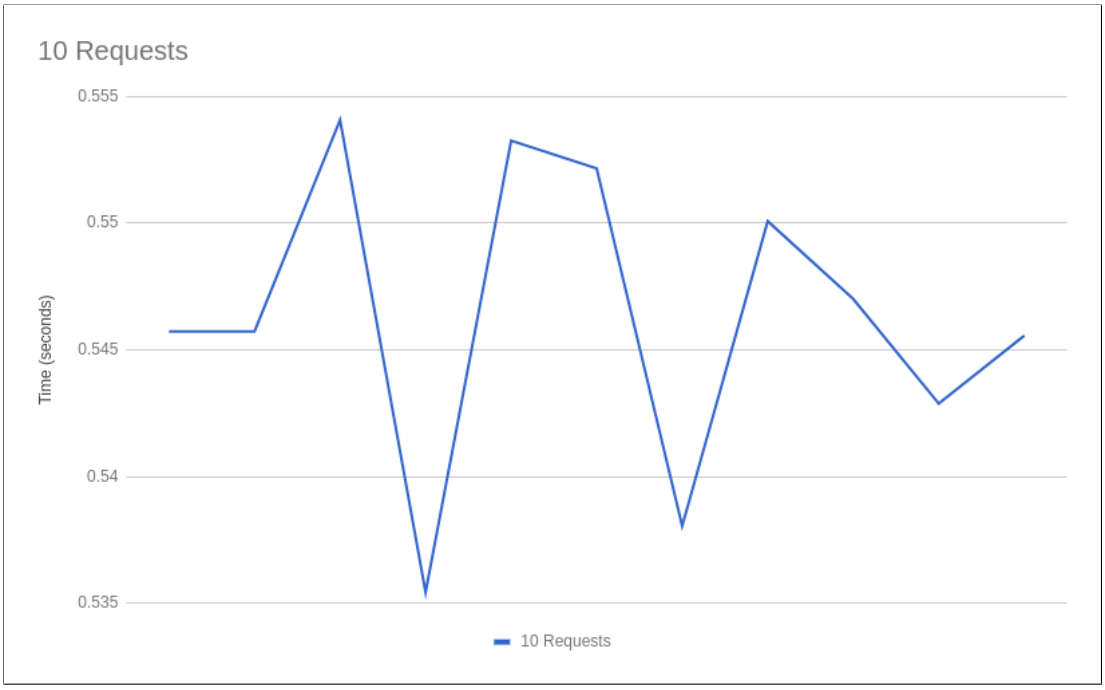


Figure 5.1: Event Participation Monitoring response time (10 requests)

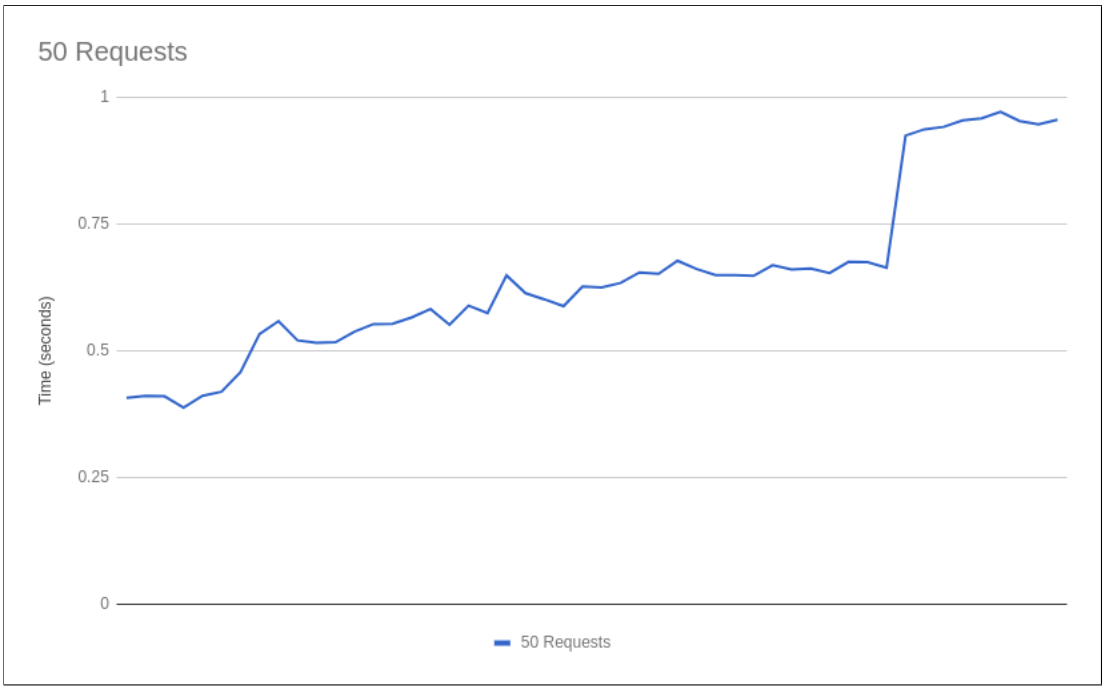


Figure 5.2: Event Participation Monitoring response time (50 requests)

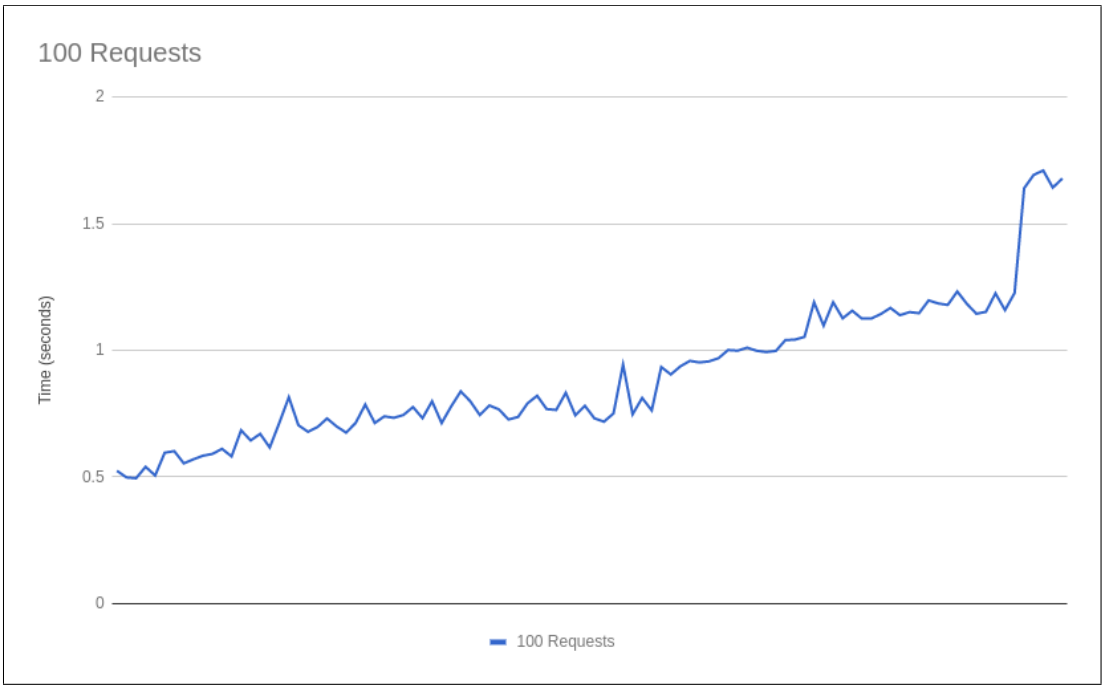


Figure 5.3: Event Participation Monitoring response time (100 requests)

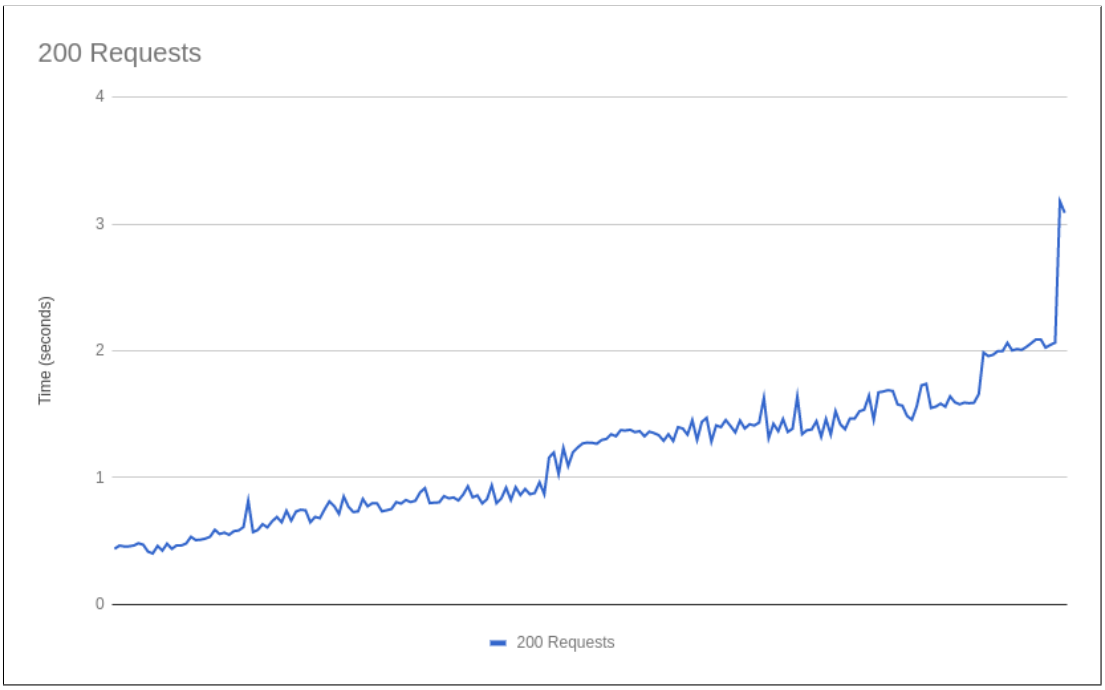


Figure 5.4: Event Participation Monitoring response time (200 requests)

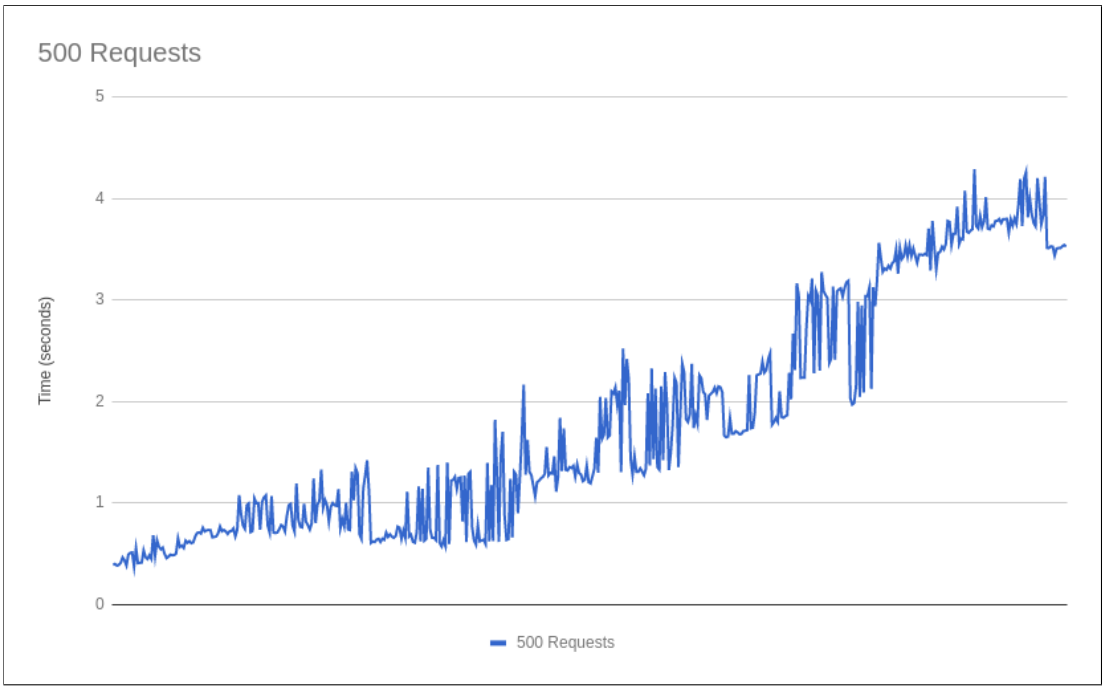


Figure 5.5: Event Participation Monitoring response time (500 requests)

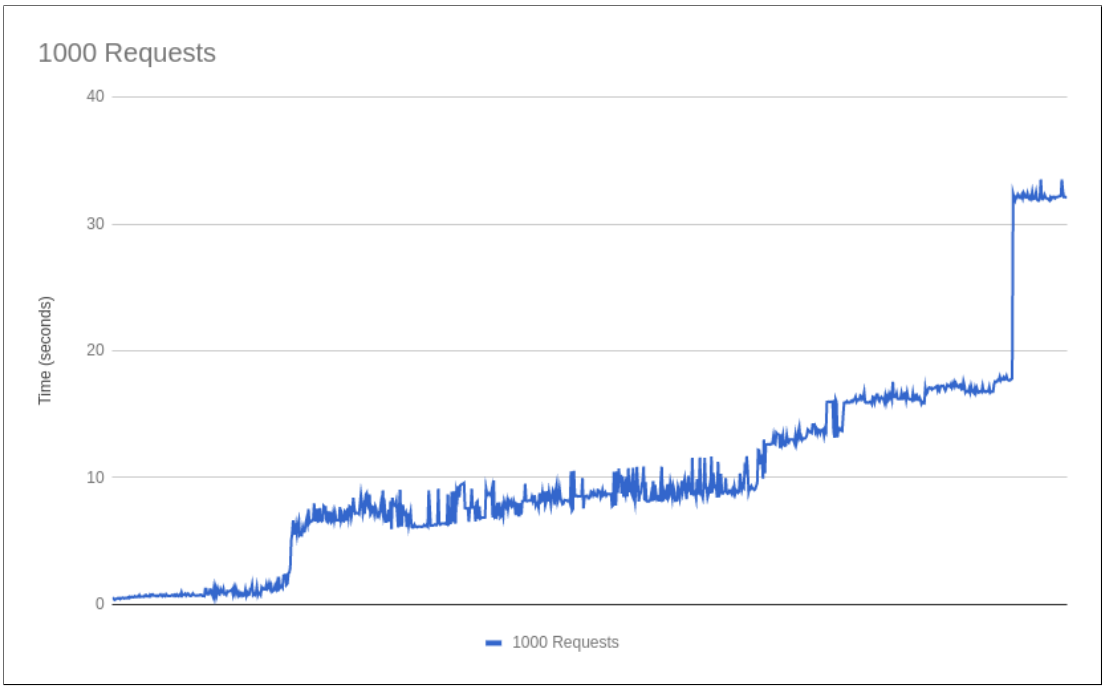


Figure 5.6: Event Participation Monitoring response time (1000 requests)

Cloud API serving OpenWeather forecast

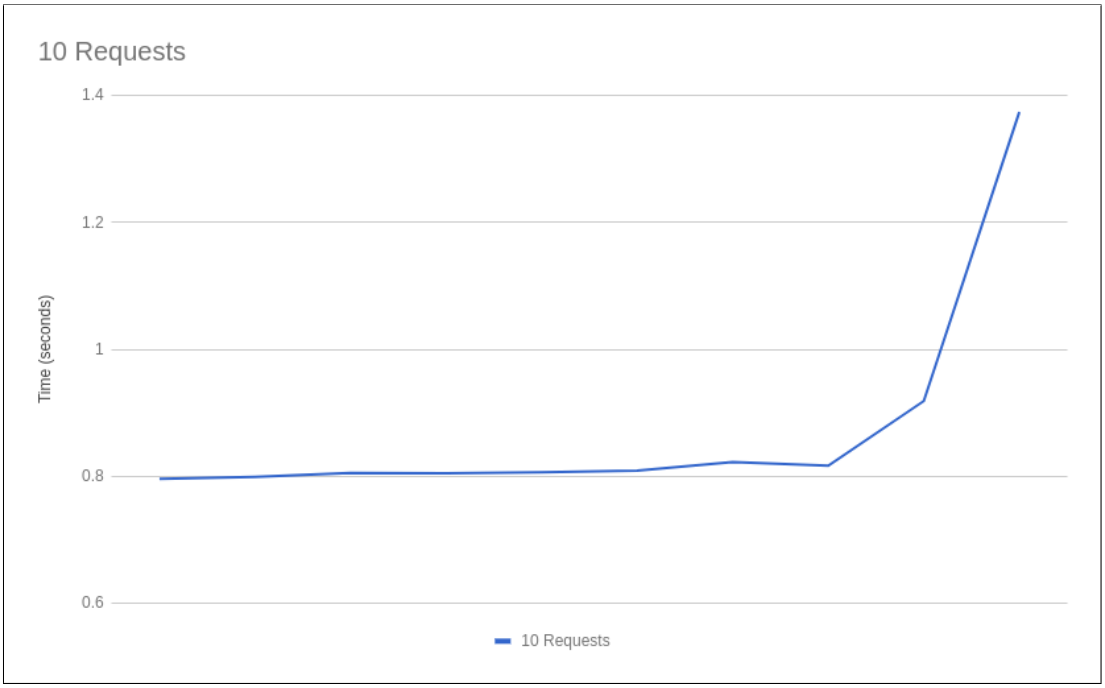


Figure 5.7: OpenWeather response time (10 requests)

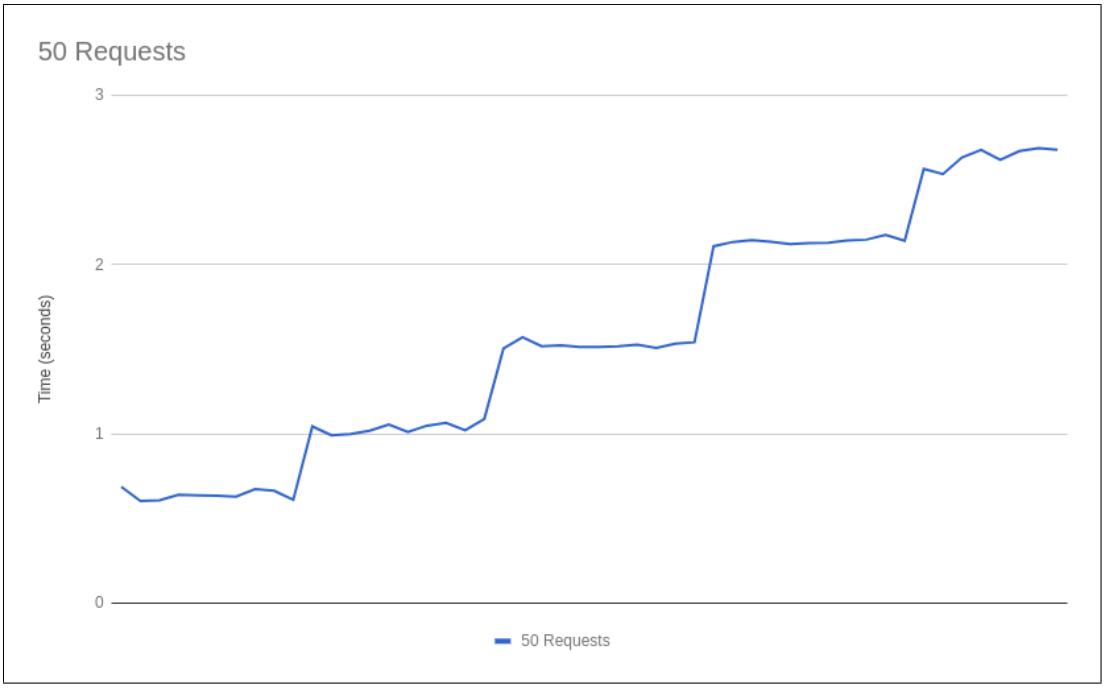


Figure 5.8: OpenWeather response time (50 requests)

Twitter feed

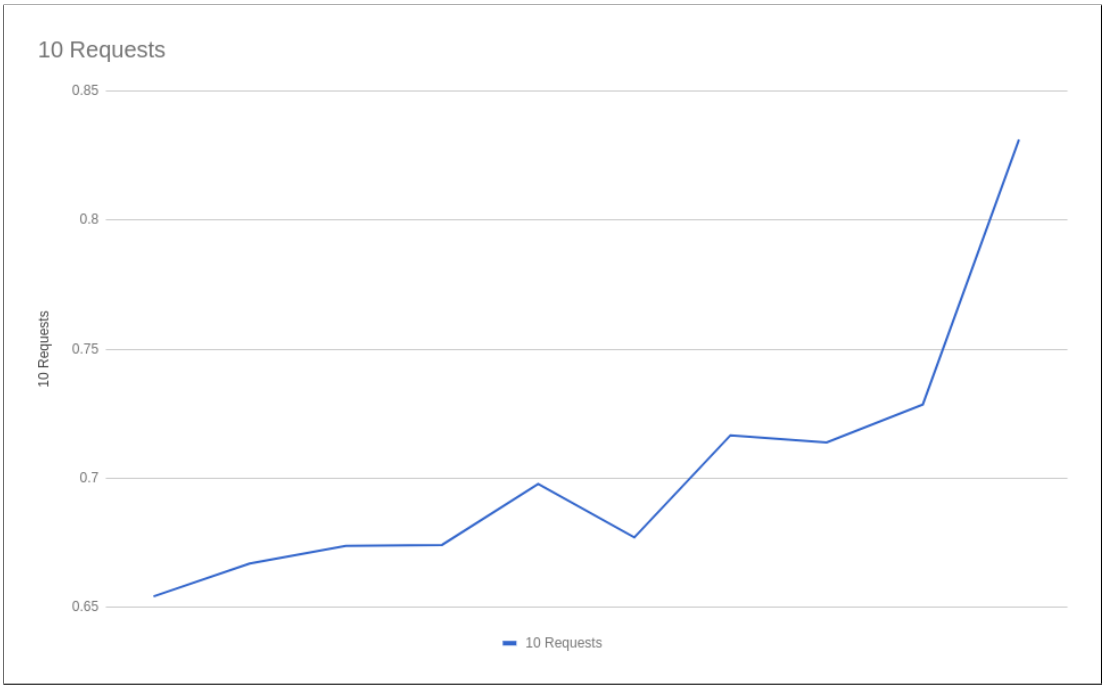


Figure 5.9: Twitter response time (10 requests)

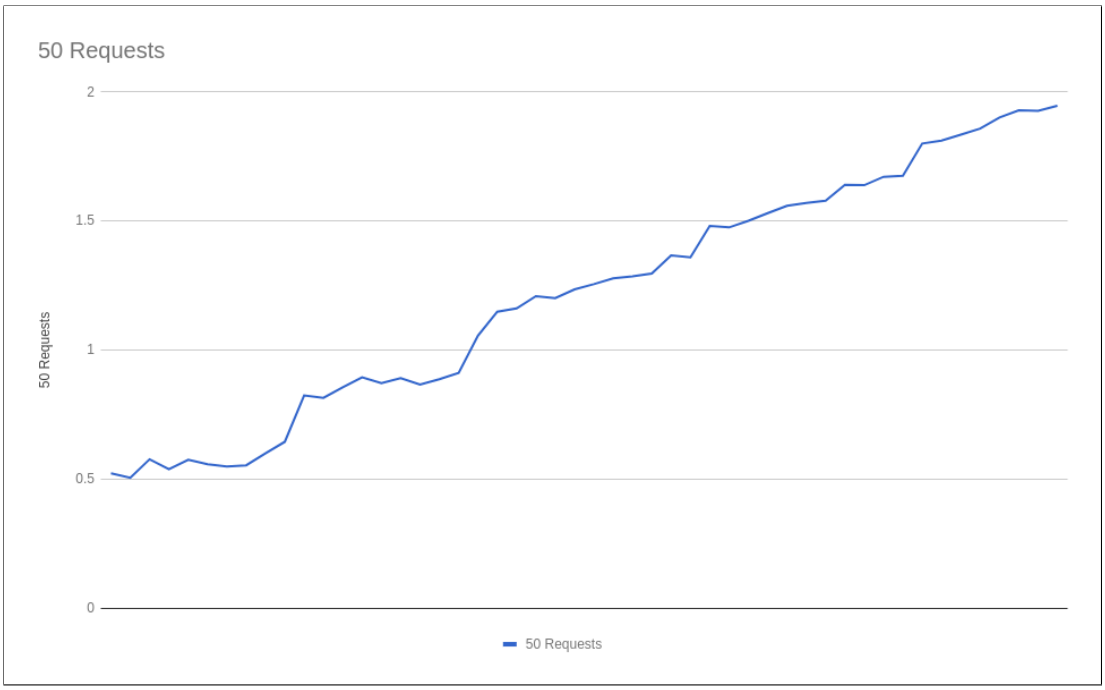


Figure 5.10: Twitter response time (50 requests)

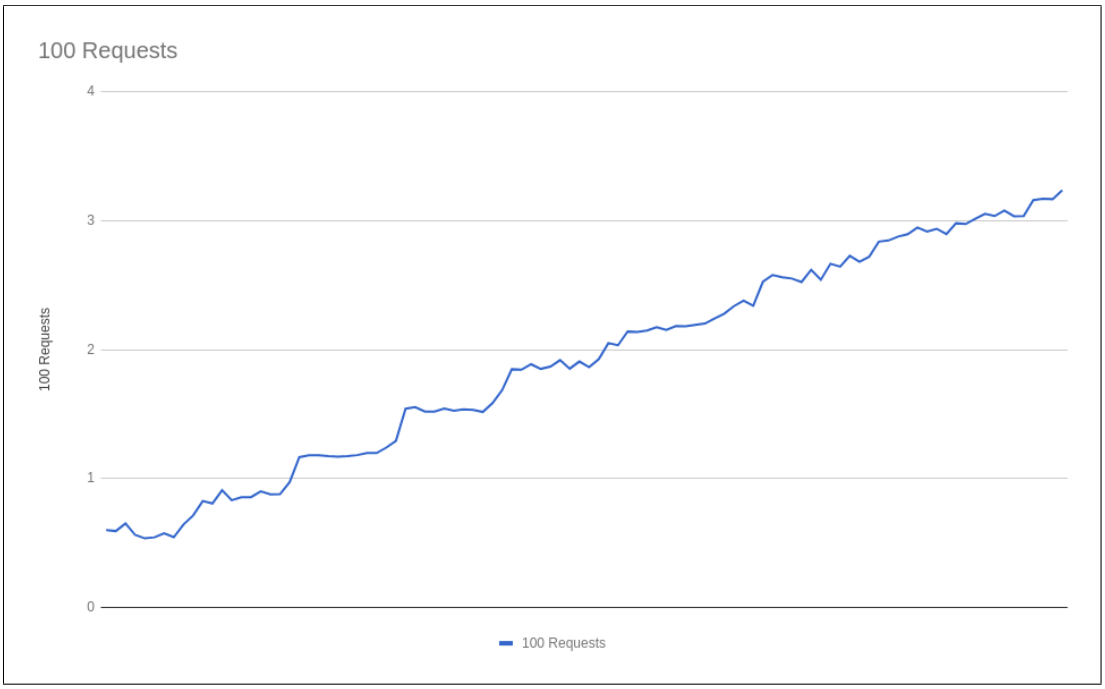


Figure 5.11: Twitter response time (100 requests)

Cloud API serving Twilio SMS notifications

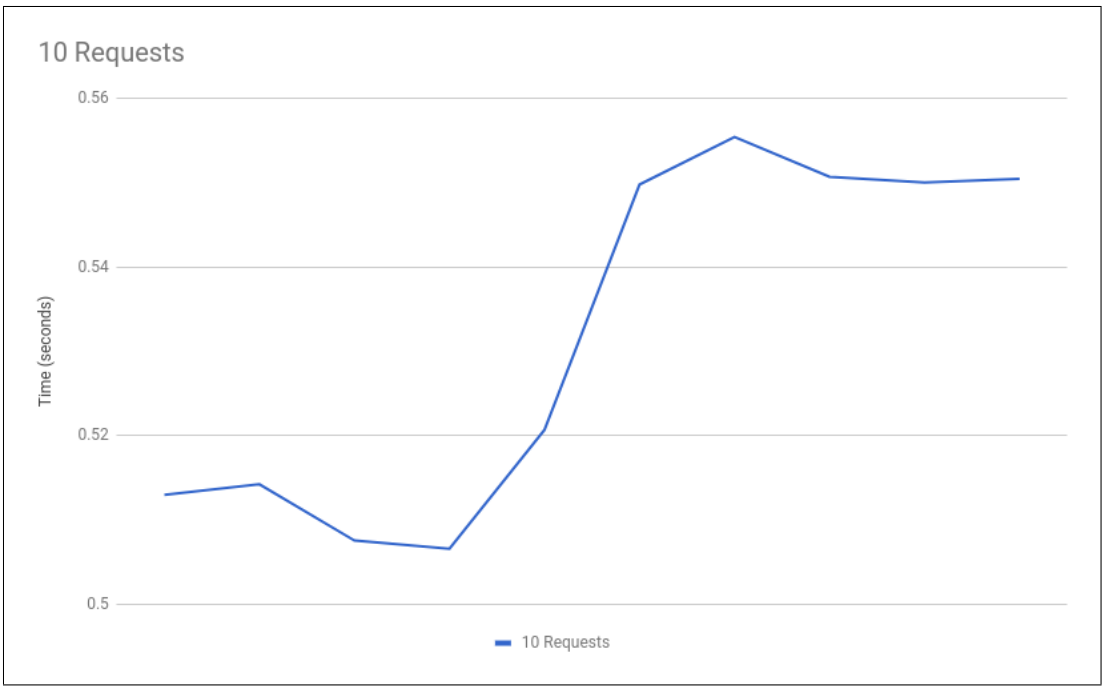


Figure 5.12: Twilio response time (10 requests)

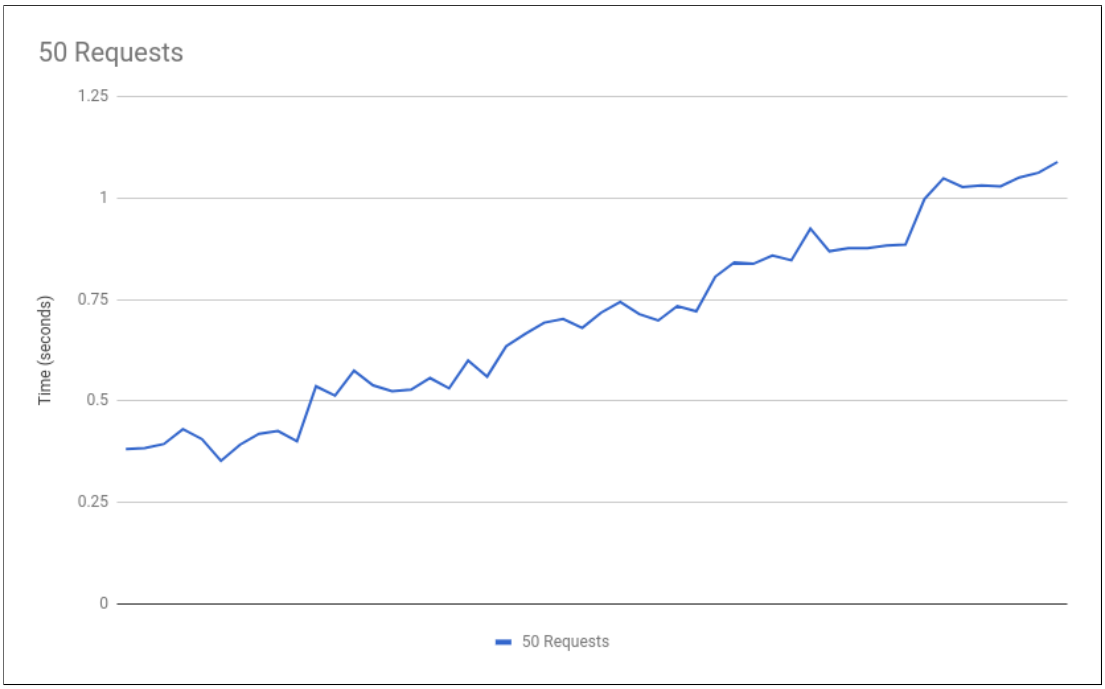


Figure 5.13: Twilio response time (50 requests)

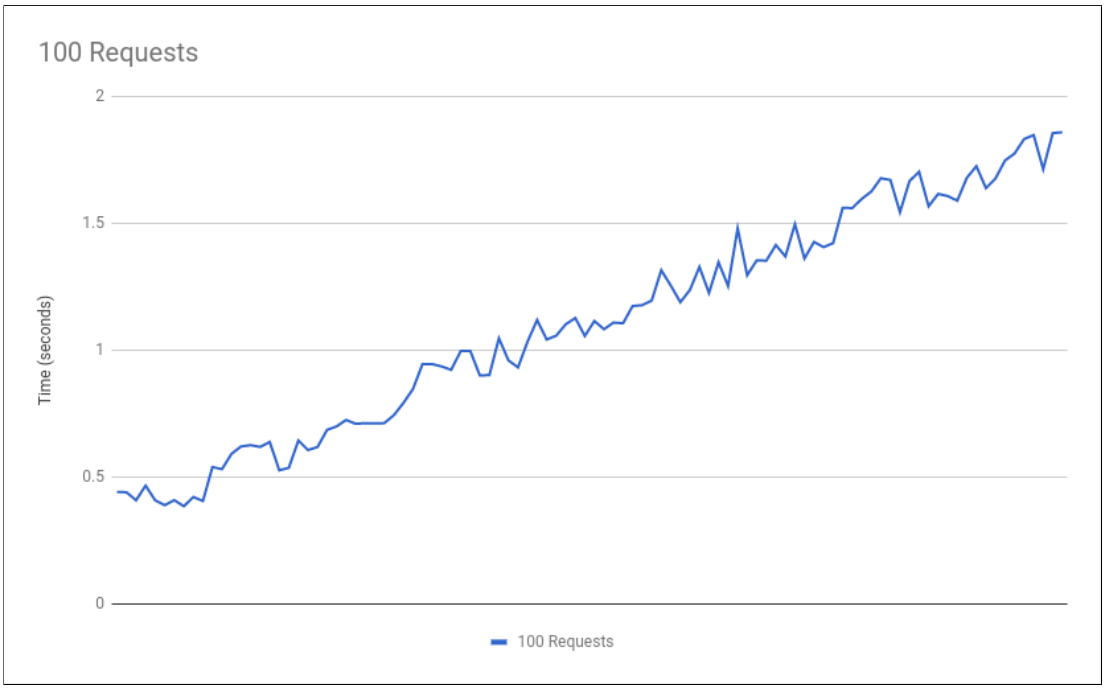


Figure 5.14: Twilio response time (100 requests)

Server resources utilization - The test of Cloud API response speed was also utilised to measure server resources usage. Via CloudWatch, a feature inbuilt in AWS EC2 service, we analysed CPU and Incoming/Outcoming Network operation history. Even when 1000 requests arrived to our Cloud API, the server CPU peaked at only 13.6%, while the average stayed below 2%, showing that our Cloud Instance has a potential to handle larger scale use.

CPU use. X axis - Time (1 dot = 1 Hour), Y axis - CPU Utilization (%)

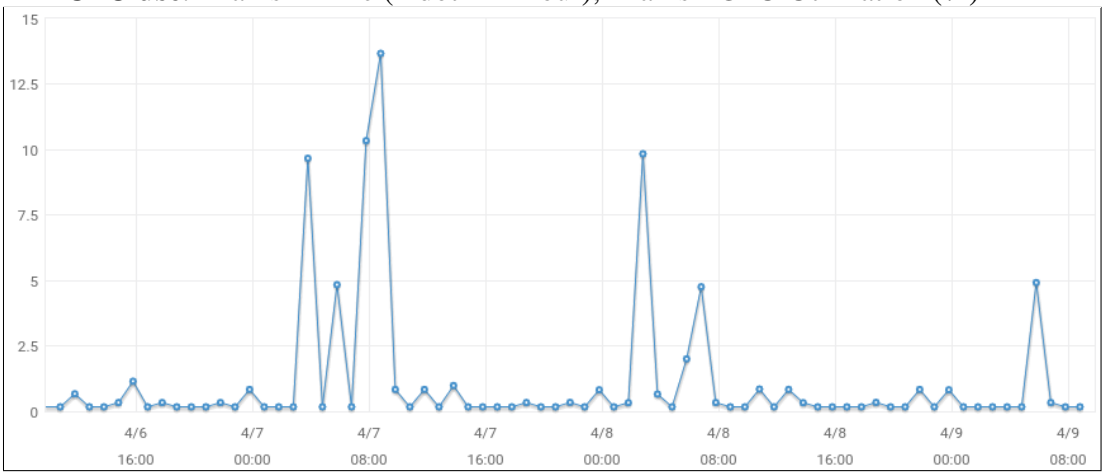


Figure 5.15: Peak CPU utilization

CPU use. X axis - Time (1 dot = 1 Hour), Y axis - CPU Utilization (%)

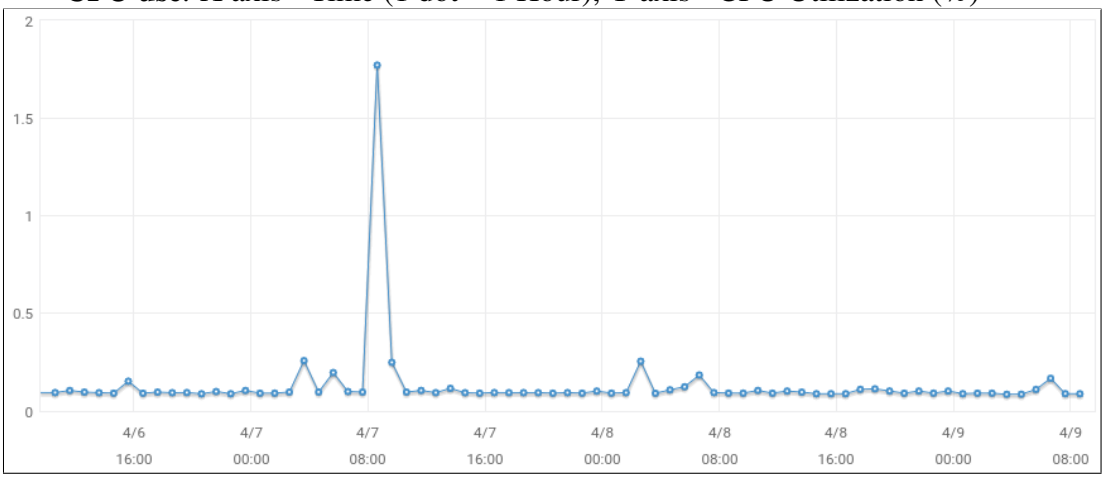


Figure 5.16: Average CPU utilization

Network performance

X axis - Time (1 dot = 1 Hour), Y axis - Incoming Network load (Bytes)

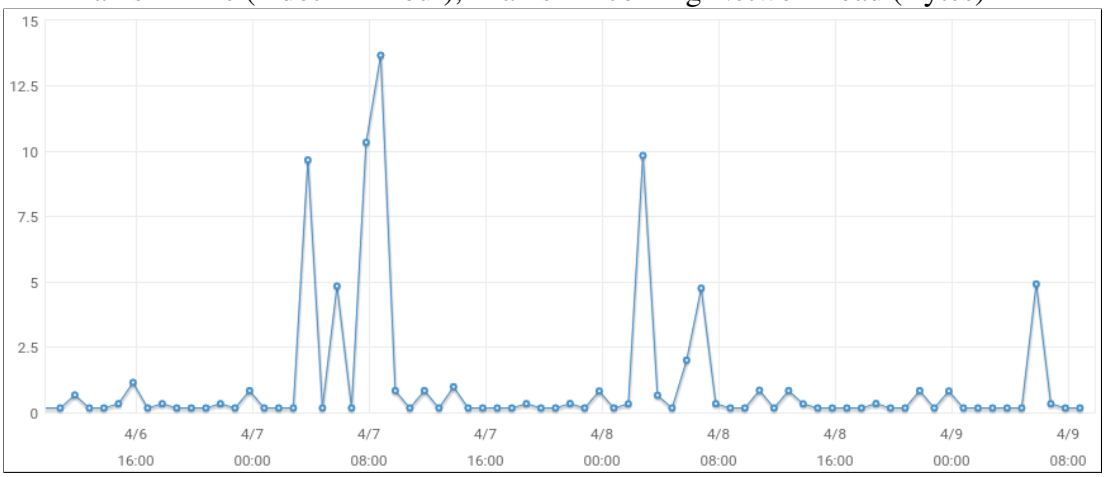


Figure 5.17: Peak incoming Network load

X axis - Time (1 dot = 1 Hour), Y axis - Incoming Network load (Bytes)

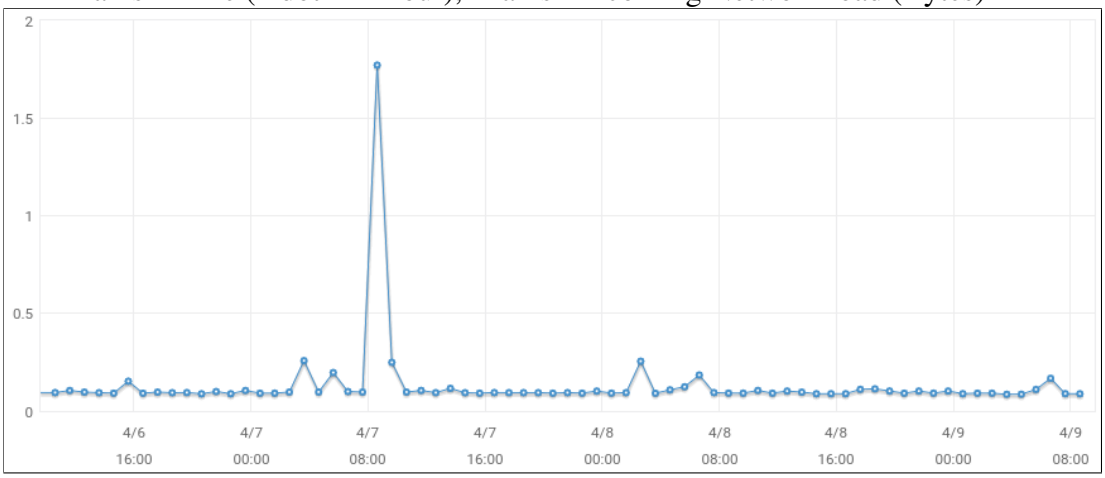


Figure 5.18: Average incoming Network load

X axis - Time (1 dot = 1 Hour), Y axis - Outgoing Network load (Bytes)

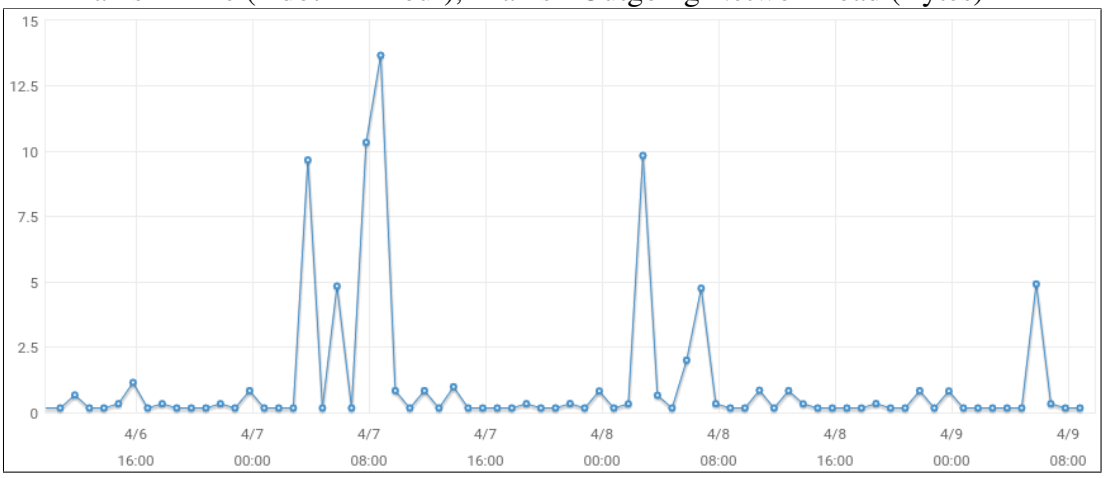


Figure 5.19: Peak outgoing Network load

X axis - Time (1 dot = 1 Hour), Y axis - Outgoing load (Bytes)

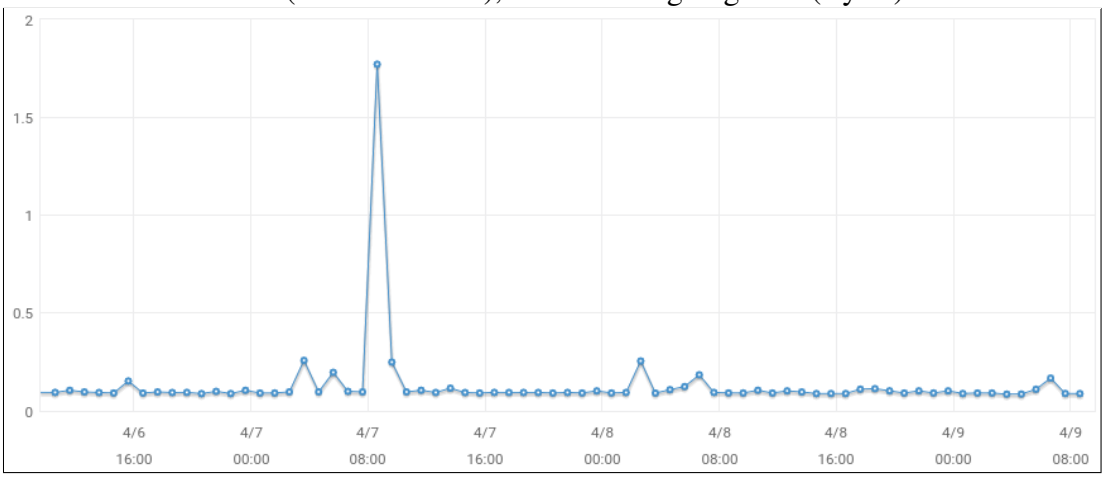


Figure 5.20: Average outgoing Network load

5.2 User and Market Study

From the early stages of development, we were continuously interacting with the target audience of our IoT applications. The participants of our studies provided a necessary critical view, helping to identify the expectations of potential clients and tailor our solutions to match them. We used semi-structured interviews 5.2.2 as a main source of feedback, the Honours Feedback day presentations (5.2.3, 5.2.3) and the surveys helped to set aside irrelevant suggestions and identify features of our IoT device that add the most value to a general public within the School of Informatics.

5.2.1 Material for client studies

For the user evaluation, we had to find a simple approach to introduce our IoT device and the Cloud, without using complex technicalities. As a solution for this problem, we created two scenarios, which explain the potential use cases in an environment familiar to the participant. We also contrast our IoT implementation against current options to show the benefits associated with our product and find what our prototype is lacking as well.

Scenarios:

1. Smart Office device - First setting described a situation when a meeting between a Personal Tutor and a student was organized and then cancelled by the faculty member at the very last minute, due to unforeseen events. Further steps of conversation pit the currently available ways of communication (email, paper sticky notes) against proposed solution (Twitter connected notes, Sending SMS notification to the professor via Twilio). The comparison is used to evaluate if our Smart Office application provides added convenience to the potential users.

2. Event Participation Monitoring - Second situation was used to introduce Event Participation Monitoring. We considered the scenario of a Functional Programming tutorial and compared the current approach of student attendance tracking with the proposed alternative. The existing system relies on tutor bringing a paper sheet to a tutorial and students signing on it with their name, surname and university id. The paper is then used by a tutor to type out the gathered data in an excel spreadsheet and email this to the ITO. The ITO staff then reads the spreadsheet and types the information into the University database.

The proposed solution automates the system by relying on student cards. In this situation, the tutor brings the IoT device we developed to the tutorial and the students tap their cards when walking into the room. Collected data is automatically sent to the university database, avoiding the extra steps of manual work. The ITO can look at the participation data via a Web Interface.

A more in-depth description of these situations is available in the Appendix.

5.2.2 Interviews

Our first step of the IoT system evaluation was a semi-structured interview. We used this study for the Smart Office and the Event Participation Tracking applications throughout the duration of the prototype development process. The interview participants were students, tutors and staff members at the University of Edinburgh. As it can be seen in table 5.26, we made sure that every version was tested on a similarly sized group of people diverse in occupation and gender.

Aims

1. Identify the usability problems and technical flaws of a current IoT system.

2. Gather additional functionality suggestions.

3. Adjust the prototype to better suit the target audience.

4. Keep track of user requirements, ensuring that IoT solution is successful from both technical and potential buyer perspective.

5. For the interviews on final implementation - determine if end result is a success and meets expectations from IT and user point of view.

Method

Participants
Since the study was taking place throughout an extended period of time, we did not specifically split it into sessions but collated the responses according to the version of our system that participant was given to interact with during the interview.

The shortened names for IoT device versions used in the interviews are:

IoT device versions used in the interviews	
V3	The first functional prototype.
V4	The first iteration with a custom built enclosure and SMS notifications functionality.
V5	The final version, with added encryption for data transfer and additional enclosure enhancements (RFID sign near the sensor).

Figure 5.21: Device versions

Figure 5.22: Prototype V3

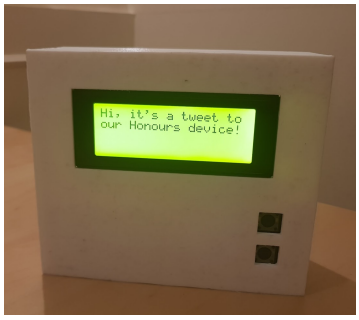


Figure 5.23: Prototype V4

Figure 5.24: Prototype V5

Abbreviations used in participant table	
UG1	1st year undergraduate student.
UG2	2st year undergraduate student.
UG3	3st year undergraduate student.
UG4/MInf	Final(4th) year student / Final (5th) year student on a 5 year MInf study programme.
Other staff	Non-academic people working at School of Informatics (E.g. ITO personnel).

Figure 5.25: Abbreviations

Interviews			
Device Version	**Participant**	**Occupation**	**Gender**
V3	P1	UG1	Male
	P2	UG4	Male
	P3	UG4	Male
	P4	MInf	Female
	P5	Tutor	Female
	P6	Other staff	Male
V4	P7	UG2	Male
	P8	UG4	Female
	P9	UG4	Female
	P10	Other staff	Male
	P11	Other staff	Female
V5	P12	UG1	Male
	P13	Other staff	Male
	P14	Researcher	Male
	P15	MInf	Male
	P16	MInf	Female
	P17	Tutor	Male
	P18	Tutor	Male

Figure 5.26: Participants list

Material
For the interviews, we were iterating though three different versions of our IoT solution, displayed in 5.21. All of these device variants supported two pieces of functionality - the Twitter-enabled sticky notes, and Event participation monitoring. Other features, such as SMS alerts via Twilio and weather forecasts via OpenWeather API, were added on a V4, then carried over to V5. The cloud side was shown to the user as well, by using our database web view on a laptop, a tablet computer or a smartphone. The interview participants were always presented a hardware solution, allowing to have a hands-on experience. To easily explain the possible use case scenarios of a device, we developed two fictional stories, with one centred around Smart Office application, and the other around Event Participation monitoring. The description of these settings is presented in section 5.2.1.

Procedure
We tried to make the interview process as similar to casual conversation as possible. (This meant that our approach to interviewing process differed from the usual practices.)

We did not arrange an interview with a participant in advance and chose to randomly start conversations with people in the Main Library, Appleton Tower, Informatics Forum, Old Medical School or Teviot Row House. The decision was based on the research done Cat Magill, Ewan Klein and Simon Chapple, who recommended a more natural discussion with the participant - "We suggest that environmental monitoring in the workplace can create more value for 'users' and reduce the risk of adverse reactions if employees are more actively involved in the design and communication process."[17].

At the start of the discussion, we stated that we were researching the ways to improve student and staff satisfaction in UoE, with an introduction of Internet-connected devices and then showed them our prototype.

A person was then asked whether he/she had 10-15 minutes free for an interview. If a positive response was received, we presented a consent form to sign and started the discussion on the IoT solution. We did not record the conversations and just noted down the key statements after the interview. The basis for such decision was an intention to make interview experience more natural and less stressful for the participant[17].

Interview structure
With participants consent granted, interview process started by presenting an example situation where our first part of the functionality, a Smart Office, would be applied 5.2.1.

After the introduction, the participant would be given IoT device we developed and asked to perform the tasks mentioned in the Smart Office example scenario:

- Press a button to read a Twitter message left by a Personal Tutor (Without mentioning which particular button)
- Read automatically shown weather forecast (Only on V4 and V5)
- Press a button to send an SMS message to a Personal Tutor (Only on V4 and V5)

During the tasks, or after completion, we would ask the user to comment on any Pros/-Cons they identified in terms of usability, functionality or if they have any further enhancements in mind.

Following the discussion on Smart Office implementation, we go through the same interview process pattern with Event Participation Tracking application. The difference between the two - a different use case setting - Tutorial participants marking 5.2.1 and changed task - Scanning given RFID cards by tapping them on a device (Without mentioning particular location).

The implementation of our IoT system was changing in between the interviews, as we were making progress with the development of our device and our server client while implementing additional improvements suggested by the previous participants of this study.

Results and Discussion
Throughout three stages of our interview experiment, the response to Event Participation Monitoring application was very positive, while the Smart Office Application had to go through several improvements until getting positive feedback.

Interviews with prototype Ver.3
During this first round of interviews, we used a device that did not have any kind of enclosure, with visible wiring and various other parts of the system. Due to this non-user friendly design, our interviewees struggled when choosing a button to press for Twitter feed to be updated. On the other hand, as the RFID scanner had a commonly used symbol5.27, participants had no problems finding where to tap the card.

Throughout the testing of V3, interviewees response to Smart Office implementation was moderate. We received and reacted to the following suggestions to improve on the next version of implementation (All suggestions below are made with a hypothetical scenario in mind.):

Figure 5.27: RFID sign

- **P4** stated that even though a student would be able to read a Twitter message left by the professor, there would still be no way of contacting the professor if he/she has not left a message or does not have good email inbox checking habits.
  - On V4, we added SMS notification functionality to add a two way communication.
- **P2** suggested that when there are no recent tweets, a device could be showing some useful information for the people who are passing by.
  - On V4, a weather forecast feature was introduced, showing the predictions

for the next 3 hours.

- **P1** suggested showing a QR code on the screen, with the contact information of the IoT device owner.
  - We left this feature for future improvements since the current LCD screen is only capable of showing characters, not graphics, and we did not have enough time to work on a more advanced graphical interface of our IoT prototype.

The responses we received while discussing the Event Participation Monitoring feature greatly differed from Smart Office application. Instead of giving feedback for improvement, our interviewees focused more on explaining the flaws of a current method (Paper and Pen). The problems stated below were a sound basis for the usefulness of an IoT system that we propose.

- A tutor, **P5**, mentioned that students who arrive to the tutorial late, start searching for the participation sheet. The shuffling around of participation sheet then distracts other students and the tutor, negatively impacting the tutorial quality in process.
- **P5** also stated that currently every tutor sends participation data to the ITO using different layouts, thus adding to an overall fragmentation of the process. Also, the lack of response from ITO means that a tutor is never sure that data was indeed put to use. As a result, some of the tutors stopped marking participation completely.

Interviews with prototype Ver.4
The second round of interviews was done with more advanced prototype, using initial version of the 3D box we designed. The V4 implementation also gained two features which were responses to issues stated on our first session - Twilio SMS notifications and OpenWeather weather forecasts.

There was a very visible change in style of responses from the interviewees, with more focus on usability. This was mostly due to the shift from a very prototypic look of our device, to a more user friendly enclosure. During the first round, we noticed that people did not put much attention to the layout and structure of the text displayed on the screen, with most attention put to the underlying functionality. An introduction of a more user friendly design, shifted participants focus to usability issues.

- **P7**, **P9** and **P11** pointed out that when a device is waiting for a card to be scanned, it displays a phrase "Tutorial participation monitoring" on a screen. They found such statement unclear and confusing, suggesting to change it with some instructions about the scanning process instead.
  - Reacting to the issue, we changed the text to be "Check yourself! Tap your student card on me." for the V5 prototype.
- **P8**, **P10** and **P11** commented that a box lacks proper indication of the RFID area scanning, making the student and staff cards scanning confusing.

  - To mitigate the problem, a design of a box was adjusted for a V5, with an RFID symbol etched in the area above the RC522 sensor.

The Smart Office implementation was met with positive reactions, especially by the staff, who suggested the SMS notification feature to be useful during the lunch break time. The only proposed change for office application was to use Facebook to send a notification. Facebook would require an Internet connection and a dedicated Facebook account, instead of just a phone number, which is often not feasible from the privacy standpoint.

The feedback on Event Participation Monitoring was overwhelmingly positive. Over this round of interviews, the students **P8** and **P9** mentioned the same issue as a tutor identified in a first round. They stated that students who come in late and start shuffling around the participation sheet, disturb other tutees and negatively impact tutorial performance.

During this interview round, we suggested an improvement to the participation monitoring - Sending an email to PT or ITO, when the system detects a student is constantly skipping tutorials. The response was positive from the staff (who want to ensure better teaching quality), while students had a negative view of the additional layer of control imposed on them.

Interviews with prototype Ver.5
Our third and final iteration of interviews had a predominant focus on usability. The last iteration of the device was already with several improvements added to the design (RFID location symbol, better on-screen instructions for event monitoring task), but we noticed that the more refined version we presented, the more expectations were raised, which enabled us to improve the prototype further. The improvements suggested over this round are set for future development.

- **P12**, **P14** and **P16** suggested adding support for people with disabilities, specifically mentioning a text-to-speech option.
- **P17** and **P18** pointed out that as tutors, they should not be expected to constantly look at the device when a student is coming in. They suggested adding a speaker with several different sound notifications, which would indicate if a student marked tutorial presence successfully or not.
- **P13** mentioned that having a dock with a charger could benefit the ease of use of the implementation.
- **P15** found that a system lacked support for the cases when a student wants to attend a tutorial on a different timeslot than he/she is registered to.

5.2.3 Live presentations

First presentation (6th of March)
The Honours presentation on the 6th of March was meant for gathering feedback from University staff, students and tutors, just like the interviews study. The received responses were then used to evaluate the current state of a device and adjust planned improvements to better suit needs of UoE community. We also checked if findings from the interviews were matching up with opinions of a wider public within the School of Informatics.

Aims

1. Research if the market requirements identified during the interviews, are applicable to wider audience.
2. Assess the usefulness of the device for different types of university personnel.
3. Examine the first impressions of people getting accustomed with the IoT device.

Second presentation (22nd of March)
The second presentation on the 22nd of March presented an opportunity to compare the community responses on two very different versions of our device. The main aim was to find out if the usability and other identified topics of concern on our IoT implementation have changed since the presentation on 6th of March.

Aims

1. Gather the feedback on an improved version of our prototype, with changes made according to expressed concerns on a first presentation.
2. Evaluate the change of potential customers feedback when presented with a device in a more user friendly enclosure.
3. Asses if the requirements and expectations an audience has for our IoT infrastructure were met.

Method

Participants

March 6th Presentation with V3 of IoT device	
Participant Group	**Count**
UG3	2
UG4	2
MInf	2
Tutor	1
Researcher	2
Professor	1
Other staff	2

March 22nd Presentation with V5 of IoT device	
Participant Group	**Count**
UG3	4
UG4	3
MInf	1
Researcher	1
Professor	5
Other staff	1

Material on the 6th March Honours Feedback day
Due to our focus being solely on getting feedback for the current version of the IoT solution (V3), we brought only a device itself and paper forms for anonymous feedback (along with consent forms). The V3 device on March 6th did not include Twilio integration, weather forecast and Web View of the database. These features were either not in place (Web View), or not yet sufficiently robust for a demo (Twilio and Automatic Weather Forecast).

For the presentation of the IoT solution we were using the same two scenarios as and in the later stages of interviews and surveys (Presented in 5.2.1 and 5.2.1).

Procedure on the 6th March Honours Feedback day
The Honours project feedback day was a public event taking place in the Informatics Forum. Its main purpose was to enable final year students to get feedback on their Honours projects. The event was advertised throughout a School of Informatics; thus most of the visitors had an IT background.

In order to attract a high amount of attention to our project, we decided to emphasize our focus on hardware development. To achieve this, we reduced the amount of possible distractions (posters, laptops, etc.) and the only items we had on the table was the V3 prototype along with the paper sheets on which, after discussion, people could write down their positive/negative feedback and suggestions (in case they did not feel comfortable telling them directly).

Hardware projects were unusual to the audience; therefore people who approached our table were quite curious to have a hands-on experience. To the interested visitors, we gave a short introduction about device functionality (card reading for events monitoring, showing latest tweets) and suggested to try out the device.

While the user was interacting with our prototype (or afterwards), we explained the main motivations behind the development of the product, using the two fictions mentioned in a paragraph above. This step was followed by a discussion, answering any

questions that arise, or hearing out what features the person found most useful, which functionality was missing and if there was anything that seemed less relevant.

Results on the 6th of March Honours Feedback day
The first feedback day allowed us to see our IoT implementation from a completely different point of view and change the following stages of device development due to the feedback received on the 6th of March presentation.

The people who approached our stand were mainly from computer security background. They were impressed with Event Participation Monitoring, Smart Office applications, and expressed interest in seeing additional functionality, such as a demo ITO view of a system (which we did not have at the time). At the same time, concerns about security and privacy were mentioned, and a large number of questions regarding security were asked.

The key questions were:

- What private information is accessed from a staff/student card?
- How secure is communication between an IoT device and a Cloud API? Is it encrypted?
- Is it possible to connect private Twitter account?
- How secure is the server itself?
- Is the Cloud API using HTTPS to access other API's (Twitter, Twilio, OpenWeather, ArrestDB)?

Thus at the end, even though our device offered convenient solutions, the researchers, professors and other more privacy-conscious people explicitly stated security as the main reason why they would not use the device.

Material on March 22nd Honours Feedback day
Our second presentation was used to show the final version of an IoT implementation (V5), along with the whole development story. We created a poster, which showed a summary of our work 11, brought all 5 versions of our prototype 5.28 and used a laptop to show a database structure on a Web View 4.7 4.8 4.9.

Procedure on March 22th Honours Feedback day
The presentations on 22nd of March were identical to the previous feedback event on 6th of March. The main difference was that the number of visitors was a lot higher and our final prototype was ready.

As our implementation was already finalized, we decided to focus on the presentation of the whole system, instead of showing only the part that user interacts with.

We presented our final iteration along the previous versions to explain the development process, and used poster along with the database web view to tell how the Cloud enabled a whole process. In addition, the two scenarios were used once again, to present possible use cases. We wrote down all the gathered feedback after the event, while several professors also sent us their evaluations by email.

Figure 5.28: Presentation

Results on March 22nd and Discussion

The participation at Honours Feedback days were the most beneficial evaluation steps taken during the project. For the 22nd of March event, we mitigated the security problem. The V5, which we brought to the presentation, had an encrypted communication with the Cloud, and the Cloud was communicating with other APIs through HTTPS. Along with these changes, we added support for Twitter private accounts and a Web View for the database.

As security was not an issue anymore, most of the visitors were interested in a cost of making the device (Around £30-35), and running the Cloud itself (Around £8-10 per month), along with expected savings (Around £800 per semester for a single course such as Functional Programming).

We presented a live demo of out Web Interface to the ITO. The response was very positive, with several members saying that they hope to see our project implemented on a large scale within the School of Informatics. Since we solved the security issues and added new features to the prototype (enclosure, SMS notifications, weather forecast), professors and researchers now said it was "extremely likely" that they would use this improved version of our device. They found newly added SMS notification to be "extremely convenient" feature, and an improvement was immediately suggested - adding a card scanning step before sending the SMS. The scanning of a card would then allow modifying the SMS, to send individual contact details to an owner of the IoT device.

Other ideas included:

- SMS spamming protection by detecting multiple clicks. We tested the function and found out that it is already handled by the Twilio API itself.
- Device showing note left on Twitter only after visitor is authenticated via the RFID card

Overall, we were pleased with results of Honours Feedback presentations, as they allowed us to thoroughly improve the device and get positive feedback from all concerned parties within the School of Informatics. With several staff members already showing interest in acquiring the devices for personal use, we felt that both technical and market evaluations for our Smart Office device are worth to be called a success.

Achievements

The CISA[18] prize - for the best project and presentation at the Honours Project Feedback event, we won a CISA[18] award (Centre for Intelligent Systems and their Applications).

Other presentations - we had a chance to present our IoT solution to representatives of large organisations. ARM Holdings[36] and EPSRC (Engineering and Physical Sciences Research Council)[25] gave positive responses to the presentations, encouraging further development that would result in a design ready for wider scale application.

5.2.4 Surveys

At the end of our project, an online survey on the Event Participation Tracking application was issued to the UoE students and tutors. For this evaluation, we used a final iteration of the IoT project. Since we considered our IoT solution a technological success, the main goal of the survey was to determine if a business/client audience, satisfied with the end product as well.

The Smart Office application was not included in this part of the evaluation, due to the fact that an another more lengthy use case would be required. The result of such addition would increase the time necessary to finish a questionnaire, meaning that fewer people will have a long-enough attention span to finish the whole survey. Also, the Smart Office implementation already had a sizable amount of responses from March 22nd project feedback day.

Aims

1. Find out how proposed IoT solution compares to current infrastructure.
2. Evaluate the ease of use of a final IoT implementation.
3. Identify useful future extensions to the project.

Method

Participants

We managed to gather a sizable number of responses, with a total of 49 students and tutors. The student responses account for around 80% and tutors - for approximately 20%. The students' survey has a disproportional number of UG3 and UG4 students. We presume that it is because students from later years have a better understanding of the current issues in the School of Informatics, thus becoming more motivated to express their opinion on a study aiming to improve the conditions for students and staff.

The number of tutors is spread evenly between UG4 and Masters, as it is a common practice to tutor after first developing an extensive knowledge of the taught subject. There was one UG3 participant, thus enabling us to slightly broaden a covered spectrum of backgrounds.

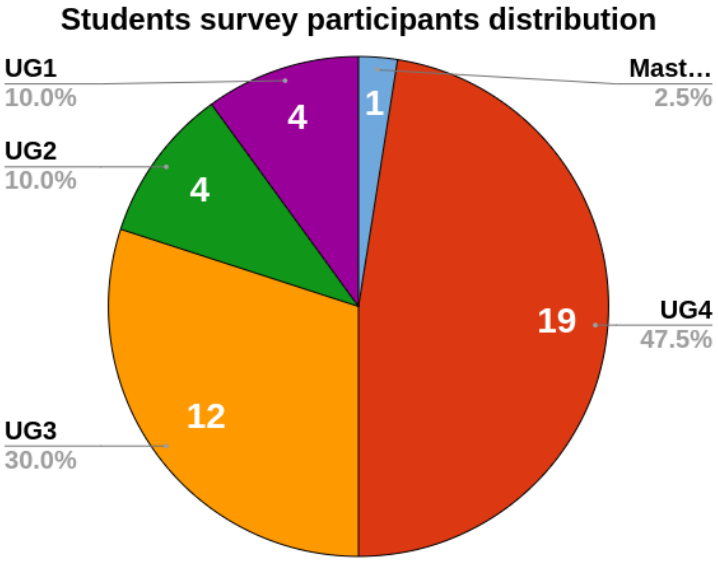


Figure 5.29: Students survey participants distribution

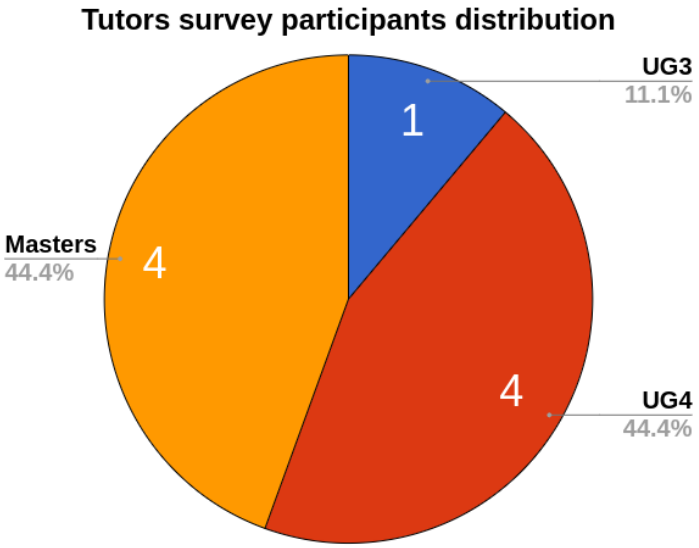


Figure 5.30: Tutors survey participants distribution

Material
The survey was created on a Google Forms service, with a focus on Event Participation Tracking functionality. A survey form contains an explanation of current and proposed solutions. They are presented in text and with photos(for a visual explanation), as scenarios showing the key differences between paper and pen method and the use of automated IoT system. Depending on the survey, questions were tailored to better represent concerns of students or tutors. Both versions of our survey are available to see in the Appendix.

Procedure
The survey was sent out to people in the School of Informatics Facebook group (2186 members) and given to people in printed out a form at the University. The survey was not mandatory to any of the participants.

Results and Discussion
Overall, the survey has shown that both students and tutors are welcoming the idea of tutorial participation monitoring via an RFID-enabled IoT device. An average student goes to around 2 tutorials(1.975) per week 5.31, while an average tutor has 2.2 tutorials per week 5.32, meaning that our system would be used regularly, enabling the increases in students/tutors satisfaction and fast pay off of the new IoT application (due to cost savings).

Most of the participants showed a high level of dissatisfaction with current paper and pen method (more than 50% of the students5.33, and 80% of the tutors5.34, stated it is "Not at all convenient" or "Not convenient"). On the follow up optional question asking to state issues of current method, the main arguments were similar to responses from interviews and presentations: shuffling of a paper sheet distracts students, students forgetting to mark presence if they come late and it , in general, being a time consuming process.

There were no tutors supporting a current approach, while the 5 students who found current method "Convenient" or "Extremely convenient" 5.33, were stating that they usually are attending tutorials randomly, ignoring the allocated timeslot. The check-in at the different tutorial is currently not available on our device, and based on responses, we found the lack of it to be the main reason of negativity to a new system, and positive view to the current application.

The answers to question asking to rate a convenience of a proposed IoT solution, had a very positive response, with 34/40 students 5.35 and 4/5 Tutors marking it as either "Convenient" or "Extremely convenient" 5.36. The 4 negative student after analysis of answers to other questions, were tied down to the same reasons as in a paragraph above - these students preferred disorganized tutorial participation schedule, which we do not have support for yet. A single negative tutor response was motivated by the personal trait of an individual, stating that "Marking the student attendance by myself allows to better remember the names of my tutees".

Our work put into making a device more user-friendly paid off, as the question evaluating the usability of the device received 35/40 positive answers from students 5.37, and answers of all 9 tutors were positive as well 5.38.

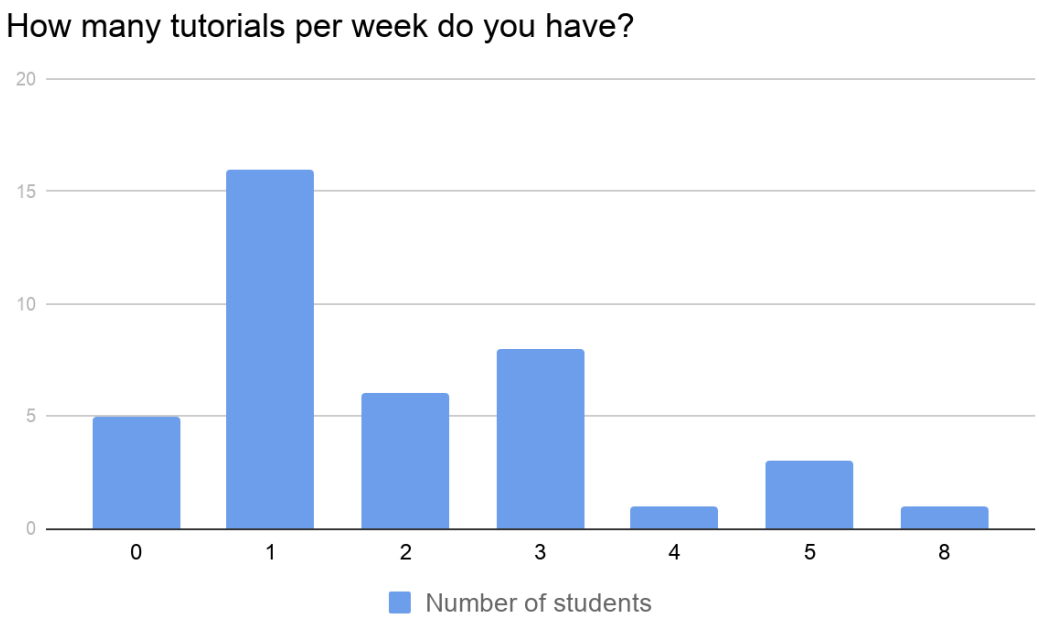


Figure 5.31: Students weekly tutorials count

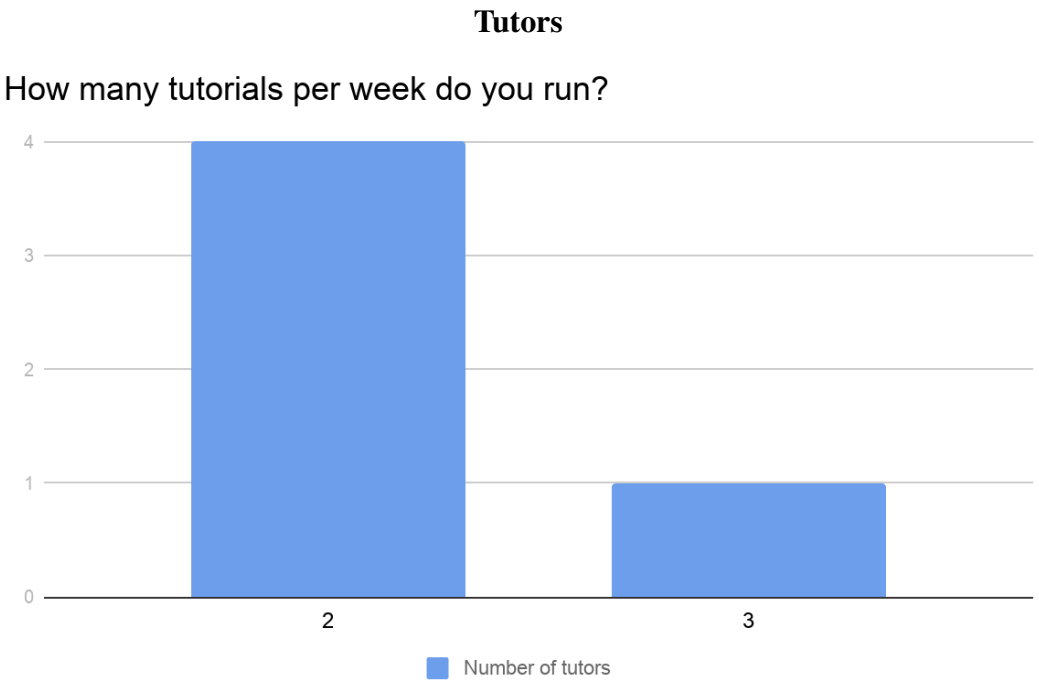


Figure 5.32: Tutors weekly tutorials count

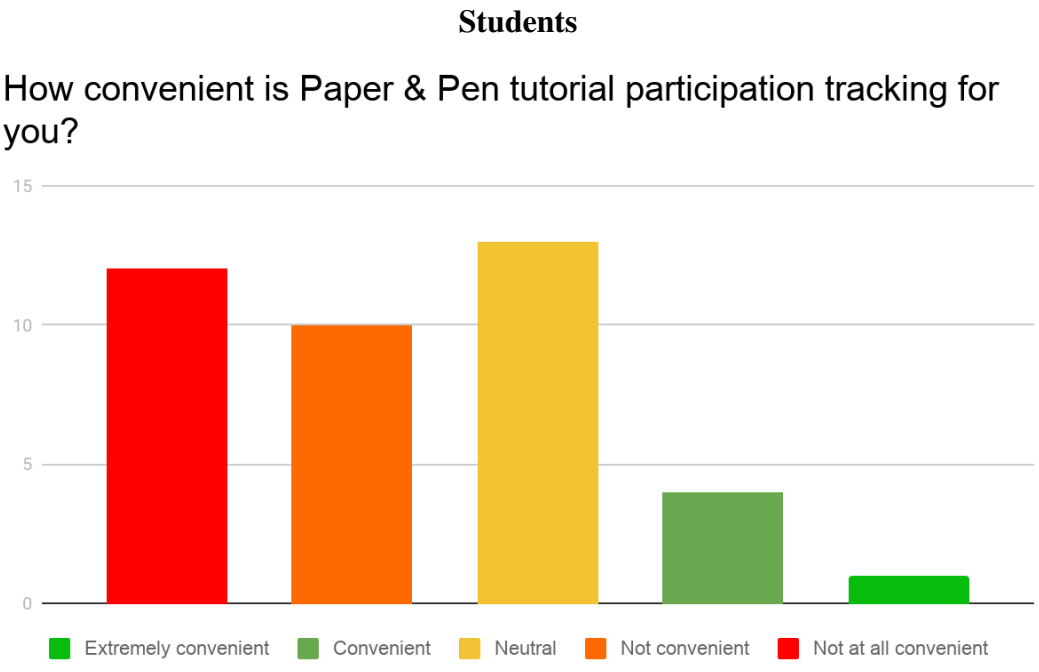


Figure 5.33: Students convenience rating of current tutorial participation tracking method

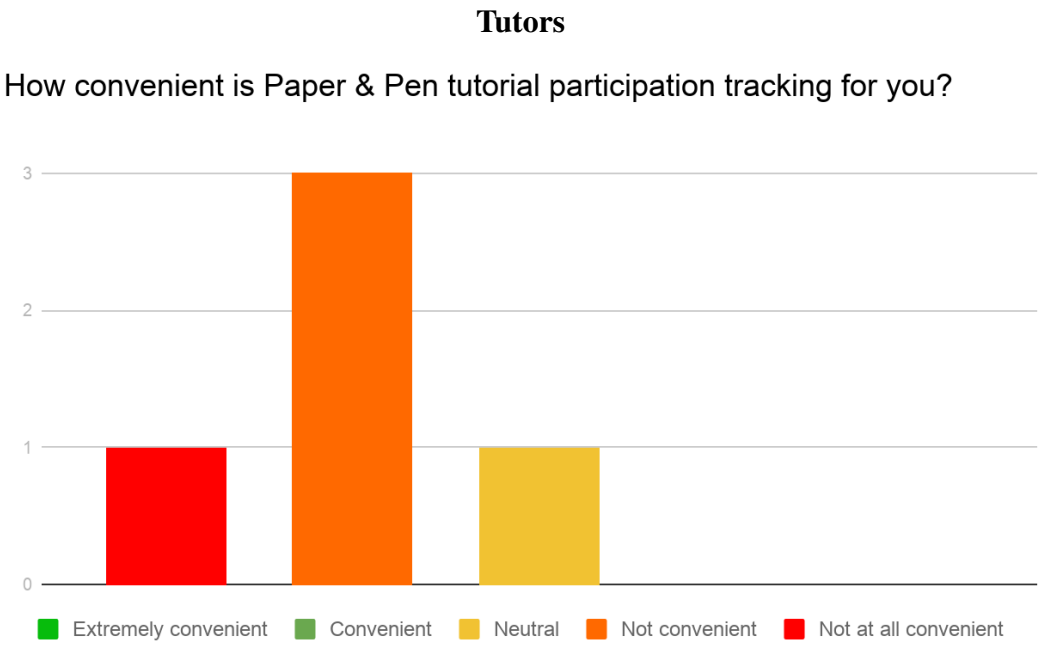


Figure 5.34: Tutors convenience rating of current tutorial participation tracking method

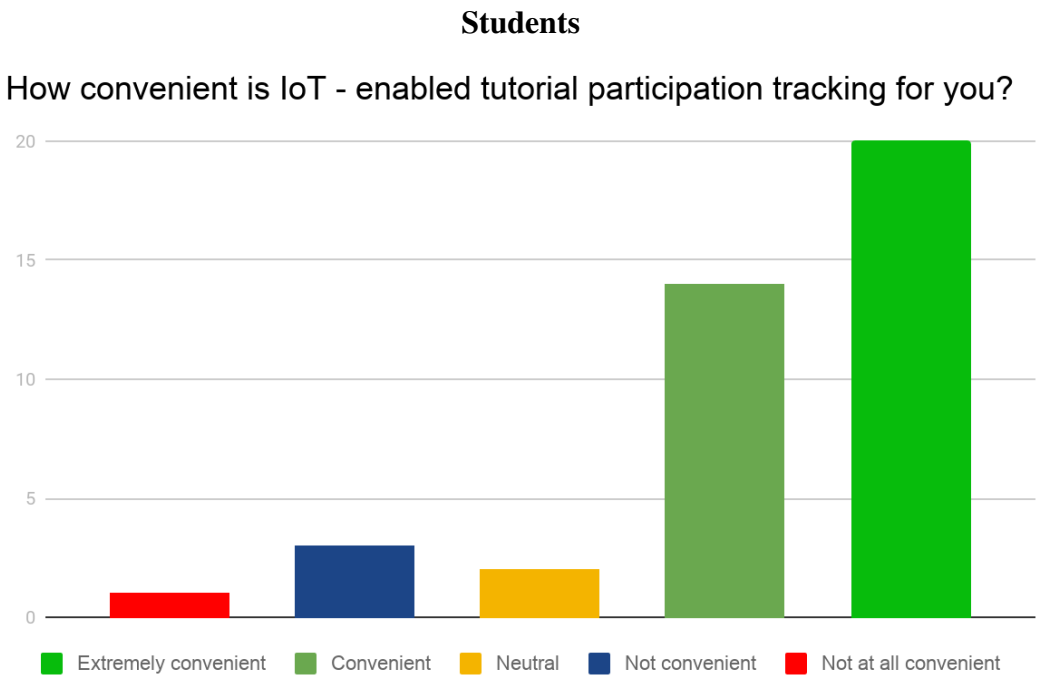


Figure 5.35: Students convenience rating of proposed IoT alternative

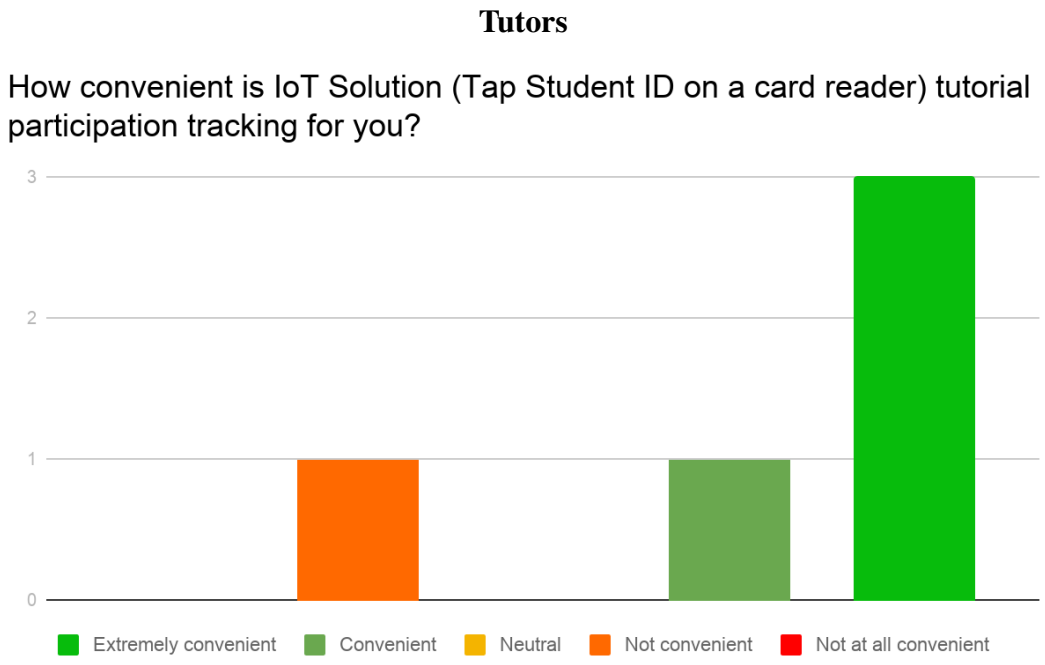


Figure 5.36: Tutors convenience rating of proposed IoT alternative

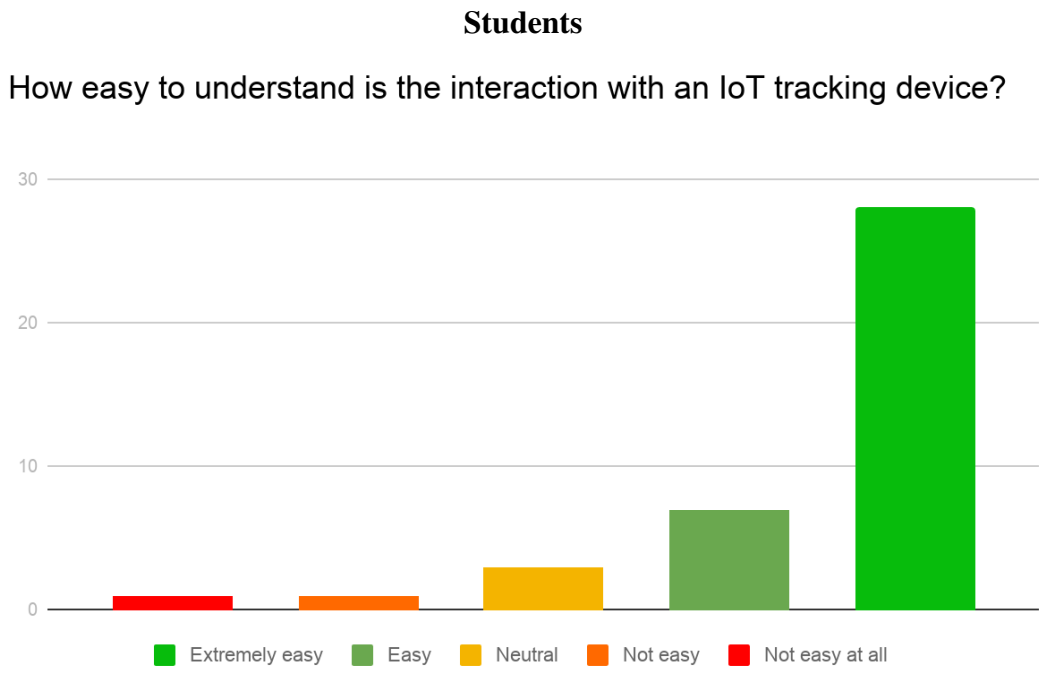


Figure 5.37: Students rating on device usability

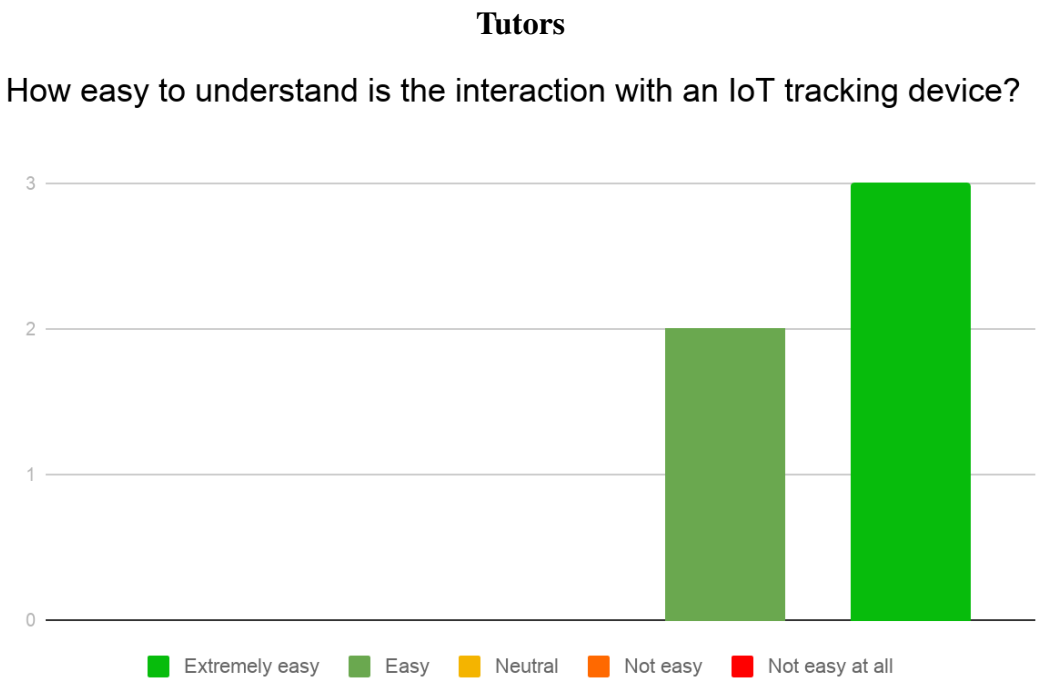


Figure 5.38: Tutors rating on device usability

Chapter 6

Future work and Conclusions

Conclusions

We have developed an end-to-end IoT system, which consists of the cloud logic functioning as a centre hub and the embedded devices that the users interact with. Our IoT solution improves the enterprise offices by providing an additional layer of communication via a Smart Office solution and benefits the university environment by automating inefficient and costly processes via an Event Participation Monitoring system.

By building an IoT infrastructure from the ground up, we identified the key issues that developers have to overcome in a new and highly unregulated world of Internet-connected solutions. We shown the whole prototyping process and documented a complicated path to the end result, requiring knowledge in several disciplines, ranging from embedded systems, cloud services and databases, to human-computer interaction or electrical engineering.

The success of our IoT prototype solution, along with positive feedback from user studies participants proves that a balance of effort put into hardware, software and target market research, results in decreased development cost and a final implementation that satisfies requirements of potential buyers.

Future work

We are aware that there still is a lot of room for improvement. Due to the time spent on solving hardware compatibility problems, we could not focus on the Cloud software as much as we would have liked. On the other hand, our struggle is a real life example of the problems that a fragmented IoT market faces.

For the future work, our first steps would be focused on finding and implementing an alternative encryption library solution. Only after this main concern of the potential customers is solved, we would start, working on other possible additions which the participants of our evaluation studies presented. These include:

1. A support for a tutor to mark a student present even if the person is allocated to different group.
2. Email reminders about an upcoming tutorial
3. Providing alternative hardware feedback, such as presenting messages both on a display, and via an inbuilt speaker, to better support people with disabilities

Additionally, a larger scale development could be done, by integrating the existing IoT solution into the University of Edinburgh system and starting a pilot study with Smart Office and Event Participation Monitoring applications. This step would also require not only a better security but and a well polished hardware implementation, tailoring the decive specifically for the two potential use cases. Furthermore, an even more low-level solutions of power management could be explored.

Bibliography

[1] Adafruit. Rfid selection guide. https://cdn-shop.adafruit.com/datasheets/rfid+guide.pdf, accessed 20 Nov 2017.

[2] Zigbee alliance. zigbee. http://www.zigbee.org/, accessed 13 Nov 2018.

[3] Amazon. Anker powerbank. https://www.amazon.co.uk/Anker-PowerCore-Ultra-Compact-Fast-Charging-Technology/dp/B01CU1EC6Y/, accessed 10 Feb 2018.

[4] Amazon. Amazon ec2. https://aws.amazon.com/ec2/, accessed 10 Jan 2018.

[5] Amazon. Amazon s3. https://aws.amazon.com/s3/, accessed 10 Jan 2018.

[6] Amazon. Amazon web services. https://aws.amazon.com/, accessed 10 Jan 2018.

[7] Amazon. Gaoxing qc2004a with lcd2004 chip. https://www.amazon.co.uk/Gaoxing-Tech-Serial-Module-Arduino/dp/B01N358KJN/, accessed 20 Nov 2017.

[8] Amazon. Makerhawk 0.91 color lcd. https://www.amazon.co.uk/MakerHawk-Display-SSD1306-3-3V-5V-Arduino/dp/B076BJZ42H/, accessed 20 Nov 2017.

[9] Amazon. Waveshare 2.9 inch e-ink display. https://www.amazon.co.uk/dp/B0751JYW36/ref=psdc_430497031_t1_B071JFRV2S, accessed 20 Nov 2017.

[10] Inc. Amazon.com. Amazon echo (1st gen). https://www.amazon.com/Amazon-SK705DI-Echo/dp/B00X4WHP5E, accessed 14 Nov 2017.

[11] apple. Apple pay. https://www.google.co.uk/search?q=apple+pay&oq=apple+pay&aqs=chrome..69i57j0l5.4601j0j4&sourceid=chrome&ie=UTF-8, accessed 22 Mar 2018.

[12] Arduino. Simple arduino uno - esp 8266 integration. https://create.arduino.cc/projecthub/circuito-io-team/simple-arduino-uno-esp-8266-integration-dba10b, accessed 10 Jan 2018.

[13] Atmel. Atmega328/p datasheet complete. http://ww1.microchip.com/downloads/en/DeviceDoc/

Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf, accessed 13 Oct 2017.

[14] Alix Axel. Restful php api for sqlite, mysql and postgresql databases. https://github.com/alixaxel/ArrestDB, accessed 19 Nov 2018.

[15] Omer Siar Baysal. Rfid522 door unlock. https://github.com/omersiar/RFID522-Door-Unlock/blob/master/LCD/LCD.ino, accessed 20 Sep 2017.

[16] Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full aes-192 and aes-256. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, pages 1–18, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[17] Ewan Klein Cat Magill and Simon Chapple. I am not a number: Towards participatory IoT monitoring in the workplace. pages 1–9, 2018.

[18] CISA. Centre for intelligent systems and their applications. http://web.inf.ed.ac.uk/cisa, accessed 7 Apr 2018.

[19] Wikipedia contributors. ipad (1st generation). https://en.wikipedia.org/wiki/IPad_(1st_generation), accessed 3 Feb 2018.

[20] Wikipedia contributors. iphone (1st generation). https://en.wikipedia.org/wiki/IPhone_(1st_generation), accessed 3 Feb 2018.

[21] Wikipedia contributors. Block cipher mode of operation. https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#/, accessed 8 Feb 2018.

[22] Terry Dunlap. The 5 worst examples of iot hacking and vulnerabilities in recorded history. https://www.iotforall.com/5-worst-iot-hacking-vulnerabilities/, accessed 18 Mar 2018.

[23] Diana Eftaiha. An introduction to apache. https://code.tutsplus.com/tutorials/an-introduction-to-apache--net-25786, accessed 19 Nov 2018.

[24] Let's Encrypt. Getting started. https://letsencrypt.org/getting-started/, accessed 12 Mar 2018.

[25] Engineering and Physical Sciences Research Council. Epsrc description. https://medium.com/iotforall/how-to-be-strategic-about-building-your-iot-prototype-c8171936ec53, accessed 1 Mar 2018.

[26] ESP8266.com. Esp8266 developers forum. http://www.esp8266.com/, accessed 15 Jan 2018.

[27] Espressif. Esp8266 low power solutions. https://www.espressif.com/sites/default/files/9b-esp8266-low_power_solutions_en_0.pdf, accessed 13 Jan 2018.

[28] fda. U.s. food & drug administration. https://www.fda.gov/, accessed 18 Feb 2018.

[29] Fitbit. Fitbit. https://www.fitbit.com/uk/home, accessed 2 Dec 2018.

[30] Google. Google cloud. https://cloud.google.com/, accessed 10 Jan 2018.

[31] Google. Google home. https://store.google.com/gb/product/google_home, accessed 2 Apr 2018.

[32] google. Google pay. https://pay.google.com/about/, accessed 22 Mar 2018.

[33] Google. Google iot core. https://cloud.google.com/iot-core/, accessed 3 Mar 2018.

[34] Gspin. Esp8266 light sleep. https://community.blynk.cc/t/esp8266-light-sleep/13584, accessed 17 Feb 2018.

[35] Erico Guizzo. Send a tweet to your office door. https://spectrum.ieee.org/geek-life/hands-on/send-a-tweet-to-your-office-door, accessed 20 Sep 2017.

[36] Arm Holdings. Arm description. https://medium.com/iotforall/how-to-be-strategic-about-building-your-iot-prototype-c8171936ec53, accessed 3 Apr 2018.

[37] StackOverflow insights. Most popular databases technologies. https://insights.stackoverflow.com/survey/2018/#technology-databases, accessed 9 Jan 2018.

[38] P. Patras T. Spink J. Classen, D. Wegemer and M. Hollick. Anatomy of a Vulnerable Fitness Tracking System: Dissecting the Fitbit Cloud, App, and Firmware. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies '18*, pages 42:2–42:3, 2016.

[39] Andrei Klubnikin. Iot development challenges: Why do iot projects fail? https://r-stylelab.com/company/blog/iot/iot-development-challenges-why-do-iot-projects-fail, accessed 17 Oct 2018.

[40] Nest Labs. Nest learning thermostat. https://nest.com/thermostats/nest-learning-thermostat/overview/, accessed 18 Mar 2018.

[41] Erhana Lankus. Arduino latest christmas tweet reader. https://github.com/erhanalankus/Arduino-Latest-Christmas-Tweet-Reader, accessed 20 Nov 2017.

[42] Lantronix. Component lifespan considerations for industrial iot oems. https://www.lantronix.com/blog/component-lifespan-considerations-for-industrial-iot-oems/, accessed 15 Feb 2018.

[43] James Lewis. Arduino to esp8266, 5 reasons to switch. https://www.baldengineer.com/esp8266-5-reasons-to-use-one.html, accessed 15 Jan 2018.

[44] WORLD MEDIA ONLINE LTD. Iot world forum 2017. http://iotinternetofthingsconference.com/, accessed 19 Oct 2018.

[45] Rob Matheson. Moneyball for business. http://news.mit.edu/2014/behavioral-analytics-moneyball-for-business-1114, accessed 14 Nov 2017.

[46] Lee Mathews. Hackers lock down hotel rooms in a new twist on ransom attacks. https://www.forbes.com/sites/leemathews/2017/01/30/hackers-lock-down-hotel-rooms-in-a-new-twist-on-ransom-attacks/, accessed 18 Mar 2018.

[47] mbed. Ble modes and profiles. https://docs.mbed.com/docs/ble-intros/en/latest/Introduction/BLEInDepth/, accessed 13 Nov 2018.

[48] Robert McCool. Apache http server project. https://httpd.apache.org/, accessed 19 Nov 2018.

[49] Christina Mercer and Hannah Williams. Internet of things examples: Best uses of iot in the enterprise. https://www.computerworlduk.com/galleries/cloud-computing/internet-of-things-best-business-enterprise-offerings-3626973/#7, accessed 15 Feb 2018.

[50] Microsoft. Microsoft azure. http://azure.microsoft.com/, accessed 10 Jan 2018.

[51] Julie Mullins. Tips for selecting an iot platform. https://www.zentri.com/tips-for-selecting-an-iot-platform/, accessed 14 Nov 2017.

[52] Ayla Networks. How to select the right iot platform. https://www.mouser.com/pdfdocs/SelectingtheRightIoTPlatformAylaWhitePaper.PDF, accessed 14 Nov 2017.

[53] University of Edinburgh. Experimenting with iot. http://iot.ed.ac.uk/projects/, accessed 3 Apr 2018.

[54] CSO Online. 465,000 abbott pacemakers vulnerable to hacking, need a firmware fix. https://www.iotforall.com/5-worst-iot-hacking-vulnerabilities/, accessed 18 Mar 2018.

[55] openweathermap. Weather api. https://openweathermap.org/api, accessed 12 Feb 2018.

[56] Finn Pierson. Why cloud computing is so popular and how it transforms business. https://www.espressif.com/sites/default/files/9b-esp8266-low_power_solutions_en_0.pdf, accessed 17 Mar 2018.

[57] Daniel Price. How to be strategic about building your iot prototype. https://medium.com/iotforall/how-to-be-strategic-about-building-your-iot-prototype-c8171936ec53, accessed 10 Sep 2017.

[58] RasPi.TV. Raspberry pi zero - power measurements. http://raspi.tv/2015/raspberry-pi-zero-power-measurements, accessed 13 Oct 2017.

[59] ESP8266 Github repository contributors. Discussion on tls support issues. https://github.com/esp8266/Arduino/issues/2733, accessed 8 Feb 2018.

[60] PUC RIO. Lua. http://www.zigbee.org/, accessed 6 Jan 2018.

[61] semtech. Lora technology. https://www.semtech.com/technology/lora, accessed 10 Jan 2018.

[62] sjm. Enisa. https://www.sjm.com/professionals/resources-and-reimbursement/technical-resources/cardiac-rhythm-management/connectivity-and-remote-care/remote-care/merlinathome-wireless-transmitter-model-ex1150?halert=show&clset=af584191-45c9-4201-8740-5409f4cf8bdd%3ab20716c1-c2a6-4e4c-844b-d0dd6899eb3a, accessed 11 Apr 2018.

[63] SQLite. Sqlite. https://www.sqlite.org/index.html, accessed 9 Jan 2018.

[64] Statista. Market share of wearables unit shipments worldwide by vendor from 1q'14 to 4q'17. https://www.statista.com/statistics/435944/quarterly-wearables-shipments-worldwide-market-share-by-vendor/, accessed 14 Nov 2017.

[65] statista. Do you personally use a smartphone?* - by age. https://www.statista.com/statistics/300402/smartphone-usage-in-the-uk-by-age/, accessed 3 Feb 2018.

[66] EverSQL Team. Most popular databases in 2017 according to stackoverflow survey. https://www.eversql.com/most-popular-databases-in-2017-according-to-stackoverflow-survey/, accessed 9 Jan 2018.

[67] James Thrasher. Rfid vs. nfc: What's the difference? https://blog.atlasrfidstore.com/rfid-vs-nfc, accessed 20 Nov 2017.

[68] Twilio. Twilio docs. https://www.twilio.com/docs/, accessed 10 Mar 2018.

[69] Twitter. Get tweet timelines. https://developer.twitter.com/en/docs/tweets/timelines/overview, accessed 22 Sep 2017.

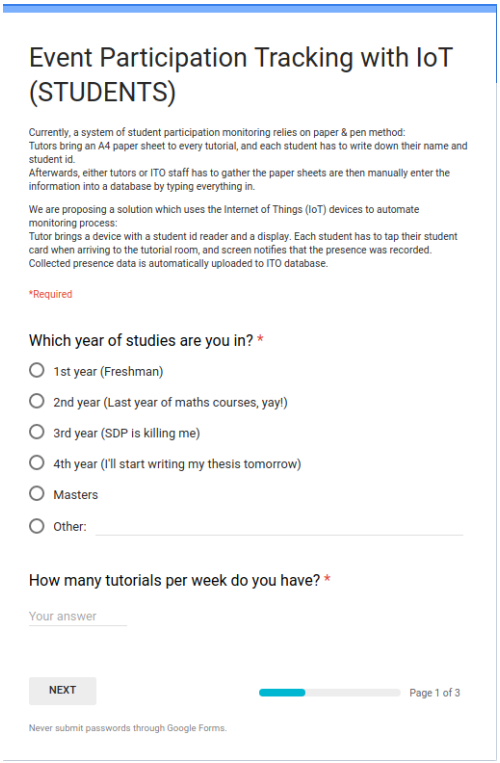
[70] Evan Vosberg. Javascript library of crypto standards. https://www.npmjs.com/package/crypto-js, accessed 20 Mar 2018.

[71] Stephen Watts. Saas vs paas vs iaas: What's the difference and how to choose. https://www.bmc.com/blogs/

saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/, accessed 19 Nov 2018.

Appendices

Survey for students and tutors



Event Participation Tracking with IoT (STUDENTS)

Currently, a system of student participation monitoring relies on paper & pen method:
Tutors bring an A4 paper sheet to every tutorial, and each student has to write down their name and student id.
Afterwards, either tutors or ITO staff has to gather the paper sheets are then manually enter the information into a database by typing everything in.

We are proposing a solution which uses the Internet of Things (IoT) devices to automate monitoring process:
Tutor brings a device with a student id reader and a display. Each student has to tap their student card when arriving to the tutorial room, and screen notifies that the presence was recorded.
Collected presence data is automatically uploaded to ITO database.

*Required

Which year of studies are you in? *

- 1st year (Freshman)
- 2nd year (Last year of maths courses, yay!)
- 3rd year (SDP is killing me)
- 4th year (I'll start writing my thesis tomorrow)
- Masters
- Other:

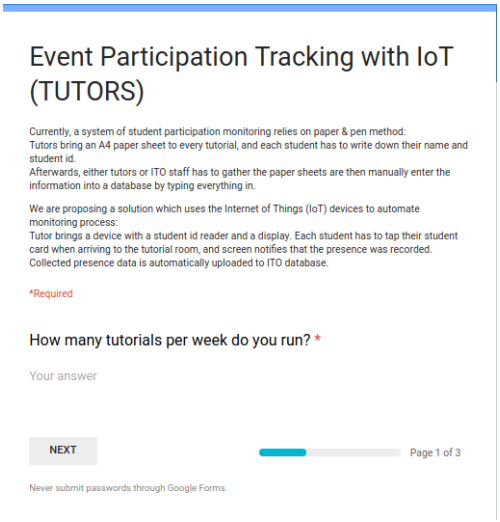
How many tutorials per week do you have? *

Your answer

NEXT Page 1 of 3

Never submit passwords through Google Forms.

Figure 1: First page, students



Event Participation Tracking with IoT (TUTORS)

Currently, a system of student participation monitoring relies on paper & pen method:
Tutors bring an A4 paper sheet to every tutorial, and each student has to write down their name and student id.
Afterwards, either tutors or ITO staff has to gather the paper sheets are then manually enter the information into a database by typing everything in.

We are proposing a solution which uses the Internet of Things (IoT) devices to automate monitoring process:
Tutor brings a device with a student id reader and a display. Each student has to tap their student card when arriving to the tutorial room, and screen notifies that the presence was recorded.
Collected presence data is automatically uploaded to ITO database.

*Required

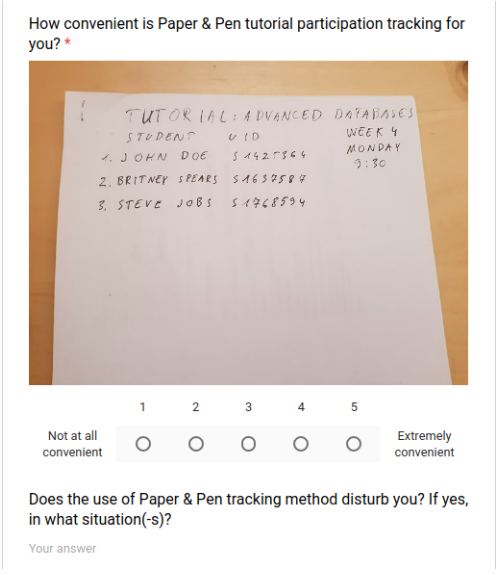
How many tutorials per week do you run? *

Your answer

NEXT Page 1 of 3

Never submit passwords through Google Forms.

Figure 2: First page, tutors



How convenient is Paper & Pen tutorial participation tracking for you? *

	1	2	3	4	5	
Not at all convenient	○	○	○	○	○	Extremely convenient

Does the use of Paper & Pen tracking method disturb you? If yes, in what situation(-s)?

Your answer

Figure 3: Second page, shared. The paper is written by hand to represent real situations.

Figure 4: Beginning of third page, shared.

Situation 1, You come to the right tutorial & tap the card.

Figure 5: Situation 1, shared.

Situation 2, You are in a wrong tutorial room, and we give you directions to a right one.

Figure 6: Situation 2, shared.

Situation 3, You mixed up a day, and randomly popped up at a random tutorial.

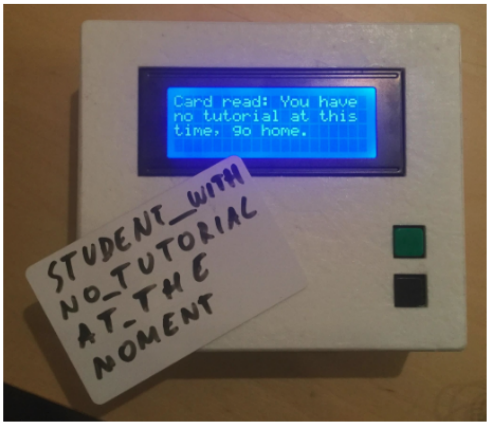


Figure 7: Situation 3, shared.

Situation 4, You are hacker with fake card.

Figure 8: Situation 4, shared.

How convenient is IoT(Tap Student ID on a card reader) tutorial participation tracking for you? *

	1	2	3	4	5	
Not at all convenient	○	○	○	○	○	Extremely convenient

How easy to understand is the interaction with an IoT tracking device? *

	1	2	3	4	5	
Not easy at all	○	○	○	○	○	Extremely easy

If you would add a feature to this device, what would it be?

Your answer

BACK SUBMIT Page 3 of 3

Figure 9: Situation 3, shared.

How convenient is IoT Solution (Tap Student ID on a card reader) tutorial participation tracking for you? *

	1	2	3	4	5	
Not at all convenient	○	○	○	○	○	Extremely convenient

How easy to understand is the interaction with an IoT tracking device? *

	1	2	3	4	5	
Not easy at all	○	○	○	○	○	Extremely easy

Would IoT device have any impact on your tutorial?

Your answer

If you would add a feature to this device, what would it be?

Your answer

BACK SUBMIT Page 3 of 3

Figure 10: Situation 4, shared.

Smart Offices and Event Participation Tracking via Internet of Things

4th Year Honours Project by Justin Alisauskas
Supervisor - Paul Patras

Event participation monitoring

Currently, a monitoring of student attendance at the tutorials and labs relies on the paper & pen method:

I. Tutors bring an A4 paper sheet or a table with list of people in the group to the tutorial.
II. Each student has to write down their name and student id.
III. Afterwards, either tutors or ITO staff have to manually enter this information into a database.

A single course costs estimation for paper and pen method - **£802.4** (Only for the time taken to do the manual steps)

We are proposing a solution which uses the Internet of Things (IoT) devices to automate monitoring process:

I. Tutor brings a device with a student id reader and a display.
II. Each student has to tap their student card when arriving to the tutorial room, and screen notifies that the presence was recorded.
III. Collected presence data is automatically uploaded to ITO database.

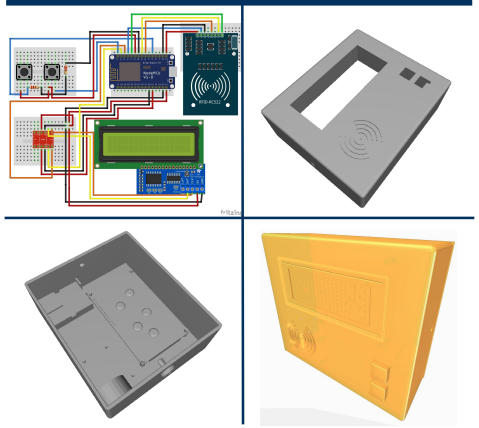
Bonus: If you went to the wrong room, our IoT device will give the directions to the right location

The Cloud

- Our IoT application heavily relies on a Cloud infrastructure. IoT devices serve as the endpoints which gather data and send it to the main hub – the Cloud.
- The Cloud handles interaction with the IoT device via a custom API.
- It processes requests and does the heavy lifting, such as parsing Twitter Feed, sending SMS via Twilio API and querying, as well as modifying the database.
- The use of Infrastructure as a Service (IaaS), in our case - Amazon Web Services, allows to scale on demand, reducing the costs associated with the upkeep of the system.
- We embrace the IoT shift away from devices that must to do everything on their own. tTansitioning to a cloud oriented development, enables the use of extremely low powered IoT devices.

Our Goals

- Our main focus is to identify areas within University, which could benefit from IoT and then create a working 'proof of concept' solution.
- We aim to show that IoT applications are the key to improving work conditions in university campuses, and in enterprise companies, saving time, energy and hard-earned money with minimal investment
- Our functioning examples are here to prove that IoT devices are useful not only in personal use applications, but in the education market and enterprise environment as well.
- Even with small budget we still succeed in creating several iterations of a prototype device, showing that the biggest concern – high development and implementation cost – is nothing but a myth.



Smart office device

Leave a note from anywhere

We have developed an IoT device, that is aimed to replace the sticky notes used to put on reminders or notes for colleagues when leaving the office.

No matter where you are, just Tweet to your private account and our IoT device will display it for everyone in the office to see.

Be notified about people waiting for you

There are times when we forget a meeting and go out for lunch instead, or an unexpected guest arrives at our office while we are not present.

Since communication between colleagues and supervised students is mainly based on emails, it gets incredibly difficult to reach out to someone in a hurry. Especially if one of the conversation participants is not proactive inbox-checker.

For these cases, our IoT device has an SMS notification system. With a press of a button, a device owner gets the notification about visitors presence. Let's call it an Internet-connected doorbell.

These features work well together

- A Personal Tutor arranged a meeting with a student. Unexpectedly a PT has to leave for another, more important meeting. He/She then tweets to our IoT device "Sorry, going to be late. Be there in 20 mins."
- Student arrives at the Informatics Forum, walks up to a 3rd floor and sees a message on the device. Since this person is extremely impatient, he/she presses a button to send a SMS notification to the PT, informing PT that there is a visitor at the door.

Bonus: While the device is not in use, it shows weather forecast, e.g. to inform passers-by about an upcoming apocalyptic snowstorm.

The IoT Device

- Our IoT device is using a low-power NodeMCU board, with embedded WiFi chip. With addition of power saving mode, even the prototype has an expected battery life of more than 2 weeks.
- Device uses RFID sensor to read the UID of a student card. The UID, as well as device ID, are then sent to a Cloud, which processes the data. The response from a server is then shown on the screen of IoT device.
- The two b
- We designed and 3D printed several iterations of our device enclosure, showing that sensors still fun

Figure 11: 22nd of March Presentation poster