Bachelor’s thesis

Information and Communications technologies

2021

Jesse Smedberg

CLOUD-BASED IOT ELECTRIC SCOOTER

– Benefits of micromobility vehicles and cloud services for individuals

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and communications technologies

2021 | 42, 7

Jesse Smedberg

CLOUD-BASED IOT ELECTRIC SCOOTER

Benefits of micromobility vehicles and cloud services for individuals

The popularity of personal light electric vehicles has seen a significant increase in recent years. In city streetscapes, the popularity increase is most noticeable as electric scooters and electric cars. This thesis documented a project where an electric scooter comparable to a rentable electric scooter was built.

The requirements for this thesis project were acquired by researching the features of rentable and commercial electric scooters. In the thesis project, an electric scooter was implemented according to the specifications of the acquired motor. Additionally, GPS tracking, always online capabilities, and remote start-up were implemented on the electric scooter so that, the electric scooter could be tracked and turned off remotely if stolen.

The back end was built on Amazon Lightsail based on the Flask web framework. The endpoints were first tested locally with Postman and over the internet after routing was implemented with NGINX and uWSGI. The electric scooter's data is sent to the back end from an Arduino MKR1500 using NB-IoT radio technology, where it is processed and saved into MongoDB. The device's reliability was tested with an overnight stress test, where the Arduino continuously sent data to the back end. During testing, the HTTP library used in the project was found getting stuck when sending or receiving data multiple times, interrupting the running program indefinitely.

The goals of the thesis project were achieved adequately, and a minimum viable product was built. However, the electric scooter's web communications are unreliable because of issues found during testing with ArduinoHttpClient. The scooter's development will be continued after the completion of the thesis, with the goal of fixing the issues found during testing such as, changing the HTTP protocol used to the MQTT protocol meant for IoT applications.

KEYWORDS:

Micromobility, Cloud-server, Amazon Web Services, Arduino, IoT, Electric scooter

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2021 | 42 sivua, 7 liitesivua

Jesse Smedberg

PILVIPOHJAINEN IOT-SÄHKÖPOTKULAUTA

Henkilökohtaisten sähkökulkuvälineiden suosio on viime vuosina lisääntynyt huomattavasti. Kaupunkien katukuvassa suosio on näkynyt parhaiten sähköpotkulautojen ja sähköautojen yleistymisenä. Tämän opinnäytetyön tarkoituksena on dokumentodia projekti, jossa rakennetaan ominaisuuksiltaan vuokrattavaan sähköpotkulautaan verrattavissa oleva sähköpotkulauta.

Opinnäytetyön vaatimukset selvitettiin tutkimalla vuokrattavien sekä myynnissä olevien sähköpotkulautojen ominaisuuksia. Opinnäytetyössä toteutettiin sähköpotkulauta, joka rakennettiin hankitun sähkömoottorin mukaan. Lisäksi sähköpotkulautaan tehtiin GPS-seuranta, onlinevalmius sekä etäkäynnistys ominaisuudet. Näin potkulaudan sijaintia voidaan seurata mahdollisen varkauden tapahtuessa ja sekä potkulauta voidaan kytkeä pois päältä kontaktittomasti.

Potkulaudan ominaisuuksia varten rakennettiin Amazon Lightsail alustalle Flask verkkokehykseen pohjautuva backend, päätepisteet testattiin toimiviksi Postman sovelluksen avulla ensin paikallisesti ja sen jälkeen verkon ylitse NGINX ja uWSGI reitityksen valmistuttua. Potkulaudalta data lähetettiin backendiin Arduino MKR1500 NB-IoT:lla, josta se tallennettiin MongoDB:hen. Laitteiston luotettavuus testattiin yön yli kestävällä stressitestillä, jossa Arduino lähettää jatkuvasti dataa palvelimelle. Testauksen aikana Arduinossa huomattiin projektissa käytetyn HTTP- kirjaston pysäyttävän ohjelmansuorituksen.

Projekti saavutti tavoitteet pienimmän toimivan laitteen toteutuksesta. Sähköpotkulaudan verkkoliikenne on kuitenkin vielä epäluotettavaa ja ohjelmiston kehitystä pyritään jatkamaan tulevaisuudessa eteenpäin toteutuksen aikana opittujen vajaavaisuuksien pohjalta mm. vaihtamalla HTTP-verkkoprotokolla IoT-laitteisiin tarkoitettuun MQTT-kommunikaatioprotokollaan.

ASIASANAT:

micromobility, cloud-server, Amazon Web Services, Arduino, IoT, electric scooter

CONTENTS

APPENDICES

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

A	Amperes
ADC	Analog to digital converter
API	Application Programming Interface
ARM	Advanced RISC Machines
AVR	Advanced Virtual RISC
AWS	Amazon Web Services
B	Byte
BLDC	Brushless direct current
BMS	Battery management system
CAN	Controller Area Network
CPU	Central Processing Unit
DC	Direct current
EC2	Elastic Compute Cloud
ESC	Electronic speed controller
FaaS	Function as a service
FSESC	A version of VESC produced by Flipsky
GET	HTTP request method used for getting data
GPS	Global Positioning System
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
Hz	Hertz
IaaS	Infrastructure as a service
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Ingress Protection Code
IP	Internet Protocol address
I2C	Inter-Intergrated Circuit

JSON	JavaScript Object Notation
jQuery	A JavaScript library for easy HTML DOM manipulation and event handling
Kbit/s	Kilobits per second
LTE	Long-Term Evolution
LTE-M	Long-Term Evolution for Machines
mAh	milliampere-hour
MKR1500	Arduino MKR1500 NB-IoT
MongoDB	Mongo database
MoSCoW	A method used to prioritize development
MQTT	Message Queuing Telemetry Transport
NB-IoT	Narrowband Internet of Things
NCA	Lithium Nickel Cobalt Aluminum Oxide battery cathode
NGINX	An open-source web server software.
NoSQL	Not only SQL
PaaS	Platform as a service
POST	HTTP request method used for sending data
RAM	Random Access Memory
REST	Representational State Transfer
RTOS	Real-time Operating System
SaaS	Software as a service
SPI	Serial Peripheral Interface
SQL	Structured Query Language
SSH	Secure Shell protocol
UART	Universal asynchronous receiver-transmitter
uWSGI	Web Server Gateway Interface implementation usually used with Python web applications
V	Voltage
VESC	An open-source electronic speed controller created by Benjamin Vedder.

1 INTRODUCTION

In recent years, the world has seen significant developments in personal transportation's electrification. The change is seen as an increase in electric bicycle availability at more affordable price points and in the forms of electric vehicles available. Besides electric bikes, other prevalent forms of electric vehicles are electric scooters and electric cars. The electric scooter has become such a norm in major city streetscapes that it has become almost unnoticeable how often rentable scooters are seen despite their eye-catching colors. The four rentable scooter companies most people living in major cities are likely to be familiar with are LIME, Bird, TIER, and Voi, which were all founded during 2017 and 2018. As cities are becoming more environmentally conscious, low and no-emission zones are being established. Micromobility vehicles are a great way to have access to the facilities around these zones fast. These qualities make the rentable scooter an appealing choice for short-distance travel around the city. However, the insignificant payments of a scooter's rent accumulate rather quickly into a sizeable amount of money. The savings and personal mobility make owning an electric bike or scooter an appealing option.

The objective of this thesis project is to build an electric scooter for short-range, low carbon emissions travel, with the goal of making a minimum viable product.

The thesis is structured as follows. Chapter 2 introduces the technologies on a general level. Chapter 3 discusses the requirements for the thesis project. Chapter 4 covers the implementation based on requirements. Chapter 5 analyzes test results, and chapter 6 summarizes what was successful and lacking and what will happen in the future with the project.

2 TECHNOLOGIES

This chapter covers the technologies used in the thesis project. This chapter introduces microcontrollers and their usage in IoT applications, battery technologies, and cloud services. The chapter’s structure goes from metal to cloud, meaning the introduction starts with material things. Chapter by chapter, the introduction covers more and more non-material subjects, with each subchapter introducing a subject with increasing computing power, emphasizing cloud services.

2.1 Battery

2.1.1 Lithium battery chemistries

The most common battery chemistry is a lithium-based cathode paired with a graphite anode and either a gel or a solid electrolyte and a porous separator in between. A Lithium Nickel Cobalt Aluminum Oxide cathode, or NCA for short, providing the best overall performance compared to the other chemistries. Lithium with Cobalt Oxide, Manganese Oxide, and Nickel Manganese Cobalt Oxide reach similar voltage levels but lack capacity. Chemistry gaining traction as an electric vehicle battery in recent years is Lithium Iron Phosphate, or LFP, because of their long life span, lower cost compared to NCA, and general safety compared to other chemistries. [1]

NCA is a jack-of-all-trades chemistry, with the only compromise in the safety of the cells. After the battery starts burning, it is nearly impossible to put out the flames, and it is preferred to let bigger lithium-Ion fires burn out. LFP batteries are generally less prone to fires thanks to their higher thermal resistance compared to NCA batteries. LFP batteries’ safety and cost are the main reasons for the chemistry being eyed by the electric vehicle industry. [2]

2.1.2 Cell form factors

The 18650 is a form factor first developed in the early 1990s and later standardized in IEC 62133. The cells have a radius of 18mm and a length of 65mm. This sizing gives the form factor its name, with the extra zero coming from the exact length. [3]

Figure 1. A 18650 cell.

The cells using NCA have an excellent energy density, providing up to 231.5 Wh/kg of energy, with a peak voltage of 4.2 V and a capacity of 3400 mAh. The typical cell weight is around 45.5 g. The energy density makes 18650 cells great for applications where the energy sources weight matters, such as an electric vehicle. [4]

Tesla has developed their version of the 21700 Li-ion cells, which is supposed to be the successor to the popular 18650 cells. The 21700 has an improved energy density of 300 Wh/kg, compared to the 231.5 Wh/kg a good quality 18650 has, which is an increase of 29,5% to energy density, with a 25% increase in the cell weight on average. The increase in energy density means fewer cells need to be produced to achieve the same amount of energy storage. [5]

2.1.3 Light electric vehicle battery packs

Electric bikes and scooters come in a few semi-standard voltages. The most common nominal voltage ratings are 24, 36, and 48 volts. However, some manufacturers express their battery packs peak voltage instead of the nominal voltage, which could cause some confusion to less knowledgeable consumers. A typical 48-volt battery pack is comprised of thirteen series and six parallel 3.7-volt NMC or NCA cells. A battery pack usually has a Battery Management System (BMS), which keeps the cell charge on a similar level, extending the pack’s lifespan.

The e-bike industry has tried to standardize the battery pack connectors but to no avail. The industry is full of peculiar designs, with almost every manufacturer using a proprietary connector, like the Bosch’s connector in Figure 2. The Rosenberg Power Data Connector was pushed as the standard connector in the mid-2010s, but it never gained enough traction to become the go-to connector. However, the connector is still being used in Specialized e-bike batteries. [6]

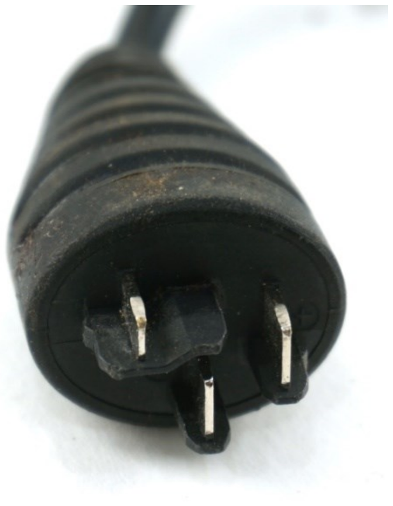


Figure 2. A bosch battery connector.

2.2 Drivetrain

A drivetrain is a combination of multiple parts, which work together to get power to the wheels. These typically include the Electronic Speed Controller (ESC), drive belts, gears, motor, and batteries. Typically in an electric scooter, an ESC, a motor, and a battery make up the drivetrain.

2.2.1 Basics of brushless dc motors

Brushless direct current motor works by changing the polarities in the stator wires in a way that attracts and repels the wires in relation to permanent magnets, as depicted in Figure 3 [7]. The gray arrows in Figure 3 show the direction in which the wheel is spinning, and the red and blue arrows show which way the stators are pushing and pulling the outer part’s permanent magnets. The motors in electric vehicles are usually three-phased outrunner motors. An outrunner motor has its permanent magnets outside, a configuration used in hub motors. An outrunner motor is depicted in Figure 3. Switching the stators to the outside makes the motor an inrunner, which is used in mid-drive motors. They need an external controller called an Electric Speed Controller, an ESC for short, which controls the modulated signal to the stators so that the motor spins.

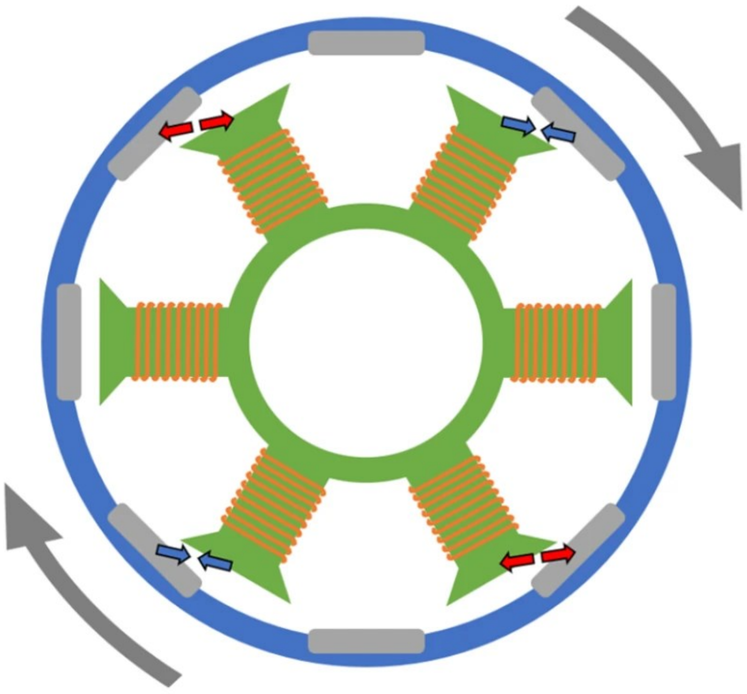


Figure 3: Brushless Motor Diagram

Figure 3. Brushless outrunner motor diagram.

2.2.2 Brushless DC motor types

BLDC motors come in many form factors, hub, mid-drive and separate inrunner Brushless DC motor. Mid-drive motors are not applicable in electric scooters, but their prevalence in electric bikes makes them worth mentioning. Mid-drive motors can provide a small footprint all-in-one motor package with sensors and actuators.

The hub motor is most often seen in electric scooters around the cities and makes an excellent option for a low maintenance system in a commercial setting. Hub motors have close to zero moving parts, making them less prone to wear, unlike a gear system used with a separate BLDC or mid-drive motor.

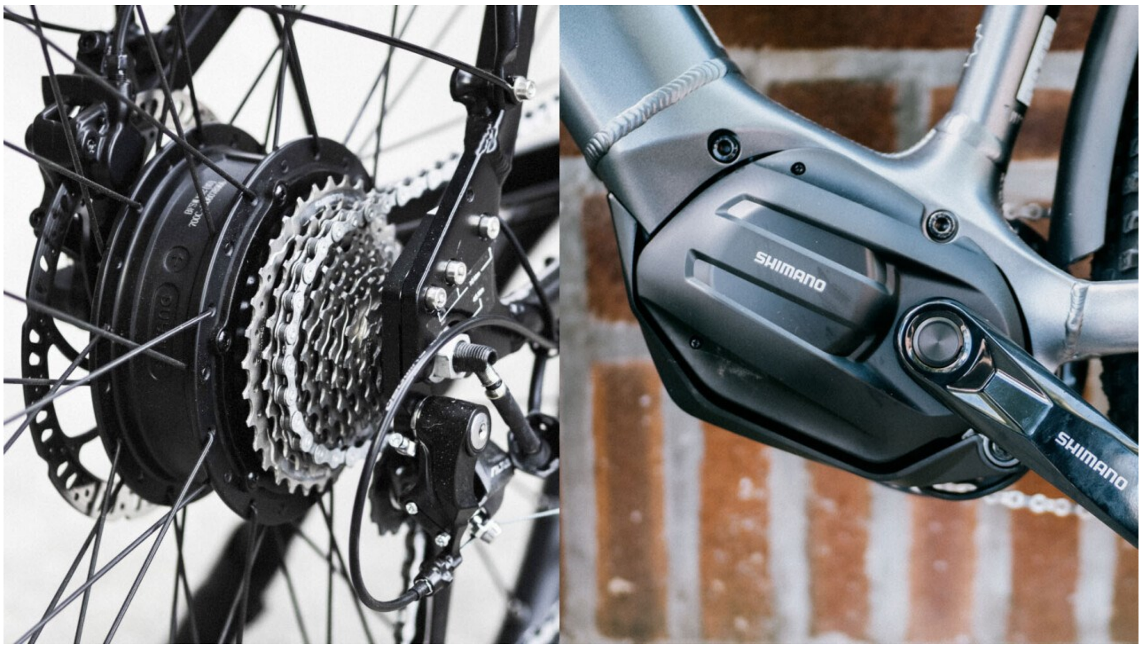


Figure 4. Hub motor (left) and mid-drive (right).

Separate inrunner BLDC motors are often used in electric skateboards because the motor's shaft can be fitted with either a belt drive or a wheel when used as a direct drive motor.

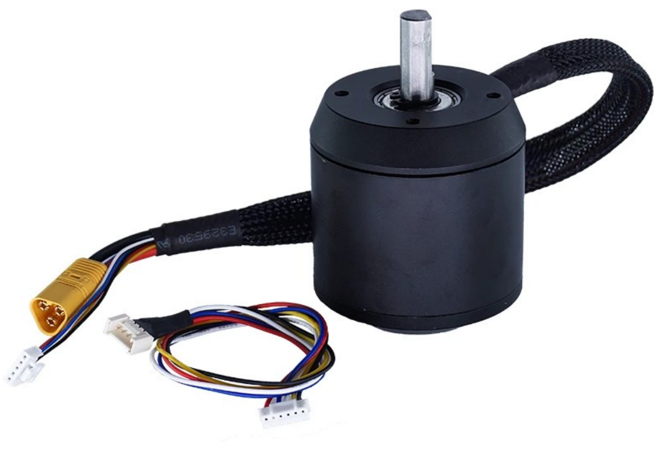


Figure 5. A Flipsky inrunner motor.

2.2.3 Electronic Speed Controllers

Electronic Speed Controller’s primary purpose is to produce signal similar to the signals seen in Figure 3 according to the control signal received. The low side MOSFET, which is denoted with a minus sign in in Figure 6, is kept on, while the high side is switched to create the magnetic field through the appropriate stator wires. The control signal can be either a PWM signal from a microcontroller via cable or Bluetooth or a value from a potentiometer throttle. Modern ESCs are usually comprised of a microcontroller, a gate driver integrated circuit, and MOSFETs controlled by the gate driver. ESCs usually come as black boxes with no way to know what is happening inside, although this is less common, thanks to Benjamin Vedder. [8]

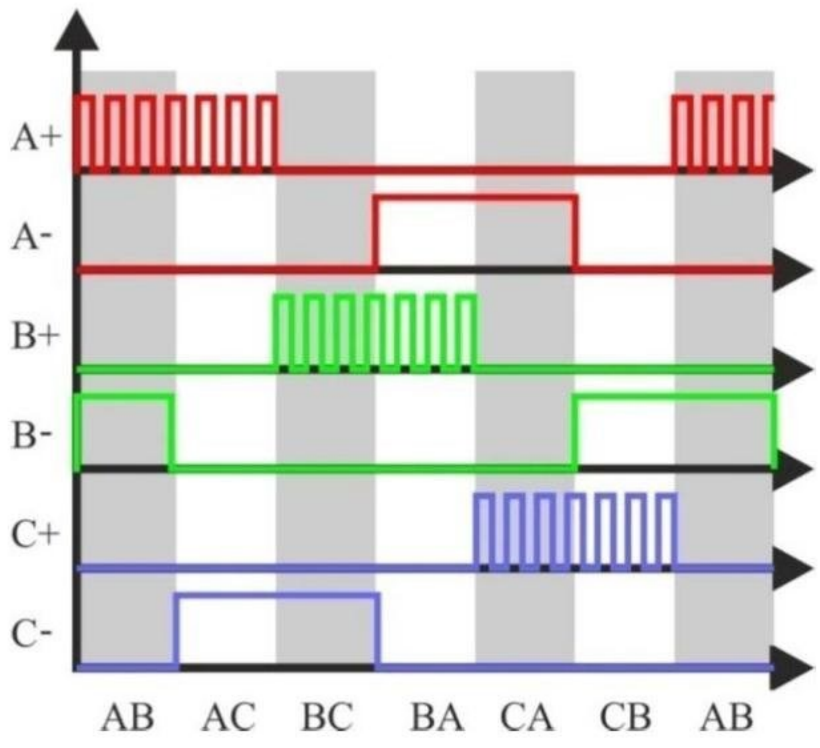


Figure 6. Typical High side switching BLDC commutation.

Benjamin Vedder started the project in 2011, and the Vedder ESC had its first release in 2014. The VESC is an open-source ESC with a plethora of settings to be tinkered with. The VESC has been the base for many variations, such as the FSESC used in the thesis project, with higher voltage and current ratings than the original 60V 50A model. The VESC has an STM32F4 microcontroller, a DRV8302 gate driver, and IRFS7530 MOSFETs. The VESC project has been a significant success, finding its way to numerous DIY builds worldwide, including this thesis project in the form of an FSESC. [8]

2.3 Microcontroller

2.3.1 What are microcontrollers?

During 1970 and 1971, Gary Boone invented the first microcontroller called TMS1000. The microcontroller came with three whole kilobits of program memory. The microcontroller didn’t see widespread adoption on release, but rather a few years later, in 1974, it was made available to the general market. The microcontroller then saw far-reaching usages in various devices, such as microwave ovens and video games. [9]

Microcontrollers can easily be confused with microprocessors. However, the differences between the two are pretty significant. A microcontroller should be thought of as an all-in-one package containing everything a simple computer would need. Microcontrollers have a mix of programmable memory for the program, input-output ports for communicating with peripherals, and interfaces like SPI and I2C. A microprocessor is there to process the data according to the instructions in the program memory. [10]

Microcontrollers function based on the data they get from their peripherals. For example, they can read the value of a thermistor connected to one of the pins, approximate the temperature based on the data read. Then transmit the data through UART or wirelessly with Bluetooth, NB-IoT, or Zigbee. Microcontrollers are the preferred option for many small electronics today, which need some logic because of their low power consumption, relatively high computing power, and low price.

2.3.2 Modern applications

Nowadays, microcontrollers are a cheap and easy way to get into embedded programming, with microcontroller board designers like Arduino, SparkFun, and STM32. The designs are often based around ARM Cortex-M and AVR ATmega series microcontrollers, like ATSAMD21 and ATmega328p. Arduino’s and Sparkfun’s boards are compatible with Arduino IDE, a great development tool for beginners with close to no setup.

Any device with even a hint of logic most likely has a microcontroller inside. From toothbrushes to coffee grinders, microcontrollers are truly everywhere. Microcontrollers internet connectivity is increasing year by year, with continuous development on better IoT networking protocols like Thread. Microcontrollers have developed to a point where their processing power is enough to run basic machine learning models.

In larger systems, a single microcontroller usually controls one functionality. For example, one microcontroller controls the lighting in a car, and another one controls the windows. In automotive applications, the controllers communicate with the central controller through CAN bus.

2.4 AWS and NB-IoT

The following subchapters first cover the IoT architecture as a whole, and then in the later chapters, cloud services and NB-IoT are covered. Cloud services are covered by the common types of cloud services, and NB-IoT's capabilities are presented.

2.4.1 Basic IoT architecture

A basic IoT architecture is as depicted in figure 7. On the lowest layer, there are the IoT devices themselves. They gather data, control devices and communicate with a hub or an edge device, with an appropriate IoT protocol. After acquisition, the data is sent to the cloud with an IoT network connection. There can be some data filtering and processing done in the edge device before the data is sent to a cloud database located in a data center. [11]

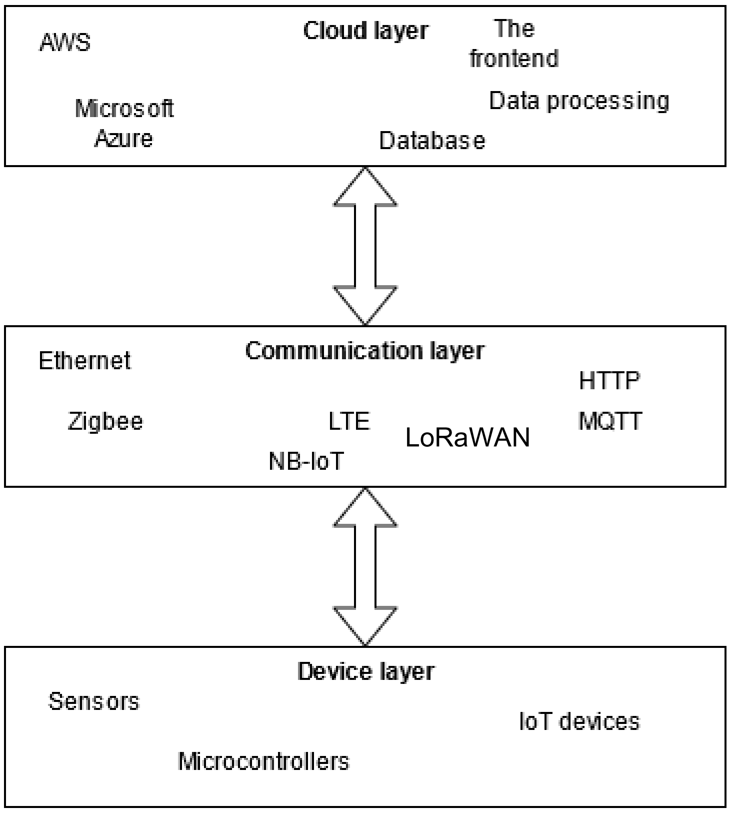


Figure 7. An example of what each of the IoT architecture layers contains.

The lowest layer contains all the devices that control and gather the data from the environment. The device could be anything from a wired security camera to a wireless door lock. This is the layer where the Arduino MKR1500 used in the thesis project is located.

Some popular networking layer protocols and networks are HTTP, MQTT, LoRaWAN, ZigBee, NB-IoT, and Ethernet. These protocols transmit data between devices in an IoT architecture, be it the hub or straight to the backend server. Ethernet and ZigBee, when used inside, are used close to their hub devices due to their limited ranges. NB-IoT enabled devices can skip the hub device altogether and connect straight to the cloud because the devices are using LTE networks to communicate with the backend server. HTTP and MQTT are messaging protocols, which determine the format of the data sent. [11]

The last layer in the IoT stack is the backend, where the data is processed and saved. The whole backend could be contained inside a single Amazon data center, thanks to all of the services Amazon provides. The data is then finally being processed and processed and cleaned before saving. The data can be used for simple visualization or training machine learning models and anything in between.

2.4.2 Cloud services

A cloud service, in short, is a service that runs in a data center, and the service is used through the internet. This lets the company making the service forgo building its own data infrastructure and implement the service in a data center. The data center can then focus on improving their servers and creating a cloud platform of their own for service deployments, like Amazon Web Services and Microsoft Azure. Both Amazon and Microsoft have built many services for their data center services, including databases, IoT gateways, and rentable machine learning computing power. They're both providing Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) for the end-user. [12]

Cloud services can be split into four as-a-service categories: software, platform, infrastructure, and function. Software-as-a-service is probably the most common cloud service that a consumer uses, and the category includes services like YouTube, Outlook, and MongoDB. Software services are used through the web without the need for an installation. Platform-as-a-service is what AWS and Microsoft Azure provides, with their corresponding platforms. They provide companies with infrastructure and a wide range of tools for deployment, all on a single platform. PaaS is paid by the services used, which lets the company pick and choose from the platform's services, thus possibly cutting costs. Infrastructure-as-a-service was used in the thesis project with Amazon Lightsail. IaaS is the most similar to renting a standard computer for use, but with ease of expandability and lower maintenance costs, thanks to the infrastructure being in the cloud. IaaS rents out computing power and storage to clients for building their service. Amazon Lightsail and Elasctic Compute Cloud (EC2) are pretty similar in the product they provide. Both services have a virtual private server builder tool, with Lightsail having a more streamlined process for deployment. Both services have premade profiles to suit the client's needs. Function-as-a-service forgoes standard server infrastructure altogether and provides the client an endpoint, which has a single function. The run time of these functions varies greatly depending on how often they are run, because the function gets unloaded from the server after a while. [13]

2.4.3 NB-IoT

Narrowband IoT is a 3GPP standardized radio technology, which lets IoT devices communicate through existing mobile carrier networks, either on the same bandwidth as GSM signals, between the main LTE channels, or standalone [14].

NB-IoT was first introduced in 3GPP release 13, and it is classified as a 5G technology [15]. However, NB-IoT does not necessarily use fifth-generation mobile networks to communicate because of its ability to work on older networks. NB-IoT has a peak sending data rate of 66kbps and a receiving data rate of 26Kbps, dropping as low as a few kilobits per second in hard-to-reach areas [16].

Figure 8. NB-IoT logo.

Compared to other low-power wide-area networks, NB-IoT has a clear disadvantage with its power consumption. According to Semtech, NB-IoT uses more power when sending and receiving than LoRaWAN. [17] Even when counting at the higher data transfer speeds, the power consumption of NB-IoT devices in a longer time scale significantly reduces the time between battery maintenances. Consuming 110mA on average per transmission is up to five times more power used per transmission than LoRaWAN. [17] NB-IoT makes up for the power consumption by working independently without an external hub. Devices using NB-IoT need a bigger battery to compensate for the increased power usage if the device is planned to be used without maintenance for years.

3 REQUIREMENTS

This chapter covers the requirements set for the thesis using the MoSCoW method [18] and studying the rentable scooters found in Turku. This thesis aims to create an electric scooter with a similar feature set comparable to Vois Voiager 3X. [19] The scooter to base the project is chosen on its similarity with TIERs and LIMEs scooters which look almost identical. Regarding that, Voi likely has not removed features from newer models of their scooters.

According to the Voi scooters website, the Voiager 3X has front and brake lights, breaks, fast charging, 4G connectivity, swappable battery, display with the current zone information, and availability status, which includes battery status. Having the scooter inform if parking is available or if the area you are currently in is a slow zone implies the scooter has GPS capabilities, which can be added to the list of features from the website. MoSCoW method is used to determine the necessary features and nice-to-have features. [19]

Table 1. Feature set implementation order determined with MoSCoW method.

1	The scooter must have GPS capabilities.	Must
2	A controller for motor.	Must
3	The scooter needs a place to save the data.	Must
4	The scooter is always online.	Must
5	The scooter needs a server with high uptime.	Must
6	The scooter can be remotely turned on.	Should
7	The scooter should have extra fault detection.	Should
8	Location and data should be visualized.	Should
9	An aesthetically pleasing website.	Could
10	Accelerometer to track the scooter without GPS	Could

3.1 Controller

As determined in table 1, the controller board must have internet connectivity, GPS capabilities, and enough interfaces to work with the peripherals if they're not on the board. GPS and internet connectivity should be on the controller board chosen for convenience. Arduinos MKR line of products, aside from the MKR 4000, all have internet connectivity in some form. However, only the MKR 1500 has both internet connectivity and GPS built-in. The MKR 1500 has internet through LTE-M and NB-IoT. The data transfer rates of both protocols are adequate for the thesis project, LTE-M sending and receiving data at a rate of 375 kbit/s and 300 kbit/s and NB-IoT sending and receiving data rate of 62.5 kbit/s and 27.2 kbit/s, according to the datasheet of the onboard u-Blox SARA-R410M-02B. [20]

3.2 Electronics

3.2.1 ESC

Benjamin Vedders open-source electronic speed controller has easy access to customizing the ESC to the needs of the project using the GUI made for it. It is rated for up to 60 volts and 50 amps continuous current. The ESC should be the centerpiece on which the rest of the vehicle is based around. Choosing the right ESC at the beginning makes the rest of the project more manageable. Because the VESC is open-source, plenty can be configured in the VESC tool's settings to the project's needs.

Thanks to the VESC being open-source, it has multiple alternatives with variance in price and power delivery capabilities. In the thesis project, an FSESC4.20 by Flipsky was used. The FSESC is based on VESC4.12 and has the components made to fit a smaller overall size otherwise, it is identical in functionality

3.2.2 Battery pack

The battery cell used for the electric scooter battery pack is the NCR18650B. The cell has a rated capacity of 3400 mAh, and an average tested maximum voltage of 4.180 V. The cell was chosen based on the manufacturer's reputability and the cell's capacity. Since VESC is rated for up to 60 V, the battery pack has to have a maximum voltage below 60 V. A battery pack with 14 cells in series is just below the rated voltage, but a 13s battery is recommended because the motor driver on the VESC is rated at 60 V, including voltage spikes.

It is highly recommended to add a battery management system to the battery pack. Otherwise, the battery pack has no balancing for the cells or regulation for the output amperage. The BMS used in the thesis project is a generic Chinese 13s BMS rated at a nominal voltage of 48 V and 40 A. The BMS limits the amount of current in and out of the battery to some extent.

Parallel cells should be added to the battery pack according to the motor's amperage rating. NCR18650B has a C rating of 2, making the maximum recommended current output 6.8 A per cell. The maximum current output from a li-ion cell is a lot higher than the recommended value. To not use the batteries at their limit, six cells are put in parallel to make the maximum current output 40.8 A, which is just over the maximum current output of the BMS.

3.2.3 Motor

The scooter's and driver's weight must be taken into consideration when choosing a proper motor. The scooter weighs approximately 40 kg, and the usual driver weighs between 60-70 kg. A 36 V motor should be enough to reach the legal speed limits comfortably with good acceleration as long as enough current is allowed to the motor at low speeds.

Considering the motor controller on the VESC, a three-phase motor must be used in conjunction with it. The motor chosen for the project was a 1000 W 48 V brushless dc motor for bicycles. It is rated as 470 RPM and with the motor being fitted to a 12-inch wheel that gives the scooter a maximum speed of 30 kilometers per hour. It needs to have its maximum RPM reduced in the VESC to match the legal speed limit of 20 kilometers per hour.

3.2.4 Anti-Spark Switch

In the thesis project, an anti-spark power switch was used to enable the remote power switching features. The anti-spark switch's button must be replaced with a relay for it to be controlled by the Arduino. In the thesis project, a Flipsky Anti-Spark Switch Smart Enhanced is used. It is rated for 13s li-ion battery configuration and 200 A continuous current. It is a MOSFET anti-spark switch. Anti-spark switches mostly come in two forms. A connector with a cable loop, which completes the circuit and MOSFET-based anti-spark switches where the MOSFETs are opened with, for example, a relay like in the thesis project.

3.3 Backend

Backend requirements in this thesis project are minimal. The server needs only to serve the electric scooter and presumably a couple of users on the website at once. The database is accessed seldom, and the queries are not complex.

Databases can be categorized into two groups, SQL and NoSQL. SQL stores data in tables with relational data marked with a primary-secondary key system, while NoSQL can be a document, key-value, graph, or wide-column stores, as stated in Mark Smallcombe's article [21]. The database of the thesis project does not have any special requirements and was chosen on its familiarity and ease of implementation. In the future, a time-series database should be used because of its applicability to IoT projects. Usually, IoT devices are used for tracking, logging, and controlling. IoT projects can make use of edge devices, which offload storage and computing from the IoT device. In the thesis project, an edge device was not used, and the server does data storage and data display.

3.3.1 Database

For the thesis project, MongoDB was chosen because of its ease of implementation, flexibility, and familiarity. A time-series database, like InfluxDB or AWS Timestream, should be used in the future. The goal of the thesis was to build a minimum viable product version of the electric scooter, so quick and straightforward implementation was prioritized. MongoDB, Inc. provides a free tier of their database with 512 MB of storage which is more than enough for the thesis project.

3.3.2 Server

The server's minimum requirements are minimal. It must run Ubuntu 18.4, which minimum system requirements are 1 GHz CPU frequency, and 1 GB of RAM.

In the thesis project, Amazon Lightsail was used to deploy the server and host it. Lightsail has multiple premade configurations to pick from, but Ubuntu configuration was used with 1 GB of RAM in the thesis project.

The web server was implemented following a guide by DigitalOcean. The guide goes through implementing the web server and reverse proxy with NGINX and uWSGI. NGINX serves as a reverse proxy for the uWSGI server, routing HTTP traffic to the right uWSGI processes, while uWSGI serves the requests to and from the flask application. [22]

The server application is based on Flask, a python web application microframework. It provides the bare minimum to start the project, with an extensive library of extensions to be tailored to the developer's needs. The Flask application serves the single-page web application and RESTful API.

3.4 IP-Rating

The chassis of the electric scooter's deck contains all of the vehicle's electronics. It must not let any water inside while driving in the rain or driving through a puddle. Also, the deck should not let an amount of dust inside that would harm the electronics. Consulting the IEC 60509:1989 standard, the rating required is at least IP56, at which point the deck would be protected from most of the dust particles and high-pressure water coming from any direction [23]. However, IP57 rating should be considered for extra protection.

4 IMPLEMENTATION

This chapter describes the implementation based on the requirements set in chapter 3.

The chapter starts with implementing the physical components, such as the motor, battery, and sensors. After which, the code inside the Arduino MKR1500 is covered. Finally, ending with the implementation of the cloud server and frontend.

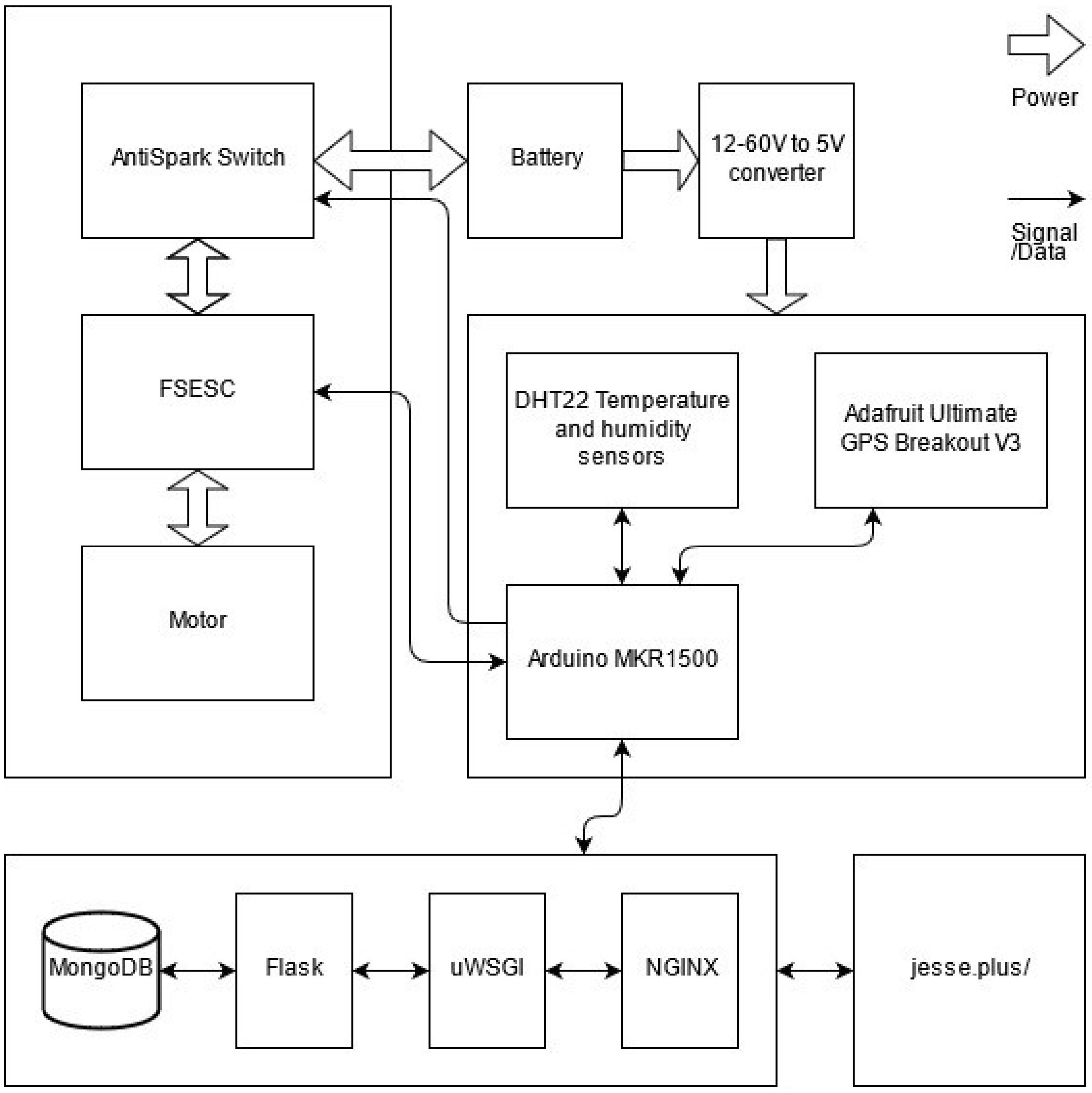


Figure 9. A block diagram depicting the technology stack of the thesis project.

The sensor block with the Arduino MKR1500 is further opened up in Figure 6, presenting the connections more clearly.

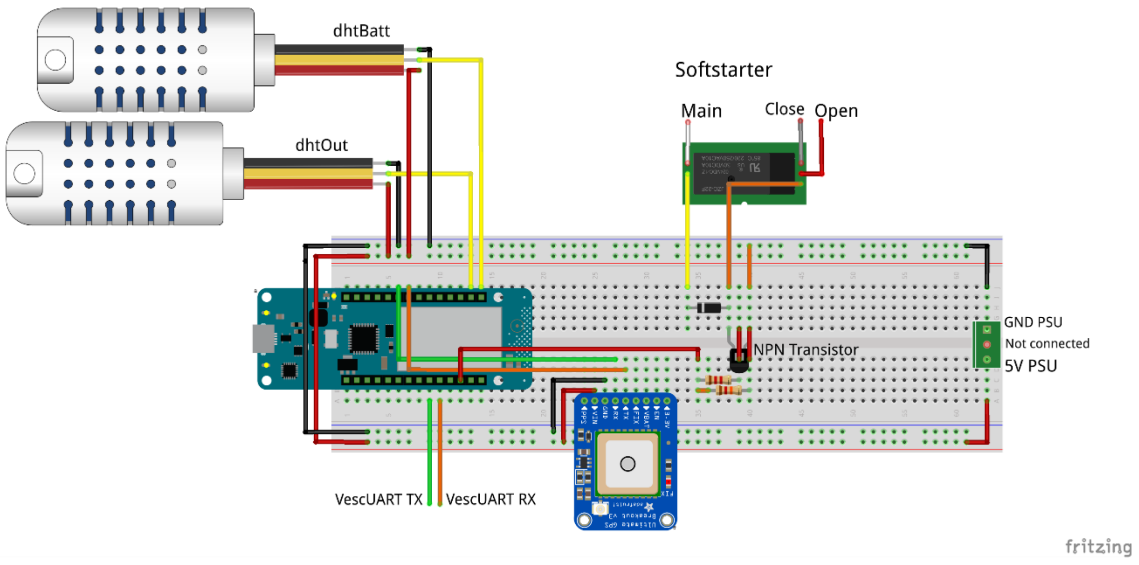


Figure 10. The sensor block diagram..

4.1 Electronics

4.1.1 AM2302

AM2302 module monitoring battery temperature is connected to Arduino pin 6, and module monitoring out temperature is connected to pin 7. The modules Vcc and ground pins are connected to the power rails of the breadboard. In Picture 1, these are located in the top left.

4.1.2 Adafruit Ultimate GPS Breakout V3

The module has its rx pin connected to the MKR1500s tx pin, and the module’s tx pin is connected to the MKR1500s rx pin. The module can then be communicated with using the hardware serial of the MKR1500. In Picture 1, this is located at the middle bottom.

4.1.3 FSESC

The FSESCs RX is connected to pin 1 on the MKR1500, and TX is connected to pin 0. The throttle is connected to the ADC, ground, and 3.3 Vpins on the VESC. Using an analog throttle with the FSESC, instead of a PWM signal from the MKR1500, separates the two devices, enabling the scooter to be still driven in case of a software failure on the MKR1500.

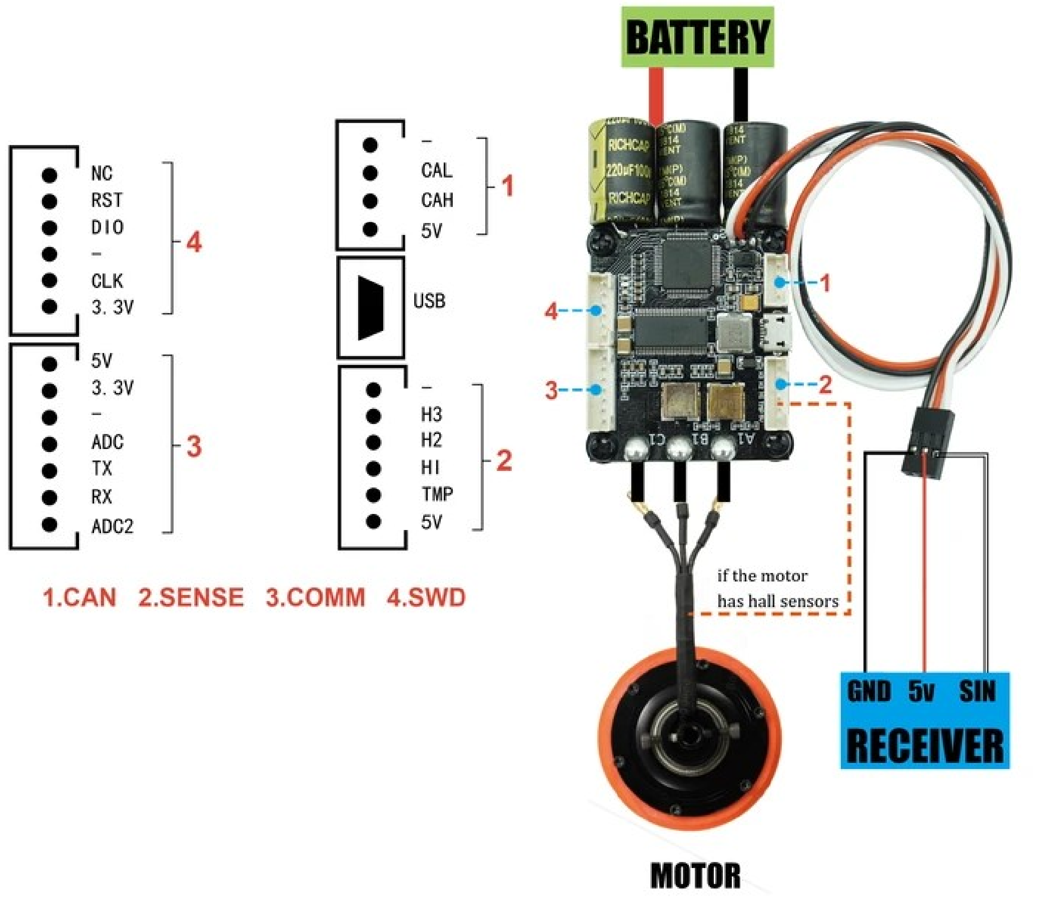


Figure 11. A picture of the Mini FSESC4.20 with the connector pinouts visible from Flipsky's website.

4.1.4 Power electronics

In the thesis project, a battery pack with a 13s6p configuration is used. It was built in a 26 by 3 cells configuration spanning most of the scooter's deck. The cells used have a maximum capacity of 3400 mAh according to the specifications and a voltage range of 2.5-4.2 V. There is a voltage cutoff set in the VESC to cut off power to the motor at around 38 V, which is at a cell voltage of around 2.9 V.

Battery output is connected with an XT60 connector to the 12-60 V to 5 V 3 A converter and the anti-spark switch through a 20 A fuse. The anti-spark switch's power switch is connected to a relay controlled by the MKR1500. The anti-spark switch is further connected to the FSESC. The converter is connected to the breadboard’s power rail and ground, where it serves power to the electronics requiring 5 V.

The anti-spark switch’s button cable has been modifier to go through a relay which enables the MKR1500 to control the power going to the FSESC. The cable was cut, and a connector was added in between the two ends, after which the cable was put together. The connector takes the 5 V cable from the three cables and splits it to the relay and the original button. This way, the button can still be used to switch the scooter on and off if the Arduino has a software or hardware failure.

4.2 Embedded software

Libraries

In an Arduino MKRNB project, Arduino.h should be omitted from the beginning, and MKRNB.h be used in its place. MKRNB.h adds the MKR1500's core functionality and is necessary when using the controller. DHT.h is from github.com/adafruit/DHT-sensor-library. The required libraries for the project are presented in Appendix 1.

Setup

The serial setup code seen in Appendix 2 is based on Sparkfun's guide, Adding More SERCOM Ports for SAMD Boards, with tweaks on which pins and which pads were used. The serial port has to be activated with pinPeripheral() before it can be used, after which it functions as a standard hardware serial port on the MKR1500 on pins 0 and 1. The created VUART serial port is then passed to SerialVESC.setSerialPort() as a pointer. [24]

Main loop

In the code seen in Appendix 3, the program checks if enough time has passed since the last reading. NOW is a macro that casts millis() to uint32_t type, and gps_endtime is the end time of the GPS block. These are used to check when to rerun the block. The controller's code should be migrated into a real-time operating system in the future. The type of programming blocks used in the current code is similar to an RTOS task. The change would improve the code's readability and make it easier to add functionality later. The main loop is presented as a whole in Figure 12.

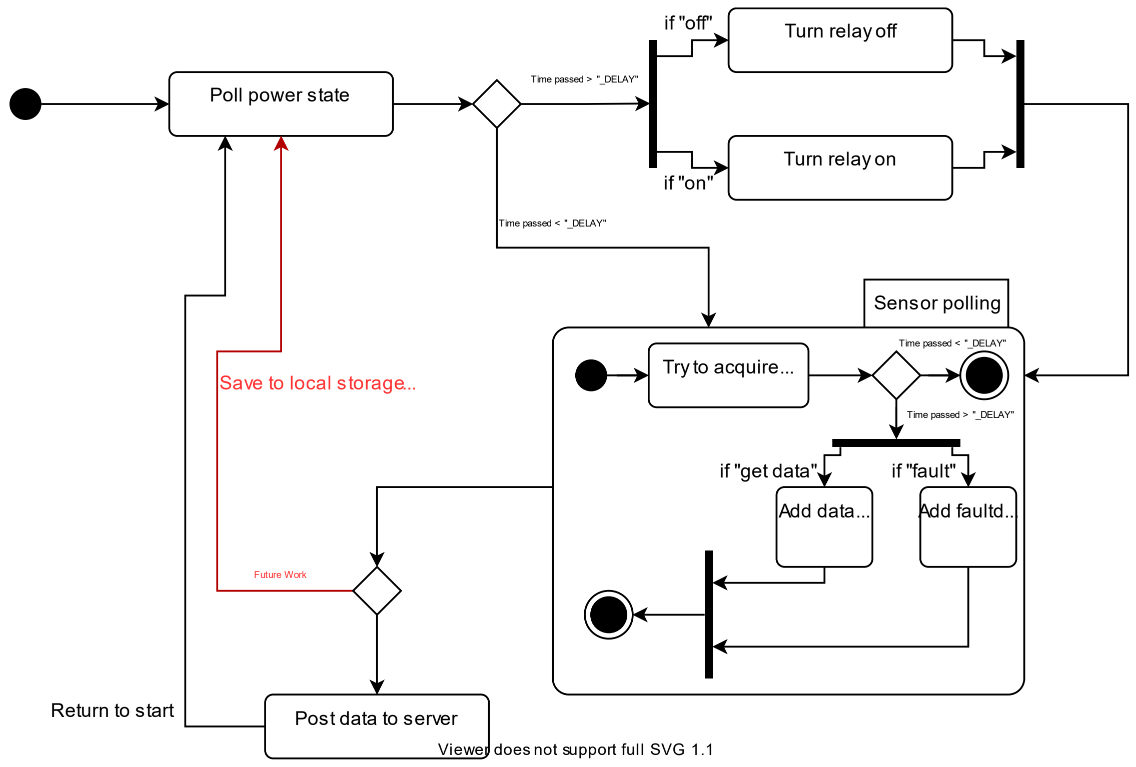


Figure 12. A state diagram of the main loop.

4.2.1 Data

Data is stored ArduinoJson library provided JSON format. The library's JsonDocument class variants work much like a JSON document typically would, except it has a defined size.

All the serial and 1-wire communications have to be initialized. DHT22 sensor is read with the objects readTemperature() function. The value returned is checked for validity with isnan(). If isnan() returns true, there has been a failure reading the temperature, and -1 is saved to the doc, and on false, the value read is saved to the doc. GPS data is acquired from SerialGPS when the Adafruit Ultimate GPS Breakout sends data to the serial port. The SerialGPS port is checked for available data periodically, according to the number of milliseconds determined by GPS_DELAY macro. The available data is then read and checked for updated data. If a new location has been acquired, and the GPS has a fix on satellites, the data is saved to a JsonObject location in the data JSON doc. Reading data from the FSESC is done through the added UART port. If the SerialVESC.getVescValues() function fails, -1 is saved to the doc's FSESC data fields. If data reading is successful, the read values are saved to the doc.

4.2.2 Power state check

The controller is periodically pinging the server to query if the scooter should be turned on or off. This is done by using the ArduinoHttpClient library get() function in the powerStateRequest() function. It sends a get request to jesse.plus/ipa/power_state, which responds with a 1 or a 0 corresponding with the power states. 1 being on, and 0 being off. The response is read with powerStateResponse(), verifying that the server return code was 200, and errors out if no 200 is returned or if the response takes over five seconds. These two functions should be combined to be a universal GET function.

The returned value is then passed to switchRelayState(), where it is checked for change. If change has happened, the relay is then turned on for either 700 milliseconds or 2000 milliseconds. A shorter relay on time closes the antispark switch circuit turning on the FSESC, and a longer on-time opens the circuit shutting down the FSESC.

4.2.3 Posting the data

Posting data to jesse.plus/ipa/controller is done with postData() function, using ArduinoJSON provided JSON format and ArduinoHttpClient client.post() function. The document is saved as a string so ArduinoHttpClient can work with the data. ContentType has to be specified as application/json as is required in HTTP. If the response code is not 200, the function reports the response code on serial, flushed and stops the client, and returns early. On a successful request, the response is logged onto serial, and the function returns 1.

4.3 Server and client

This chapter explains how the data moves in the electric scooter's IoT system, with the help of Figure 13, which presents the dataflow pattern of the system.

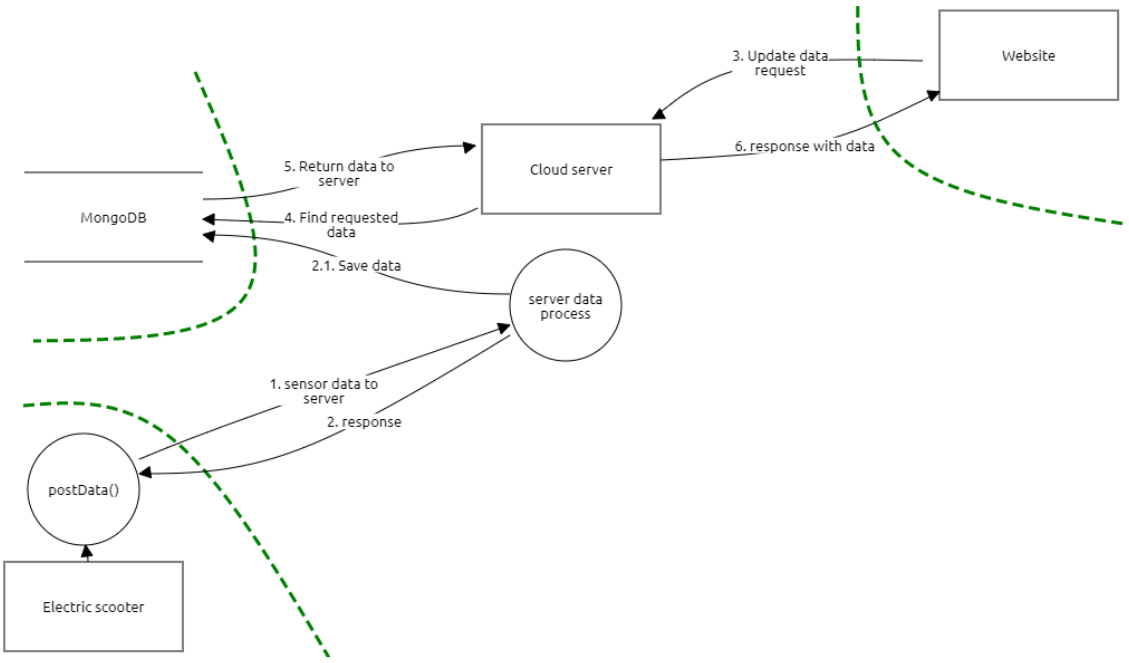
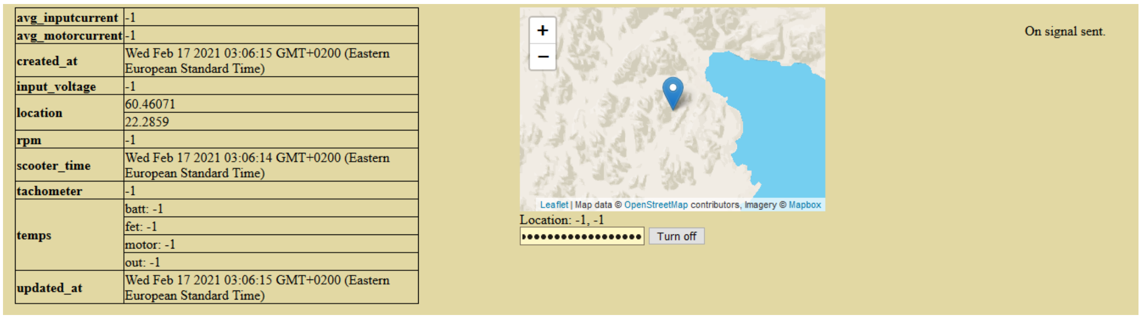


Figure 13. Dataflow diagram of the electric scooter.

4.3.1 Flask and frontend

Frontend

/ is the home page that displays the electric scooter's information and allows the electric scooter to be turned on and off remotely. The home page has javascript that keeps polling the Flask backend every five seconds to get the latest data sent from the electric scooter to the flask server. The home page communicates with the flask server with jQuery ajax calls, which are sent on button presses or at five-second intervals depending on if the scooter's data is being updated or if the scooters' power is being switched. The raw data is displayed in a table on the home page and location data is also displayed in a leafmap.js provided OpenStreetMap map element. The home page also contains a log of events that have happened during that client's open session. The data formatting and logging are done with jQuery.



avg_inputcurrent	-1
avg_motorcurrent	-1
created_at	Wed Feb 17 2021 03:06:15 GMT+0200 (Eastern European Standard Time)
input_voltage	-1
location	60.46071
	22.2859
rpm	-1
scooter_time	Wed Feb 17 2021 03:06:14 GMT+0200 (Eastern European Standard Time)
tachometer	-1
temps	batt: -1
	fet: -1
	motor: -1
	out: -1
updated_at	Wed Feb 17 2021 03:06:15 GMT+0200 (Eastern European Standard Time)

Location: -1, -1

On signal sent.

+
−
Leaflet | Map data © OpenStreetMap contributors, Imagery © Mapbox
••••••••••••••••••
Turn off

Figure 14. Picture of the website implementation made in the thesis project.

Flask

The main flask entry file is escooter_api.py. The backend is initialized in multiple parts. The main initialization function is called create_app and all of the resources and extensions are initialized inside their helper functions.

The routing on the backend is a restful API with implemented flask-restful extension. Flask-restful provides the Resource class, which is the main building of all the endpoints. The endpoints consist of classes that inherit the Resource class provided by flask-restful. The Resource class provides HTTP the usual methods used in REST APIs, which are GET, POST, PUT, and DELETE, however only GET and POST are needed.

The /ipa/controller has two HTTP methods associated with it, which are GET and POST. ControllerResource inherits the Resource class from flask_restful, which grants access to the needed HTTP methods.

The GET function seen in Appendix 4 returns the latest data from the database. It uses It pops the document id before returning, as it is not relevant information. The POST function does the opposite of the GET function and receives data from the electric scooter. The data is validated and the sender is confirmed with a long string identifier. Mongoengine is an Object-Relational Mapper for MongoDB. It provides connectivity between the server and the database, and a base document class that is used to create the Controller model for data validation.

The Document class from the mongoengine extension seen in Appendix 5 gives access to the data types used inside MongoDB documents, which enables easy validation of the incoming data. If the data received can't be saved into the appropriate field, it won't be saved into the database. No error handling for the wrong form of data is implemented in the version used in the thesis. The identifier, however, is checked to be valid and returns an internal server error.

The ControllerPowerResource is responsible for controlling the power state of the electric scooter in conjunction with the website. The state is stored in a variable inside the class, and it is retrieved with a GET method sent from the electric scooter. The state of the variable is changed by sending the desired state to /ipa/power_state with a POST method from the website along with a password.

4.3.2 MongoDB

In the thesis project, a free instance of MongoDB is used. It has 512 MB of storage, which is plenty for the application. After logging in to the MongoDB website, a new database can be made by going to the organization page and opening the project page. There a “New Project” button can be seen, and by pressing it, the user is taken to a page where the project is named. After the name is chosen, the user can add other users to the project. After creating the project, the user is taken to the Clusters page and should press the Build a cluster button and choose the free tier. In the configuration, the cloud provider should be AWS, cluster location should be eu-central-1 and its name can be changed if needed, but since the thesis project is only using one cluster to hold all the data, there is no need to change the name.

To connect to the database, the user must click the connect button, click the “Add Your Current IP address” on the pop-up window, add a username and password for the connection, and then choose to connect the application. Python is chosen as the application driver, and the connection string is seen in the box below. The connection string must then be copied over to the flask application so that mongoengine can connect to the database.

4.3.3 AWS Lightsail

The backend server is set up on Amazon Lightsail, which provides easy deployment of virtual private servers. In this thesis project, an “operating system only” instance is made with 1 GB of RAM. SSH keys must be created and downloaded so that the VPS can be connected to later. Automatic snapshots are not needed for the thesis project, but a snapshot of the server should be taken after the server is configured. The server should be assigned a static IP address to ease the development, which is done in the networking tab of the server.

After the server has initialized, it can be connected to using an SSH client. In the thesis project, Visual Studio Code Insiders is used because of its ability to act as an SSH client and a code editor on the host simultaneously. To connect to the server using Visual Studio Code, an extension called Remote – SSH (Nightly) has to be installed. After installing a new tab appears on the sidebar with an option to add SSH targets. When clicking add a new connection, the prompt then asks for an SSH connection string and the downloaded ssh key file. The server can then be interacted with using the VSCodes' built in terminal.

4.3.4 Nginx and uWsgi

After getting access to the VPS, the backend is configured by following a tutorial by Justin Ellingwood and Kathleen Juell from DigitalOcean [22]. The tutorial is followed until the end of step six, after which some more configuration must be done.

The projects escooter_api.ini file seen in Appendix 6 has its permissions modified from 660 to 666 so that NGINX and uWSGI can access the escooter_api.socket created when the service starts. The change done lets all the users read and write to the file. The permissions are changed because the current implementation of users on the back end is not finished.

5 TESTING

5.1 Electronics

During the first test with the finished drivetrain and electronics, a voltage spike caused by the battery being plugged damaged the microcontroller board. A Zener diode and a capacitor were added to the 60 V-to-5 V converter's output to prevent further damage to the circuitry. The Zener diode rated at 5.1 V is just over the other component's voltage ratings. The capacitor was added for redundancy, smoothing current spikes. The additional components succeeded in preventing further damage to the replaced parts.

The FSESC and AntiSpark Switch sustained the first few voltage spikes, but eventually, the AntiSpark Switch stopped working, as did the FSESC. When researching the cause for this failure, it became apparent that the AntiSpark Switches button was not supposed to be removed. The issue was fixed by getting a new AntiSpark Switch, where the button cable was modified to have a splitter. This way, the button functioned as it would out of the box, and it could be controlled with the relay.

5.2 Embedded software

Issues with the ArduinoHttpClient library arose during testing, which causes the software to hang while sending and receiving data. The hanging happens seemingly at random, with no apparent cause. At a minimum, it takes only a few minutes for the program to stop, with a maximum run time of a couple of days. The only way to restart the program was to reset the Arduino physically.

The problem persists and could be fixed in the future by migrating the scooter and back end to use MQTT instead of HTTP, which would likely increase the system's stability in the long run. The MQTT broker could run in the cloud with either a dedicated server, like AWS EC2, or an IoT messaging service, like AWS IoT Core.

Quite a bit of time was wasted on trying to get GPS working at all. The original GPS used for the project was the MKRGPS shield. The MKRGPS had issues finding a fix to the GPS satellites inside, but it worked fine outside. After a while, the MKRGPS stopped getting a fix altogether, at which point a new GPS breakout board was bought. The

Adafruit Ultimate GPS Breakout V3 with an external antenna also has trouble finding a fix inside, but it eventually succeeds, unlike the MKRGPS shield.

5.3 Back and frontend

While working on the server, I noticed a significant amount of disconnects on my SSH connection to the server when using a configuration with under 1 GB of RAM. The disconnects were fixed by switching to a server configuration with 1 GB of RAM. The server was running within Ubuntu's minimum specifications

The first implementation of the website was using WebSockets to communicate with the back end. There were some configuration errors on either NGINX or uWSGI, which caused the responses from the server to take an unbearably long time. However, the time between request and response was consistent at around twenty to thirty seconds. After being unable to pinpoint the issue, WebSockets were cut from the implementation in favor of a RESTful API. This fixed all the problems with server-client communication.

The current implementation of the map element does not use the data it gets from the server to draw the current location of the scooter. For unknown reasons, the map draws a marker in Greenland instead of Turku. The log has problems on smaller screens, where it goes behind the map element. The issue should be fixed by making the elements dependent on the user's screen size.

When setting up users and trying to prevent root access, the roles having ssh access got blocked out of the system. During development, this occurred multiple times before locking the root account behind a password and leaving it at that.

UWSGI had issues with permissions, which caused the escooter_api.socket file to be inaccessible at runtime. The problem was fixed by changing the permissions of escooter_api.socket in the escooter_api.init from 660 to 666

6 CONCLUSIONS

The objective of this thesis was to build an electric scooter with a similar feature set compared to a typical rentable electric scooter. The goals were met adequately. The controller and server work in unison, sending information both ways. The scooter's current status and location can be checked from its website. The scooter can be turned on remotely from the website using a password. Most of the must-be-implemented features from Table 1 were implemented successfully. Always-online functionality was not implemented in time because of issues found during testing of the vehicle.

The ArduinoHttpClient library raised several obstacles during development. Sometimes when sending a request to the server, the ArduinoHttpClient hangs and does not return anything, causing the controller Arduino to need a reset.

Website and server were first made using WebSockets, but those were deemed unusable. There likely were configuration errors on NGINX and uWSGI, which caused unbearable latencies when communicating using the website. Using a RESTful API on the server fixed all the issues with server-client communication.

Reflecting on the work implemented on the electric scooter opens many opportunities for future improvement. The codebase became quite difficult to read on the controller, and it is proving a great challenge to add features. NGINX and uWSGI were left to bare minimum configuration to get the other parts of the electric scooter to work. This meant leaving out HTTPS, leaving the website quite vulnerable until support is added. The FSESC is operating on its 60-volt limits with a fully charged battery, which is quite frankly bad design. Having voltage so close to the rated voltage of FSESC could easily cause hardware failure since the inductor kickback coming from the motor can reach incredibly high voltage.

Future work

To make future development easier, the program should be split up, and a real-time operating system should be implemented. The program is already operating like an RTOS, making switching systems a logical step.

The MKR1500 has an in-built GPS module in the SARA-R410 module. It should be used in tandem with the Adafruit GPS module to provide more accurate location tracking

ArduinoHttpClient must be changed in a later version so that MQTT could be the communication protocol. This change should provide more reliability than the ArduinoHttpClient, but it greatly depends on the implementation. The change also adds the need for an MQTT broker, which should be done with AWS IoT Core. MKR1500 has to have encryption enabled in the SARA-R410M module for AWS IoT Core to allow traffic from the microcontroller.

The server's security should be increased by adding authentication in Flask and adding HTTPS support in uWSGI and NGINX. At its current state, the website provides users the minimal features needed. In the future, users should see the trips that have been driven instead of just the current location. Website graphics should also be updated to provide the user increased clarity of what is happening in the electric scooter. The website should have pages added to provide more information about the project.

The electric scooter's acceleration is slow. The settings in the FSESC should be tweaked for better performance at lower speeds. Adding a HALL sensor could help at the beginning of the acceleration by helping the FSESC know the current electrical rotation of the motor and thus increase current on the correct phase at the right time.

REFERENCES

[1] Types of Lithium-ion Batteries – Battery University [Internet]. Batteryuniversity.com. 2021 [cited 16 May 2021]. Available from: https://batteryuniversity.com/learn/article/types_of_lithium_ion

[2] Colthorpe A. Choice of lithium iron phosphate not a 'silver bullet solution' for safety [Internet]. Energy Storage News. 2020 [cited 16 May 2021]. Available from: https://www.energy-storage.news/news/lfp-vs-nmc-not-a-silver-bullet-solution-for-safety

[3] A look at Old and New Battery Packaging – Battery University [Internet]. Batteryuniversity.com. 2020 [cited 24 April 2021]. Available from: https://batteryuniversity.com/learn/article/battery_packaging_a_look_at_old_and_new_systems

[4] Quinn J, Waldmann T, Richter K, Kasper M, Wohlfahrt-Mehrens M. Energy Density of Cylindrical Li-Ion Cells: A Comparison of Commercial 18650 to the 21700 Cells. Journal of The Electrochemical Society [Internet]. 2018 [cited 23 May 2021];165(14):A3284-A3286. Available from: https://iopscience.iop.org/article/10.1149/2.0281814jes

[5] All Things You Need to Know about Tesla 21700 / 20700 Battery [Internet]. DNK Power. 2019 [cited 24 April 2021]. Available from: https://www.dnkpower.com/teslas-mass-production-21700-battery/

[6] Connectors - Learn [Internet]. Ebikes.ca. [cited 23 May 2021]. Available from: https://ebikes.ca/learn/connectors.html

[7] Joner E. How Brushless Motors Work and How to Test Them [Internet]. RCbenchmark. 2020 [cited 28 April 2021]. Available from: https://www.rcbenchmark.com/blogs/articles/how-brushless-motors-work

[8] Vedder B. VESC Project [Internet]. Vesc-project.com. 2021 [cited 23 May 2021]. Available from: https://vesc-project.com/

[9] Ahmad T. Tracing the Origins of Arduino: Part 1: The AVR Microcontroller [Internet]. Element14.com. 2018 [cited 5 May 2021]. Available from: https://www.element14.com/community/docs/DOC-88981/l/tracing-the-origins-of-arduino-part-1-the-avr-microcontroller

[10] Difference between Microprocessor and Microcontroller [Internet]. Electronics For You. 2016 [cited 28 April 2021]. Available from: https://www.electronicsforu.com/technology-trends/learn-electronics/difference-between-microprocessor-and-microcontroller

[11] IoT technologies and protocols [Internet]. Azure.microsoft.com. [cited 5 May 2021]. Available from: https://azure.microsoft.com/en-us/overview/internet-of-things-iot/iot-technology-protocols

[12] What are cloud services? [Internet]. Redhat.com. [cited 6 May 2021]. Available from: https://www.redhat.com/en/topics/cloud-computing/what-are-cloud-services

[13] What is the cloud? | Cloud definition [Internet]. cloudflare.com. 2021 [cited 10 May 2021]. Available from: https://www.cloudflare.com/en-gb/learning/cloud/what-is-the-cloud

[14] IDG TECHtalk. What is NB-IoT? [Internet]. 2018 [cited 13 May 2021]. Available from: https://www.youtube.com/watch?v=pf7wcl1IZYc

[15] Narrowband IoT (NB-IoT) [Internet]. Thales Group. [cited 13 May 2021]. Available from: https://www.thalesgroup.com/en/markets/digital-identity-and-security/iot/resources/innovation-technology/nb-iot

[16] Le Bras R. What is the Difference in Data Throughput between LTE-M/NB-IoT and 3G or 4G? | Internet of Things [Internet]. Internet of Things. 2019 [cited 13 May 2021]. Available from: https://www.gsma.com/iot/resources/what-is-the-difference-in-data-throughput-between-lte-m-nb-iot-and-3g-or-4g

[17] Mohan V. 10 Things About LoRaWAN & NB-IoT [Internet]. Inside Out Semtech's Corporate Blog. 2018 [cited 12 May 2021]. Available from: https://blog.semtech.com/title-10-things-about-lorawan-nb-iot

[18] MoSCoW method - Wikipedia [Internet]. En.wikipedia.org. [cited 17 June 2021]. Available from: https://en.wikipedia.org/wiki/MoSCoW_method

[19] Voi Vehicles - Our Voi scooters and bikes [Internet]. Voi. [cited 13 May 2021]. Available from: https://www.voiscooters.com/voi-vehicles

[20] Product summary | SARA-R4 (x2B) series | Multi-band LTE-M / NB-IoT and EGPRS modules [Internet]. u-Blox; 2019 [cited 17 June 2021]. Available from: https://www.u-blox.com/sites/default/files/SARA-R4-x2B_ProductSummary_%28UBX-16019228%29.pdf

[21] Smallcombe M. SQL vs NoSQL: 5 Critical Differences [Internet]. Xplenty. 2020 [cited 14 May 2021]. Available from: https://www.xplenty.com/blog/the-sql-vs-nosql-difference

[22] Ellingwood J, Juell K. How To Install Nginx on Ubuntu 18.04 | DigitalOcean [Internet]. DigitalOcean. 2018 [cited 14 May 2021]. Available from: https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-18-04

[23] Rogers B. IP Rating Chart [Internet]. Setra.com. 2016 [cited 14 May 2021]. Available from: https://www.setra.com/blog/ip-rating-chart

[24] Adding More SERCOM Ports for SAMD Boards - learn.sparkfun.com [Internet]. Learn.sparkfun.com. [cited 14 May 2021]. Available from: https://learn.sparkfun.com/tutorials/adding-more-sercom-ports-for-samd-boards/all

The required libraries for the project

```
#include <MKRNB.h>
#include <ArduinoHttpClient.h>
#include <ArduinoJson.h>
#include <DHT.h>
#include <TinyGPS++.h>
#include "SSRelay.h"
#include <Adafruit_GPS.h>

//VescUart libraries
#include <VescUart.h>
#include <datatypes.h>
#include <crc.h>
#include <buffer.h>
#include "wiring_private.h"
```

Adding serial ports on SAMD21

```
//VescUART setup

VescUart SerialVESC;

//Initializes additional serial communications on pins 0 and 1.

Uart VUART(&sercom3, 1, 0, SERCOM_RX_PAD_1, UART_TX_PAD_0);

void SERCOM3_Handler() {

    VUART.IrqHandler();

)



//In setup()

//Assing RX function to pin 1

pinPeripheral(1, PIO_SERCOM);

//Assign TX function to pin 0

pinPeripheral(0, PIO_SERCOM);

SerialVESC.setSerialPort(&VUART);
```

Microcontroller main loop

```
//Main loop

void loop() {

    if (NOW - powerCheck_endtime > POWERCHECK_DELAY) {

            int pSReq = powerStateRequest();

            if (pSReq == 1) {

            Serial.println("power request sent");

        } else {

            powerCheck_endtime = NOW;

            return;

        }

        powerState = powerStateResponse();

        int sRSCode = switchRelayState(powerState);

        Serial.print("Relay code: ");

        Serial.println(sRSCode);

        powerCheck_endtime = NOW;

        return;

    }


    if (NOW - gps_endtime > GPS_DELAY) {

        getGPSData();

        gps_endtime = NOW;

        return;

    }
    digitalWrite(LED_BUILTIN, LOW);


    if (NOW - vesc_endtime > VESC_DELAY) {
        getVESCData();
        vesc_endtime = NOW;
        return;
    }
    if (NOW - temp_endtime > TEMPERATURE_DELAY) {
        getTempData();
        temp_endtime = NOW;
        return;
    }
    if (NOW - post_endtime > POST_DELAY) {
        doc["epoch"] = nb.getLocalTime();
        if (postData(doc, datapath)) {
            Serial.println("data POST");
        } else {
            Serial.println("data not posted...");
        }
        post_endtime = NOW;
        return;
    }
    digitalWrite(LED_BUILTIN, HIGH);
}
```

GET function used to deliver data to the frontend

```
def get(self):

      scooter_data = Controller.get_latest_data()

      scooter_data = scooter_data.to_json()

      scooter_json = json.loads(scooter_data)

      scooter_json.pop("_id")

      return make_response(scooter_json, 200)
```

The controller data model for the incoming data from the electric scooter

```
class Controller(Document):


      location = GeoPointField(required=True)

      temps = EmbeddedDocumentField(Temperatures, default=Temperatures)

      avg_motorcurrent = DecimalField()

      avg_inputcurrent = DecimalField()

      input_voltage = DecimalField()

      rpm = DecimalField()

      tachometer = DecimalField()

      scooter_time = DateTimeField()

      created_at = DateTimeField()

      updated_at = DateTimeField()
```

Configuration of NGINX after the DigitalOcean tutorial

```
# cd /etc/nginx
# cd sites-available
//Remove everything in this folder
# cd ./sites-enabled
//Remove everything in this folder
# cd ..
# sudo nano nginx.conf


http {
      upstream flask {
            server unix:///tmp/escooter_api.sock;
      }
      server {
            listen 80;
            server_name *.jesse.plus www.jesse.plus;

            location / {
                  include uwsgi_params;
                  uwsgi_pass unix:///tmp/escooter_api.sock;
                  uwsgi_ignore_client_abort_on;
            }
      }
}
```