Examensarbete, Högskolan på Åland, Utbildningsprogrammet för Informationsteknik

DESIGN OCH IMPLEMENTERING AV ETT FJÄRRVIKTMÄTNINGSSYSTEM

Anton Wärnström

2023:29

Datum för godkännande: 13.06.2023
Handledare: Kim Gylling

EXAMENSARBETE
Högskolan på Åland

Utbildningsprogram:	Informationsteknik
Författare:	Anton Wärnström
Arbetets namn:	Design och implementering av ett fjärrviktmätningssystem
Handledare:	Kim Gylling
Uppdragsgivare:	Anton Wärnström

Abstrakt

Till viltvård hör förutom jakt även att mata djuren på vintern (såsom älg, rådjur och vitsvanshjort). Matningsautomaten placeras ute i naturen, och djuren kan lätt och tryggt komma till den och äta. Oftast placeras här även viltkameror för att se vilka olika djur som rör sig i trakten. Tunnorna innehåller ca 200 kg till exempel vete eller havre. För att jägarna skall veta hur mycket mat det finns kvar i tunnorna och när de behöver fyllas på, blir de tvungna att köra till matningsstället för att se i tunnan. För att kunna minska körandet ville jag därför utveckla ett trådlöst system för att kunna avläsa tyngden på tunnan.
Syftet med det här examensarbetet är att skapa en design- och implementera ett trådlöst viktövervakningssystem. Målet med systemet är att kunna övervaka viktdatan i realtid och analysera viktdatan över en tidsperiod.
Resultatet kommer att placeras under mattunnan och kommer att användas i jaktlaget i höst.

Nyckelord (sökord)

IoT, Arduino, Viktmätning

Högskolans serienummer:	ISSN:	Språk:	Sidantal:
2023:29	1458-1531	Svenska	29

Inlämningsdatum:	Presentationsdatum:	Datum för godkännande:
03.05.2023	17.05.2023	13.06.2023

DEGREE THESIS
Åland University of Applied Sciences

Degree Programme:	Information Technology
Author:	Anton Wärnström
Title:	Design and implementation of a remote weight measuring system
Academic Supervisor:	Kim Gylling
Commissioned by:	Anton Wärnström

Abstract

In wildlife management, in addition to hunting, feeding the animals (such as moose, roe deer, and white-tailed deer) in the winter is included. The feeding station is placed in nature where the animals can easily and safely come to eat. Often, wildlife cameras are also placed there to observe the different animals in the area. The barrels contain approximately 200 kg of, for example, wheat or oats. In order for the hunters to know how much food is left in the barrels and when they need to be refilled, they have to drive to the feeding site to check the barrel. To reduce the need for driving, I wanted to develop a wireless system to monitor the weight of the barrel. The purpose of this thesis project is to design and implement a wireless weight monitoring system. The goal of the system is to monitor weight data in real-time and analyze weight data over a period of time. The resulting system will be placed beneath the feeding barrel and will be used by the hunting team in the fall.

Keywords

IoT, Arduino, Weight measurement

Serial number:	ISSN:	Language:	Number of pages:
2023:29	1458-1531	Swedish	29

Handed in:	Date of presentation:	Approved:
03.05.2023	17.05.2023	13.06.2023

INNEHÅLLSFÖRTECKNING

1. INTRODUKTION

1.1 Bakgrund

Jakt och viltvård har alltid varit en stor del av mig ända sedan jag var liten. När jag var liten fick jag alltid följa med min morfar på jakt och hjälpa med påfyllning av matningsautomaterna för djur. Nuförtiden håller jag ännu på med jakt samt viltvård och är dessutom med i ett jaktlag som utöver jakt också upprätthåller och matar vilt med hjälp av matningsautomater. Detta betyder att man utöver transport av mat för påfyllning måste hålla koll på hur mycket mat det finns kvar i automaten, vilket leder till en stor del transport med bil. Genom åren har jag märkt att det under jaktsäsongen blir en stor mängd "onödigt" bilkörande eftersom man flera gånger under veckan måste följa med matmängden. Eftersom jag har ett stort intresse för att skapa praktiska system och hjälpmedel som underlättar vardagen och ett intresse för IT så har detta lett mig till att planera och skapa en fjärrsystems prototyp för att övervaka och följa med mängden i automaterna med hjälp av IoT (eng. *Internet of Things)* och ett webbgränssnitt. Det här underlättar övervakningen och minimerar mängden körningar som behövs. Jag valde detta projekt på grund av mitt stora intresse för IoT och att skapa små system med hjälp av mikrokontrollerkort.

1.2 Syfte

Syftet med detta examensarbete är att utveckla ett system som möjliggör att man enkelt kan övervaka och följa med vikten på en matningsautomat för vilt från ett webbgränssnitt. Jag vill också få en bättre förståelse för utveckling av IoT-system och hur man kopplar ihop olika delsystem för att skapa ett fungerande system.

Jag undersöker även hur man ansluter ett Arduino mikrokontrollerkort till en webbserver via mobila GSM-nätverket (eng. *Global System for Mobile Communications*) och MQTT-protokollet (eng. *Message Queuing Telemetry Transport*) med hjälp av en MQTT-mäklare (eng. *MQTT-broker*), sparar data till en databas och slutligen visar mätdatan på ett webbgränssnitt med grafer. Jag kommer i det här arbetet att använda mig av en produkt från företaget EMQ och jag använder mig av deras MQTT-mäklarprodukt EMQX.

1.3 Metod

Eftersom jag inte hade en extern uppdragsgivare så fick jag själv bestämma samt prioritera vad som skulle utvecklas i vilken ordning för prototypen. Jag prioriterade fjärrviktmätningssystemet över de andra systemen eftersom det är det största systemet och jag har minst erfarenhet av hur jag skall utveckla det, vilket betyder att det kommer att ta mest tid för undersökning och implementering.

Fjärrviktmätningssystemet är utvecklat i två iterationer. Första iterationen är den största och viktigaste eftersom den går ut på att undersöka hur olika elektroniska komponenter fungerar tillsammans och att få en fungerande bas-prototyp som kan användas vid testning av anslutning och dataöverföring till webbservern. Andra iterationen är lite mindre och går ut på att förbereda systemet för “live” -testning genom att uppdatera material och vädersäkra systemet.

Webbservern och webbgränssnittet utvecklades på sidan av fjärrviktmätningsystemets utveckling. Jag valde att göra det på det här sättet eftersom jag märkte att det kommer att underlätta utvecklingen av webbservern och webbgränssnittet om jag kan använda mig av riktigt data från systemet för att förstå hur dataöverföringen fungerar och identifiera samt åtgärda problem som kanske inte skulle uppstå om jag använde mig av färdig bestämd testdata.

1.4 Avgränsningar

Det här arbetet kommer att användas för privat bruk, vilket betyder att designen på viktmätningssystemet inte kommer att ta i beaktande uthållighet och väder som till exempel hård köld och konstruktion av själva matningsautomaten kommer inte att ingå i mitt arbete, istället fokuserar jag på att få ihop en fungerande prototyp.

Examensarbetet kommer att vara avgränsat till att få en fungerande prototyp för ett viktmätningssyste och anslutning mellan webbservern och webbgränssnittet. Det här betyder

också att webbgränssnittet och webbservern inte från början kommer att vara designade att vara skalbara eller produktionsfärdiga.

2. Definitioner

2.1 Internet of Things (IoT)

IoT-system är en väldigt innovativ teknik som används överallt inom IT-världen, det är i stort sett ett begrepp som beskriver hur man genom att koppla ihop enheter, sensorer och system till nätverkssystem gör det möjligt för fjärrstyrning och automatisering. IoT-system kan användas inom många olika områden som till exempel industri, sjukvård, transport och smarta hem för att göra livet lättare för personer med funktionsnedsättningar. Fördelarna med IoT är att det är möjligt att samla in och analysera data i realtid och systemen är oftast relativt små och kräver en liten mängd resurser. Att börja utveckla IoT-system är dessutom relativt billigt på grund av att det finns en stor marknad för mikrokontrollerkort, sensorer och andra elektroniska komponenter, vilket gör att de massproduceras. (*What Is the Internet of Things - an Explainer | World Economic Forum*, 2021)

2.2 Arduino

Arduino (*Arduino Reference*, n.d.) är ett företag som skapar mikrokontrollerkort baserat på en öppen hårdvaruplattform. De kan programmeras för att styra elektroniska enheter och det är enkelt att integrera en stor mängd olika sensorer och andra hårdvarukomponenter med en liten erfarenhet av elektronik och programmering. Arduino har också en stor “community” som delar med sig av sina projekt och sin kod. Eftersom Arduino bara är ett mikrokontrollerkort så passar det bäst i mindre projekt där den inte behöver hantera komplexa uppgifter. Det finns flera områden som Arduino används till, allt från hobbyprojekt till professionella projekt inom industri.

2.3 MQTT

MQTT (eng. *Message Queuing Telemetry Transport*) är ett meddelandeprotokoll som används inom IoT-anslutningar. Det är ett protokoll som är enkelt och lättviktigt, designat för begränsade enheter och nätverk med låg bandbredd, hög latens eller opålitlighet. (*FAQ | MQTT*, n.d.). Skillnaden mellan HTTP-protokollet (eng. *Hypertext Transfer Protocol*) och MQTT är att HTTP är ett textbaserat förfrågan- och svarsprotokoll (eng. *request and response*) som vanligtvis används för webbsidor eller webbtjänster som kräver mera overhead per förfrågan.

MQTT-protokollet är utvecklat för att samla in data från flera sensorer eller IoT-enheter och att skicka informationen till en server som kan spara och analysera datan. MQTT används i en stor mängd olika applikationer inom IoT genom att en enhet (eng. *publisher*) publicerar meddelanden på ett visst ämne (eng. *topic*) och en eller flera andra enheter, så kallade prenumeranter (eng. *subscribers*), prenumererar på samma ämne för att få meddelanden som publiceras på det ämnet. System som det ofta används för, är till exempel automation i hemmet, industriautomation, logistik, transport och fjärrövervakning.

En nackdel med MQTT är att det inte har inbyggd säkerhet, vilket kan vara ett problem när man hanterar känslig data. För att lösa detta problem måste man använda sig av andra säkerhetsåtgärder, till exempel ett SSL (eng. *Secure Sockets Layer*) eller TLS (eng. *Transport Layer Security*) krypteringsprotokoll som används för att kryptera data som skickas över ett nätverk.

2.4 EMQX

EMQX är en tjänst utvecklat av företaget EMQ. Det är en IoT MQTT-mäklare som erbjuder funktionaliteten att enkelt samla och skicka IoT-data mellan en stor mängd enheter. Med den infrastruktur som EMQX har så stöder de flera länder och regioner runt omkring i världen med billiga, säkra och pålitliga molntjänster för 5G- och IoT-applikationer. (*Product Overview | EMQX Cloud Documentation*, n.d.)

3. SYSTEMSTRUKTUR OCH DESIGN

3.1 Strukturering av projektet

Projektet är uppdelat i tre huvudsakliga delsystem som arbetar tillsammans: 1. en IoT-enhet, 2. en webbserver och 3. ett webbgränssnitt.

IoT-systemet ansvarar för att samla in viktdata om matningsautomaten med hjälp av en lastcell som är konstruerad med trådtöjningsgivare för att mäta krafter. Systemet skickar data varje timme till webbservern via GSM-nätverket och använder sig av MQTT-protokollet. Tack vare EMQX är det möjligt att skapa en snabb och enkel dataöverföring och fungera som en bro mellan IoT-systemet och webbservern.

Webbservern är den centrala delen av systemet och ansvarar över all data som kommer in från IoT-enheten och lagrar datan i en MongoDB-databas. Webbservern är också ansvarig för att hantera alla interaktioner från användare som kommer via webbgränssnittet. Webbgränssnittet är webbapplikationen som använder sig av datan som samlats in från IoT-enheten genom att skicka en förfrågan av datan från webbservern. Webbapplikationen visar sedan datan på grafer över vissa perioder.

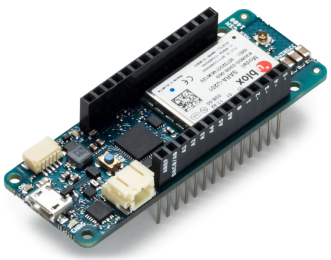
3.2 IoT-systemet

Det här IoT-systemet är uppbyggt av tre huvudkomponenter, ett mikrokontrollerkort, en signalförstärkare och en lastcell. Systemet behöver också en spänningskälla och en antenn (som hör till mikrokontrollerkortet).

3.2.1 Mikrokontrollern

För det här systemet valde jag att använda mig av Arduino MKR GSM 1400 (*fig. 1*). Jag valde den här mikrokontrollern eftersom den är liten och lätt, vilket gör att det är enkelt att integrera i olika typer av projekt. Storleken på kortet är 67,6 mm × 23 mm. Den är designad för att ansluta till GSM-nätverket och har en inbyggd SIM-kortplats samt en anslutning för en antenn, vilket eliminerar behovet för extra hårdvara men den är också kompatibel med olika sensorer och aktuatorer. Dessutom stödjer den olika kommunikationsprotokoll som TCP, HTTP, MQTT och SSL. Mikrokontrollern är lätt programmerbar med hjälp av Arduinos IDE eftersom den har tillgång till en stor mängd inbyggda bibliotek vilket underlättar utvecklingen. Det är även möjligt att automatiskt ladda ner ett tredjepartsbibliotek direkt från användargränssnittet.

Mikrokontrollern är också designad för att ha en låg energikonsumtion och har egenskapen att programmera så att den går i ett lågenergiläge (eng. *low power mode)*, allt det här gör att det passar perfekt för mitt behov eftersom systemet skall sitta nästan orört i flera månader. Mikrokontrollern har en spänningsutgång på 5 volt, vilket räcker för att driva både lastcellen och signalförstärkaren.



Figur 1. Bild på Arduino MKR GSM 1400
(*Arduino MKR GSM 1400 — Arduino Official Store*, n.d.)

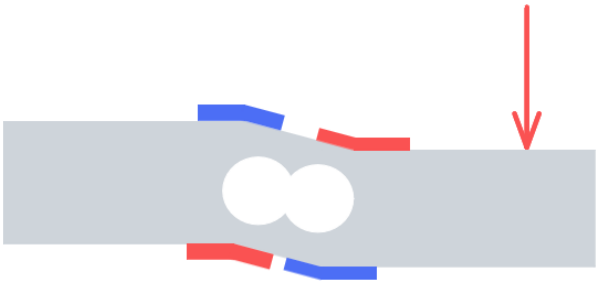
3.2.2 Lastcellen

För att kunna mäta vikten av matningsautomaten bestämde jag att använda en H30A-C3 (*fig. 2*) enpunkts lastcell. Det är en unik modell av lastcell som klarar av att mäta vikten med hög noggrannhet även om vikten inte sitter exakt på belastningspunkten. Här är lite detaljer om lastcellen:

- Material: Aluminium
- Maxkapacitet: 15-300 kg
- Noggrannhetsklass: C3, Y=7.500
- Skyddsklass: IP 65
- Design: Mätelementet är inkapslat

(*Single Point Load Cell H30A | BOSCHE*, n.d.)

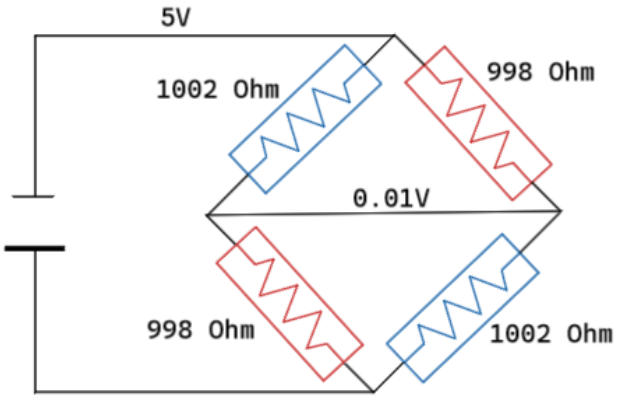
Figur 2. Bild på H30A-C3 lastcellen



Figur 3.Överdriven bild på hur töjningsmätarna deformeras när det appliceras en kraft på lastcellen

Lastcellen fungerar genom att ha en krets med fyra töjningsmätare som mäter töjningsdeformationen i materialet efter applicerad kraft som omvandlar deformationen till en elektrisk signal som kan mätas. (*fig. 3*)

De fyra töjningsmätarna är parallellkopplade i en så kallad diamantformad brokrets och den här tekniken kallas för en Wheatstone-bro (*fig. 4*) som används för att mäta små förändringar över motstånd.

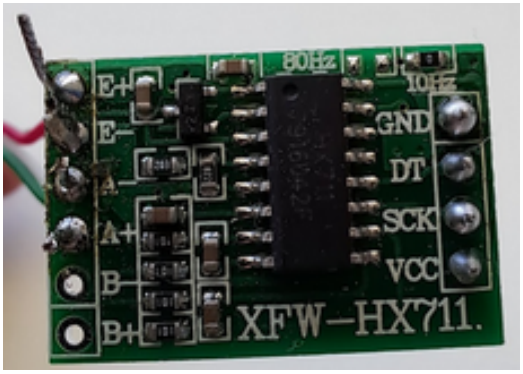


Figur 4. Elschema över hur en Wheatstone bridge fungerar

Det är möjligt att mäta spänningen över töjningsmätarna med Wheatstone-bron genom att mata en stadig elektrisk spänning till kretsen och skapa belastning på lastcellen genom att applicera kraft så att det blir en deformation i materialet. Varje deformation i materialet skapar en förändring i motstånden vilket gör att brokretsen inte längre är i balans och spänningen över motstånden förändras. Om kretsen är balanserad och lastcellen inte har någon kraft applicerad på den, så är spänningen över motståndet noll.

3.2.3 HX711

Den signal som lastcellen ger ut är väldigt liten och svår att jobba med så den måste förstärkas med en signalförstärkare för att den skall vara användningsbar. Signalförstärkaren jag valde för det här är en 24-bit analog-till-digital konverterare HX711(*fig. 5*), orsaken varför jag valde den här signalförstärkaren är för att det är en billig och pålitlig komponent, vilket passar utmärkt för det här systemet och det krävs inte någon extra programmering eller konfigurering på de interna registerna för att få den att fungera.



Figur 5. Bild på signalförstärkaren HX711

3.3 Webbgränssnittet

Webbgränssnittet är en webbapplikation som använder sig av datan som samlats in från IoT-enheten. Webbapplikationen hämtar datan från webbservern och visar sedan datan på grafer över vissa perioder och visar positionen på sensorerna på en interaktiv karta. Applikationen har utvecklats med hjälp av Javascript webbramverket Vue.js, kart API Mapbox och grafbibliotek Chart.js.

3.3.1 Vue.js

Vue.js är ett av flera moderna Javascript-baserade webbramverk som används för att utveckla webbapplikationer. Det är ett öppen källkodsprojekt som underhålls och utvecklas aktivt av ett team med både heltids- och frivilliga medlemmar. Vue.js erbjuder funktionalitet för att skapa komplexa applikationer som till exempel, routing, reaktivitet och state management. (*Vue.js*, n.d.)

3.3.2 Mapbox

Mapbox är ett företag som erbjuder flera tjänster relaterade till navigering och kartor. Jag använder mig av tjänsten Mapbox GL JS vilket är ett Javascript-bibliotek för att skapa realtids- och interaktiva kartor på webben. De erbjuder en tjänst där man kan editera och anpassa utseendet av en karta så att det passar det utseendet kunden vill ha.

Mapbox erbjuder en gratis plan med en gräns på totalt 50 000 hämtningar av kartan per månad. Vårt jaktlag består av 10 personer som kommer att använda systemet, vilket betyder

att varje person kan hämta kartan 5 000 gånger per månad. (*Mapbox*, n.d.) För mitt arbete kommer jag att skapa en karta för att visa positionen på matningsstationerna.

3.3.3 Chart.js

Chart.js är ett Javascript-bibliotek med öppen källkod som erbjuder funktionalitet för att enkelt kunna skapa interaktiva samt anpassbara grafer för användaren. I mitt arbete kommer jag att använda mig av två stapeldiagram som visar viktdata mellan vissa tidsperioder. (*Chart.js*, n.d.)

3.4 Webbservern

Webbservern är en implementering av ett applikationsprogrammeringsgränssnitt (eng. *application programming interface*) som utgör ett antal anslutningspunkter (eng. *endpoints*) och som hanterar datan med hjälp av en databas. För den här uppgiften bestämde jag mig för att skriva webbservern med programmeringsspråket Go (*Go*, n.d.) och hantera datalagring med dokumentdatabasen MongoDB och deras gratis MongoDB Atlas-klustertjänst. Go är ett programmeringsspråk utvecklat av Google, jag har redan länge varit intresserad av Go och velat förbättra mina kunskaper inom Go och ett par orsaker till detta är att det är ett modernt språk som kom ut året 2009, så det är utvecklat för att följa moderna programmeringskonventioner, och det har en väldigt enkel syntax jämfört med andra språk. Det här gör att det är enkelt att komma igång att skriva samt underlättar det att underhålla existerande kod.

För implementationen av webbservern använde jag ett antal olika Go tredjepartsbibliotek:

- github.com/eclipse/paho.mqtt.golang är ett öppen källkods bibliotek som erbjuder en enkel och effektiv implementering av MQTT-protokollet för Go-programmeringsspråket. Biblioteket är en del av Eclipse Paho-projektet. (*Eclipse Paho | The Eclipse Foundation*, n.d.)
- gin-gonic/gin är det officiella biblioteket för webbramverket Gin som är lättviktigt, snabbt och enkelt att använda. Det är byggt med fokus på flexibilitet och för att vara enkelt att använda för små projekt samt stora projekt med hög trafikbelastning. Gin har ett stort antal inbyggda funktioner och verktyg för att underlätta utvecklingen, som till exempel: middleware, routning, loggning och hantering av HTTP-felkoder.

Och eftersom det är öppen källkod så har det en stor community som hjälper till att underhålla det. (*Gin Web Framework*, n.d.)

- go.mongodb.org/mongo-driver är en officiell MongoDB-drivrutin för Go-språket som gör det möjligt att kommunicera mellan en Go-applikation och MongoDB-databasen. Drivrutinen är utvecklad av MongoDB och är en del av det större MongoDB-ekosystemet. Drivrutinen har utvecklats för att vara enkel att använda och den stöder alla grundläggande funktioner som CRUD-operationer (*Create, Read, Update och Delete*), indexering, aggregation, geospatiala sökningar, transaktioner och även möjligheten att arbeta med BSON-dokument. (*Binary Javascript Object Notation*) (*MongoDB Go Driver — Go*, n.d.)

Webbservern är implementerad för att motta data från MQTT-mäklaren, analysera samt spara datan i en databas. Figur 5 visar visuellt dataöverföringen mellan alla systemen.

Figur 5. Ett förenklat exempel hur mikrokontrollern skickar data via MQTT mäklaren till webbservern

4. SYSTEMIMPLEMENTERING

4.1 IoT-systemet

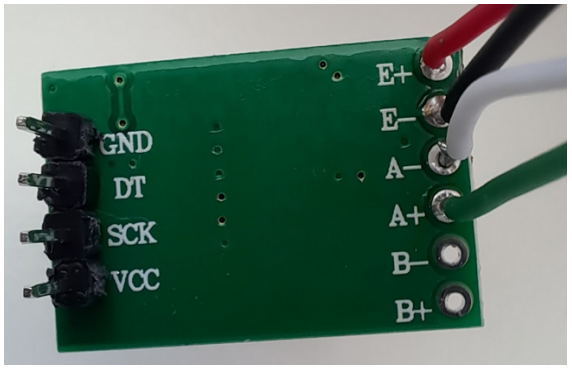
Det här löste jag genom att köpa en billig mindre lastcell med maxtyngd på 5 kg från en elektronikkomponentbutik. Samtidigt som jag köpte lastcellen köpte jag också mikrokontrollern och signalförstärkaren.

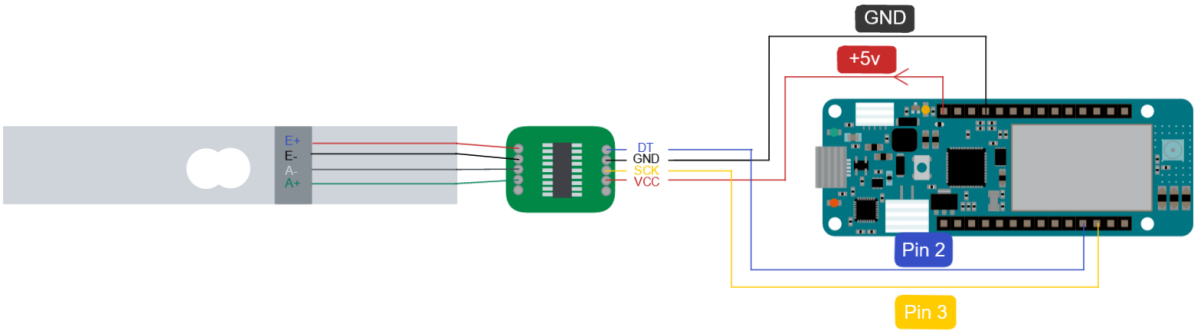
4.1.1 Fysisk prototypdesign

Nästa steg var att koppla ihop komponenterna. För det här behövde jag löda fast lastcellernas kablar till signalförstärkaren och koppla kablar mellan mikrokontrollern och signalförstärkaren. Efter ett sökande online för dokumentering på kopplingsschema för lastcellen och signalförstärkaren (*tabell 1*), hittade jag en PDF (*24-Bit Analog-To-Digital Converter (ADC) for Weigh Scales*, n.d.) för signalförstärkaren och en handledningsartikel

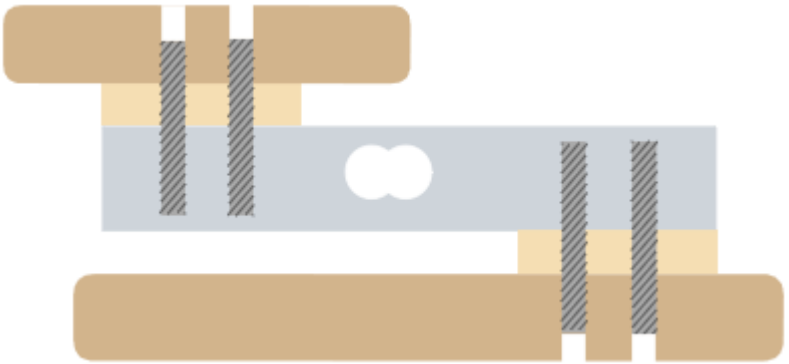
om lastcellens kablar. Tabell 1 nedan visar färgkoderna för kopplingsschema för båda komponenterna.

Tabell 1. Visar lastcellens färgkodade kablar respektive signalförstärkarens anslutningar

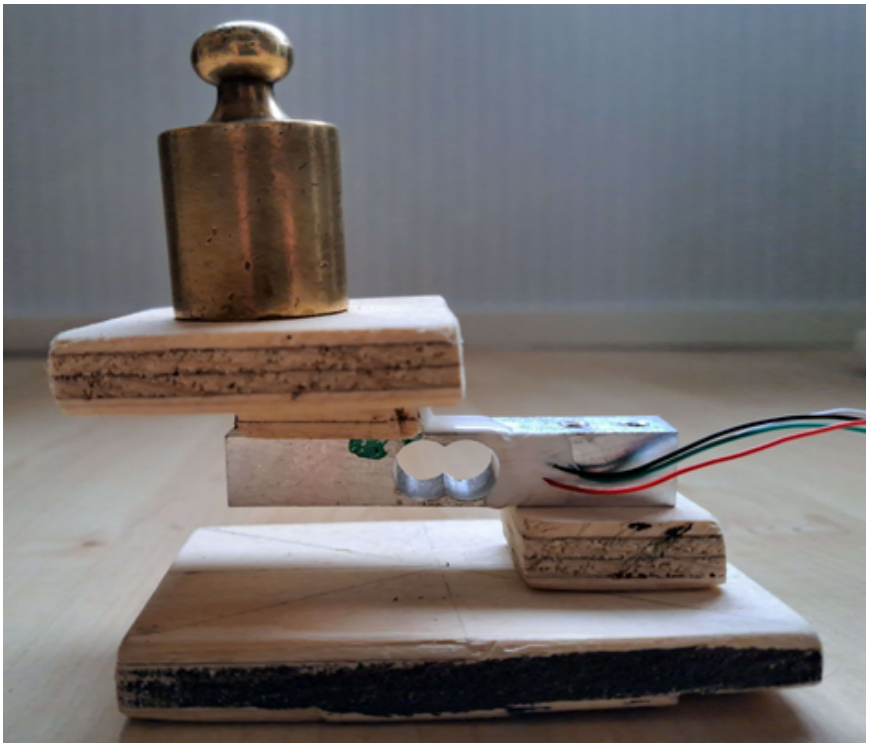
	***Lastcell***	***HX711***	***HX711***
	Röd (E+)	E+	GND
	Svart (E-)	E-	DT
	Grön (A+)	A+	VCC
	Vit (A-)	A-	SCK



Figur 6. Kopplingsschema mellan lastcell, signalförstärkaren och mikrokontrollern



Figur 7. Ritning för första prototypen av mätningssystemet med lastcellen



Figur 8. Första ihopsatta prototypen med en 200 g testvikt

Efter att jag kopplat ihop komponenterna (*Fig. 6*) behövde jag ännu bygga ett fotstöd samt en platta för vikten att sitta på. Det här gjorde jag genom att borra hål i två träskivor och skruva fast dem i lastcellen (*Fig. 7, Fig. 8*). Efter att viktmätningssystemet var ihopsatt var det dags att skriva koden för systemet.

4.1.2 Mjukvaruimplementering

Arduinos program kallas för en sketch och de är skrivna med Arduino programmeringsspråket som egentligen är C++. Varje program har en basstruktur med två funktioner: *setup()* och *loop()* som rekommenderas att användas.

setup() är en funktion som påminner om programmeringsspråket Gos eller Cs main() funktion där den är den första funktionen som körs i programmet. Meningen med denna funktion är att kunna initialisera variabler, konstanter, pinMode eller läsa in externa bibliotek.

loop() är en funktion som körs kontinuerligt och den anropas genast efter att *setup()* är klar. Meningen med den här funktionen är att kunna kontrollera mikrokontrollern och låta programmet göra ändringar och köra nya funktioner.

I mitt program använder jag *setup()* (*Figur 9*) för att ansluta till MQTT-mäklaren med *client.begin()* och sedan anropar den en *connect()* funktion som jag skrivit för att ansluta till GSM-nätverket och till slut anropar *setup()* funktionen *calibrateLoadcell()*.

```
void setup() {
  client.begin("broker.emqx.io", gsmClient);
  connect();
  calibrateLoadcell();
}
```

Figur 9. Setup funktion som ansluter till GSM nätverket samt anropar lastcell kalibrering

För att underlätta kodningen så använder jag mig av ett tredjepartsbibliotek (*Bogde/HX711: HX711 24-Bit Analog-To-Digital Converter (ADC) for Weight Scales.*, n.d.) för att ansluta lastcellen och signalförstärkaren med mikrokontrollern. *calibrateLoadcell()* funktionen börjar med att definiera två konstanta variabler, *LOADCELL_DOUT_PIN* och *LOADCELL_SCK_PIN*. De här variablerna används för att ansluta lastcellen och signalförstärkaren med mikrokontrollern.

```
HX711 scale;
const int LOADCELL_DOUT_PIN = 2;
const int LOADCELL_SCK_PIN = 3;
void calibrateLoadcell() {
  Serial.begin(38400);
  scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);
  if (scale.is_ready()) {
    scale.set_scale();
    scale.tare();
    delay(5000);
    long reading = scale.get_units(10);
    long calibration_factor = reading / 200;
    scale.set_scale(calibration_factor);
    Serial.println("Loadcell has been calibrated");
  } else {
    Serial.println("HX711 not found.");
  }
}
```

Figur 10. Kalibrering av lastcellen

Till följande initialiserar koden lastcellen och signalförstärkaren genom att anropa funktionen *scale.begin()* (*Figur 10*) med att skicka in de två definierade konstanterna *LOADCELL_DOUT_PIN* och *LOADCELL_SCK_PIN*. Om både lastcellen och signalförstärkaren är redo, fortsätter funktionen med kalibrering av lastcellen genom att anropa funktionerna *scale.set_scale()* och *scale.tare()*. Efter det väntar programmet fem sekunder. Under den tiden ska man sätta en känd vikt på vågen. Efter det kalibrera lastcellen och det här görs med en känd vikt och en kalibreringsfaktor. Den räknas ut med hjälp av formeln:

$$kaliberingsfaktor \ = \ (värde) \ / \ (vikt)$$

V*ärdet* är resultatet man får när man anropar *scale.get_units(10)* och *vikt* är en vikt man vet vikten på i gram. Med kalibreringsfaktorn kan man kalibrera lastcellen och signalförstärkaren med *scale.set_scale()*.

I mitt program utför *loop()* (*Fig. 11*) funktionen två huvudsakliga uppgifter: 1. upprätthålla anslutningen till MQTT-mäklaren genom att anropa *client.loop()* och 2. publicera data till servern varje timme. Om mikrokontrollern inte är ansluten anropas funktionen *connect()*. Före jag publicerar meddelandet anropas *getLoadcellMeasureValue()* för att hämta aktuella

viktvärden från lastcellen. Till följande skapar den en textsträng som innehåller tidsstämpeln från enhetens lokala klocka och de hämtade värdena från lastcellen. Därefter publicerar funktionen meddelandet till MQTT-ämnet */feedingstations/station_one/* med *client.publish()* funktionen.

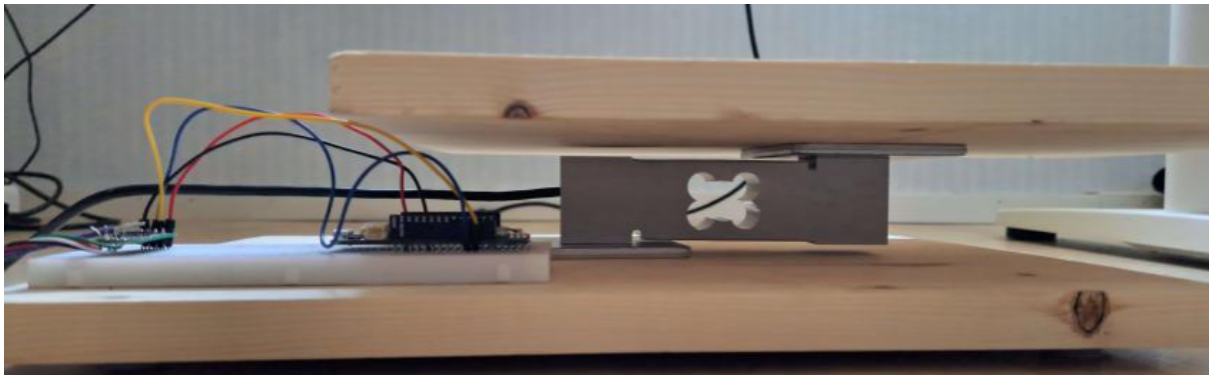
```
void loop() {
  client.loop();
  if (!client.connected()) {
    connect();
  } else {
    if (millis() - mqttPublishTimer > interval_ms) {
      String measureValue = getLoadcellMeasureValue();
      String payload = String(gsmAccess.getLocalTime()) + "-" + measureValue);
      mqttPublishTimer = millis();
      delay(1000);
      client.publish("/feedingstations/station_one", payload);
    }
  }
}
```

Figur 11. loop-funktionen som upprätthåller anslutningen samt publicerar data till MQTT-mäklaren

Det finns också ett konstant värde specificerat i millisekunder som bestämmer hur ofta data skall publiceras. Funktionen använder *millis()* funktionen för att kolla millisekunderna sedan programmet börjat. Om tillräckligt med tid har gått, kommer funktionen att fortsätta med att skapa och skicka meddelanden till servern. Det här avslutar första iterationen.

Andra iterationen är en mindre iteration där jag sätter mer fokus på att förbättra byggkvaliteten och förnya lastcellen. I den här iterationen bytte jag ut den första lastcellen med maxvikt på 5 kg till H30A-C3 lastcellen som jag skrev om tidigare i arbetet. Modellen jag valde har en maxvikt på 200 kg. Eftersom den nya lastcellen kommer att vara större och klara av högre vikter så behöver vågens storlek ändras. Enligt företaget som tillverkar lastcellen ska den klara av att mäta vikten med en hög noggrannhet inom ett område på 500 x 600 mm. Det är de dimensionerna jag använder mig av.

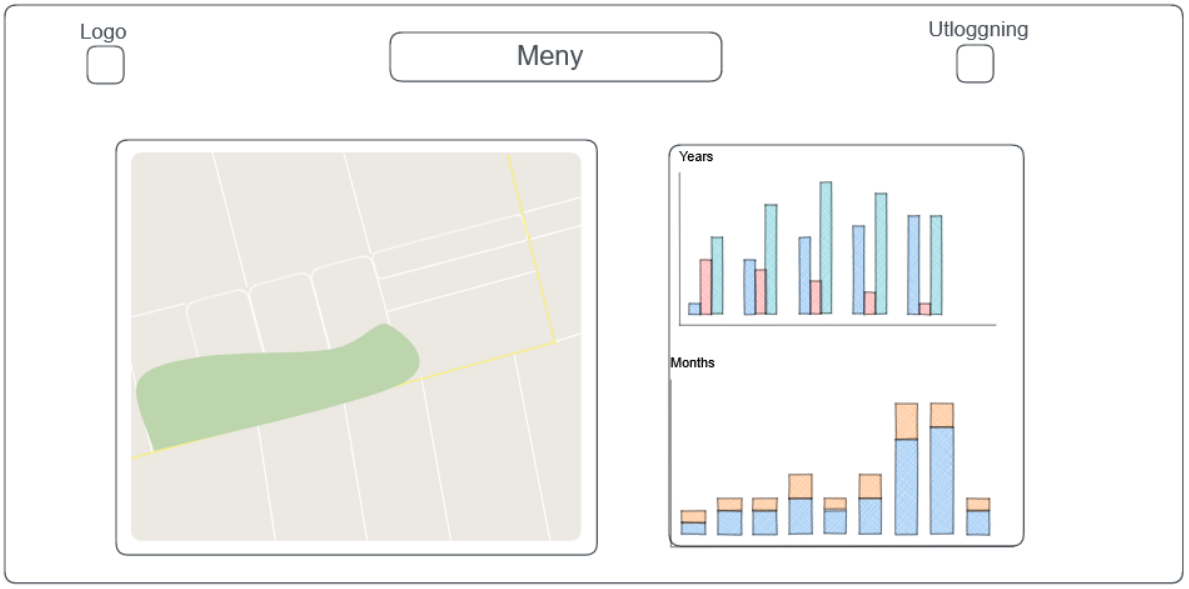
Figur 12. Andra iterationen på vågprototypen uppifrån för helhetsbild



Figur 13. Andra iterationen på vågprototypen från sidovinkel för att se lastcellen

4.2 Webbgränssnittet

Utvecklingen av webbgränssnittet började med planering för vad jag ville att man skulle kunna göra på webbgränssnittet. Relativt snabbt bestämde jag mig för att jag ville visa en karta med positionen på systemet, samt att ha grafer för att kunna följa med hur mycket djuren äter under en vecka och genomsnitt per månad på ett år. Efter att jag bestämt idéerna som jag vill implementera, började jag med planeringen av layouten. Nedan är en representation av planeringen av layouten på webbgränssnittet.



Figur 14. Layout för webbgränssnittet

För implementationen valde jag att använda mig av de här verktygen:

- Webbramverk: Vue.js (*Vue.js*, n.d.)
- Kart-API: (*Mapbox*, n.d.)
- Graf-API: Chart.js (*Chart.js*, n.d.)

Figur 15. Tydligare figur på hur komponenterna är uppdelade i layouten

Jag började med att dela upp layouten i olika komponenter: *dashboard.vue, Mapbox.vue* och *Graphs.vue. Dashboard.vue* är en så kallad parent-komponent där det finns en eller flera integrerade child-komponenter. *Mapbox.vue* och *Graphs.vue* fungerar som child-komponenter, se *Fig. 15* för en tydligare visualisering hur uppdelningen mellan komponenterna fungerar. *Dashboard.vue* komponenten innehåller också basimplementationen och layouten för sidan.

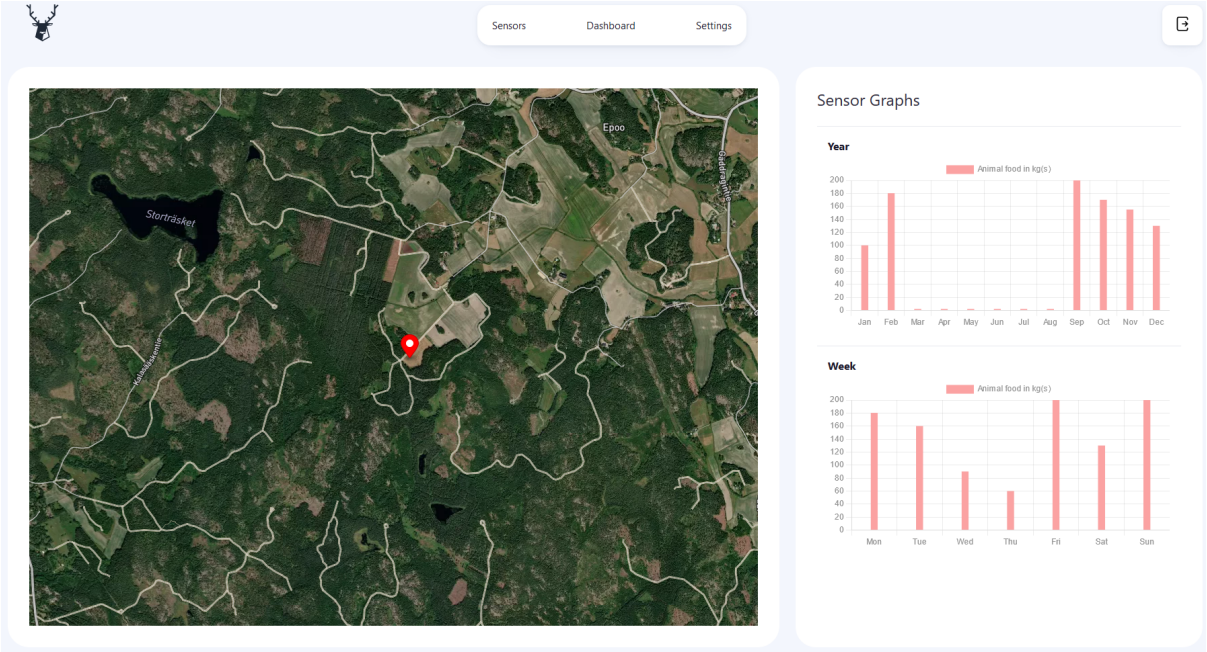
Efter att basstrukturen för layouten är implementerad så implementerar jag kartan. För det skapar jag ett konto på Mapbox: Med det här kontot får jag åtkomst till Mapbox API:et genom att använda en åtkomsttoken (*eng.* accessToken), sedan importerar jag Javascript biblioteket Mapbox-gl för att enkelt kunna skapa en interaktiv karta (*Fig. 16*).

```
const map = ref<mapboxgl.Map>();

map.value = new mapboxgl.Map({
        accessToken: "",
        container: "mapContainer",
        style: "mapbox://styles/warnstrom/cle8kpjzh008q01qm0p55d17e",
        center: [25.8099993, 60.33175],
        zoom: 13,
});
```

Figur 16. Skapande av en karta med hjälp av Mapbox-gl biblioteket

Graferna jag använder är gjorda med hjälp av ett tredjepartsbibliotek som heter *Chart.js*. Med det här biblioteket är det möjligt att enkelt skapa diagram som är responsiva och dynamiska. Nedan finns en skärmdump på första iterationen av webbgränssnittet.



Figur 17. Skärmdump av webbgränssnittet

4.3 Webbservern

Utvecklingen av webbservern började med att planera vilka ändpunkter webbservern behövde och hur jag implementerar MQTT-protokollet i Go med Eclipse Paho-biblioteket. Jag har redan erfarenhet från tidigare projekt hur jag använder MongoDB-drivrutiner för att ansluta till MongoDB och att hantera dataöverföringen till databasen och HTTP-förfrågningar med Gin-Gonic biblioteket.

Webbserverns ändpunkt är en GET-förfrågning som hämtar alla mätningar som sparats från lastcellen. I senare iterationer kommer jag att implementera förfrågningar för hämtning av mätdata mellan två datum. Dessutom planerar jag att implementera ett inloggningssystem för jaktlaget så de kan logga in med egna användarkonton.

Dokumenteringen för MQTT Eclipse Paho-biblioteket var ganska svår att förstå eftersom det inte fanns ordentliga exempel eller instruktioner. Jag börjar med att skapa en anslutning, se

Figur 18, till en MQTT-mäklare och returnerar ett MQTT-klientobjekt som kan användas för att prenumerera på ett ämne. Koden använder sig av xorshift (*Xorshift*, n.d.), se Figur 19, som är en funktion för pseudo-slumptalsgenerering för att generera en unik id för klienten. I slutet av funktionen så försöker klienten ansluta till en MQTT-mäklare med *token := client.Connect()* som returnerar en token för att kontrollera om anslutningen var lyckad eller inte.

```
func createMqttClient() MQTT.Client {
    state := xorshift64State{seed: 351293764872}
    opts := MQTT.NewClientOptions()
    opts.AddBroker("broker.emqx.io:1883")
    opts.SetUsername("")
    opts.SetPassword("")
    opts.SetClientID(string(rune(xorshift64(&state))))
    opts.SetKeepAlive(240)
    opts.OnConnectionLost = connectLostHandler
    client := MQTT.NewClient(opts)
    token := client.Connect()
    if token.WaitTimeout(20*time.Second) && token.Error() != nil {
        log.Fatal(token.Error())
    }
    return client
}
```

Figur 18. createMQTTClient skapar ett MQTT-klientobjekt

```
type xorshift64State struct {
    a uint64
}
func xorshift64(state *xorshift64State) uint64 {
    var x uint64 = state.a
    x ^= x << 13
    x ^= x >> 7
    x ^= x << 17
    state.a = x
    return x
}
```

Figur 19. xorShift funktion för generering av slumptal

5. DISKUSSIONER

5.1 Resultat

Målet med arbetet var att skapa en fungerande fjärrviktmätningsprototyp som sparar mätningsdatan i en databas och visualiserar datan på ett webbgränssnitt som ger möjligheten för användare att följa med vikten på automaten. Implementationen av de här olika systemen var mer komplicerad än förväntat eftersom det är så många olika system som skall fungera tillsammans. Fastän målet med att skapa en fjärrviktmätningsprotoyp samt visualisering av datan nåddes så finns det ändå delar av systemet som kommer att förbättras i framtiden innan det är en helt ordentligt fungerande produkt som är färdig.

5.2 Planer för framtiden

5.2.1 Fjärrviktmätningsystemet

För framtiden har jag planerat att göra en ny iteration av fjärrmätningssystemet med fokus på att uppdatera systemet så att det tål regn och snö. För den här iterationen tänker jag mig åtminstone två huvuddelar. Det första är materialbyte för plattorna på vågen genom att byta ut träskivorna till antingen behandlat trä, som tål väder, eller metall. Metall medför svårigheter, som till exempel att borra hål eller att få rätt dimensioner på plattan, som kräver att man har rätt verktyg. Andra delar som skall uppdateras är Arduino mikrokontrollern och IoT-systemet. För det här har jag planerat att sätta det i en vattentät plastbehållare.

Jag kommer även att öka spänningskällans kapacitet. Nu används fyra stycken 1,5 V parallellkopplade batterier vilka skall bytas ut till exempel till en 12 V 10 A ackumulator med större kapacitet. Eftersom Arduino använder 5 V, ändrar jag med en spänningsomvandlare spänningen från 12 V till 5 V.

5.2.2 Webbgränssnitt och Webbservern

För webbgränssnittet och webbservern har jag inte planerat några större iterationer. Webbgränssnittet kommer att uppdateras med ett inloggningssystem och framsidan för webbsidan för att förbättra användarvänligheten åt jaktlagsmedlemmarna i framtiden. Dessutom kommer jag att fråga efter feedback av jaktlagsmedlemmarna och göra ändringar efter deras behov.

För webbservern är nästa steg att implementera ett inloggningssystem med autentisering för att undvika onödiga kartladdningar från personer som inte skall använda systemet. Längre in i framtiden kommer jag att förbättra och förnya anslutningspunkterna för lättare hämtning av data.

KÄLLFÖRTECKNING/REFERENCE LIST

About Us | EMQ. (n.d.). EMQ Technologies. Retrieved April 18, 2023, from https://www.emqx.com/en/about

Arduino MKR GSM 1400 — Arduino Official Store. (n.d.). Arduino Store. Retrieved June 9, 2023, from https://store.arduino.cc/products/arduino-mkr-gsm-1400

Arduino Reference. (n.d.). Arduino. Retrieved April 25, 2023, from https://www.arduino.cc/reference/en/

bogde/HX711: An Arduino library to interface the Avia Semiconductor HX711 24-Bit Analog-to-Digital Converter (ADC) for Weight Scales. (n.d.). GitHub. Retrieved April 26, 2023, from https://github.com/bogde/HX711

Chart.js. (n.d.). Chart.js | Open source HTML5 Charts for your website. Retrieved April 28, 2023, from https://www.chartjs.org/

Eclipse Paho | The Eclipse Foundation. (n.d.). Eclipse. Retrieved April 22, 2023, from https://www.eclipse.org/paho/

FAQ | MQTT. (n.d.). MQTT. Retrieved April 17, 2023, from https://mqtt.org/faq/

Gin Web Framework. (n.d.). Gin Web Framework. Retrieved April 22, 2023, from https://gin-gonic.com/

Google. (n.d.). *Go*. The Go Programming Language. Retrieved April 22, 2023, from https://go.dev/

Mapbox. (n.d.). Mapbox: Maps, geocoding, and navigation APIs & SDKs. Retrieved April 28, 2023, from https://www.mapbox.com/

MongoDB Go Driver — Go. (n.d.). MongoDB. Retrieved April 22, 2023, from https://www.mongodb.com/docs/drivers/go/current/

MQTT Protocol Explained: The Basics and a Quick Tutorial. (2023, May 19). EMQ. Retrieved May 24, 2023, from https://www.emqx.com/en/blog/the-easiest-guide-to-getting-started-with-mqtt

Product Overview | EMQX Cloud Documentation. (n.d.). EMQ Documentation. Retrieved May 25, 2023, from https://docs.emqx.com/en/cloud/latest/overview.html#key-features

Single point load cell H30A | BOSCHE. (n.d.). BOSCHE Wägetechnik. Retrieved April 28, 2023, from https://www.bosche.eu/en/scale-components/load-cells/single-point-load-cell/single-point-load-cell-h30a

24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales. (n.d.). Sparkfun. Retrieved April 25, 2023, from https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf

Vue.js. (n.d.). Vue.js - The Progressive JavaScript Framework | Vue.js. Retrieved April 28, 2023, from https://vuejs.org/

What is the Internet of Things - an explainer | World Economic Forum. (2021, March 31). The World Economic Forum. Retrieved May 31, 2023, from https://www.weforum.org/agenda/2021/03/what-is-the-internet-of-things/

Xorshift. (n.d.). Wikipedia. Retrieved May 25, 2023, from https://en.wikipedia.org/wiki/Xorshift