NTNU
Kunnskap for en bedre verden

Overvåking og kontroll av miljøvariabler i båt, camping eller hytte ved hjelp av BLE-enheter

Morten Rostad

Master i kybernetikk og robotikk
Innlevert: juni 2018
Hovedveileder: Tor Engebret Onshus, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Oppgavetekst

Designe et system og utvikle en prototyp for overvåking og styring av miljøvariabler i båt, hytte, camping eller liknende, hvor internett og nettstrøm ikke nødvendigvis er tilgjengelig. Dette skal være et system laget for å brukes med trådløse sensorer og andre enheter. Det må designes et system for registrering av trådløse enheter, og fremstilling av data fra disse på en egen Android app. Styring av systemet skal også skje fra appen.

Forord

Denne masteroppgaven er et videre arbeid fra en prosjektoppgave [1] som ble skrevet høsten 2017 av undertegnede. Noe av det grunnleggende designet, samt enkelte deler av kildekoden er gjenbrukt fra prosjektet i fjor, selv om det er skrevet om underveis. Kapittel 1.1 og 2.1 beskriver noe nærmere hva som er gjort i prosjektoppgaven og hva som er tatt med videre i denne oppgaven. Arbeidet underveis i prosjektperioden har tidvis vært krevende, og for det meste dreid seg om å utvikle et innevevd datasystem med en tilhørende Android app. Mange tusen linjer kode ligger til grunn for teksten i de kommende sidene, og å få uttrykt alt arbeidet jeg har gjort har også vært utfordrende i skrivingen av oppgaven. De som driver med programvareutvikling til daglig vet godt at én enkelt linje med kode kan måtte skrives om hundre ganger før den enten fjernes helt eller kommer med i det endelige produktet. Linjene i denne oppgaven er ofte ingen unntak.

I både prosjekt- og masteroppgave har Tor Onshus vært min veileder. Hos han har jeg hatt 5 veiledningsmøter på omkring 15 minutter hver i løpet av våren. Her har det vært fokus på fremdrift av oppgaven, og jeg har i hovedsak fått tips om hvor jeg bør fokusere mitt arbeid for å kunne ende opp med en ferdig oppgave, samt fått noe hjelp til oppgavens disposisjon. Innholdet i hverken rapporten eller kildekoden er ikke korrekturlest av oppgaveveileder. I begynnelsen av prosjektperioden er det fra ITK-instituttet stilt til rådighet en stasjonær PC med adgang til den programvare NTNU har tilgjengelig for alle sine studenter, samt et digitalt multimeter. All annen maskinvare er fremskaffet av undertegnede selv, for egen regning.

Til sist vil jeg rette en takk til min veileder Tor Onshus og mine medstudenter, spesielt gode kontor-kamerater for et godt samarbeid, gode innspill og motivasjon underveis.

Morten Wiggen Rostad
Trondheim 02.06.18

Oppsummering og konklusjon

Denne oppgaven beskriver prosessen med å designe og delvis produsere et system for fjernstyrt overvåking og kontroll av enkelte miljøvariabler, som for eksempel temperatur, i hus, hytte, båt og liknende. Det designes et system som skal fungere over mobiltelefonnettet, uten tilgang til fast internett, og hvor det benyttes trådløse, lavenergi Bluetooth-sensorer (BLE[1]). Systemet er forsøkt designet slik at det trengs minst mulig tilsyn, slik at det kan fungere over lengre tid uten for eksempel batteriskift. Trådløse BLE-sensorer sender data til en sentral enhet som er koblet til internett via GPRS-nettet. Denne har kontakt med en online database, for datautveksling med en Android app, som sluttbrukeren kan overvåke og styre systemet fra. Systemet er designet med tanke på både lang batterilevetid og lav brukerkostnad. Datatrafikk og SMS-kommunikasjon er derfor forsøkt optimalisert for dette, slik at den årlige driftskostnaden for sluttbrukeren blir svært lav.

Oppgaven er en videreutvikling av et system designet for prosjektoppgave skrevet av undertegnede høsten 2017 på NTNU [1]. I denne masteroppgaven er de overordnede ideene og prinsippene i all hovedsak likt som nevnte prosjektoppgave. En del maskinvare er byttet ut, og det meste av programvare er produsert på nytt, av optimaliserings- og skaleringsgrunner. Som sentral enhet er det benyttet et Espressif ESP32 SoC[2] kort, sammen med et utviklingskort for SIM808 som er brukt til GPRS-kommunikasjon. Med innebygd BLE-transceiver kommuniserer denne med sensorer basert på Nordic Semiconductors nRF51 SoC. Det er utført tester på et development kit basert på samme brikkesett, hvor blant annet noen strøm- og effektmålinger er gjort underveis, for å bidra til å minske strømforbruket og øke batteritiden til systemet.

Ved arbeidets og oppgavens slutt foreligger en prototyp på et system med en sentral, en BLE temperatursensor og et rammeverk for et BLE styrerelé. Disse kan konfigureres og styres fra en enkel Android app. Det er fokusert på å maksimere batteritiden til sensoren, og ut ifra strømmålinger som er gjort er denne anslått til å kunne vare i minst ett år kontinuerlig drift uten batteriskift. Systemet er tenkt designet slik at en utvidelse av systemet skal være forholdsvis enkel å implementere, med ulike sensorer og andre enheter. Det er derfor i stor grad forsøkt å legge til rette for dette i kildekoden.

[1] Bluetooth Low Energy
[2] System on Chip

Abstract

This master thesis describes the process of designing and partially implementing a system for remote surveillance and control of certain environmental variables in houses, cabins, boats etc. The system designed will work by cellular data transfer, in places with no permanent internet connection. The sensors used will be low energy BLE[3] units. It has been a goal to design the system so that it can operate in long periods without the need for supervision (e.g. battery change). Wireless BLE-sensors send data to a central unit which is connected to the internet via GPRS. The central is communicating with an online database, and uses this to transfer information to and from an Android application. The Android app is what the end user will use watch and control the system. The system is designed with both battery optimization and cost efficiency in mind. The GPRS data traffic and SMS communication us therefore optimized, such that the end user cost is relatively low.

The thesis is a development of a system designed for a project written by the author in the autumn of 2017 at the Norwegian University of Science and Technology [1]. In this paper, the overall ideas and concepts are fairly similar to the mentioned project. Some hardware is replaced, and most of the source code for the project is either completely new, or has been severely rewritten for FreeRTOS. Hardware for the central consists of an Espressif ESP32 SoC[4], together with a SimCom SIM808 GPRS used for communication. The ESP32 has a built in BLE transceiver which is used to communicate with units based on Nordic Semiconductor's nRF51 SoC. Tests have been conducted on a Development Kit based on this chip set. These testes include power and current measurements, used to determine the battery lifetime of the BLE units.

By the end of the project period, it is produced a prototype of a system with one central, a BLE temperature sensor and a ready framework for a BLE relay. These are configurable and controllable via an Android app. There has been a focus on increasing the BLE units battery time, and based on measurements made, the battery time of the units is well above one year. The system is designed such that further improvements and additions of new types of BLE units will be easy to implement. This is partially reflected in the way the source code is organized.

[3] Bluetooth Low Energy
[4] System on Chip

Innhold

Tabeller

Figurer

Kapittel 1

Introduksjon

1.1 Tidligere arbeid

I løpet av høsten 2017 skrev undertegnede en prosjektoppgave [1] med samme tema som denne oppgaven. Arbeidet i denne masteroppgaven er basert på det som ble gjort i nevnte prosjektoppgave. Systemet som sto ferdig i prosjektoppgaven høsten 2017, fungerer som en "proof of concept" for videre arbeid. Systemet gir en sluttbruker muligheten til å styre et relé, og lese to ulike sensorverdier ved hjelp av SMS-kommunikasjon. Det er også utviklet et rammeverk for opp- og nedlasting av disse verdiene via en online SQL-database.

Sensorene det er snakk om ble ikke laget som trådløse, men heller for å vise hva som kunne gjøres med sensorverdier som ble logget. All programvare og design ble utviklet med tanke på at denne skulle videreutvikles og være skalerbar for et større system, slik det er beskrevet senere i denne oppgaven. Noe av kildekoden som ble produsert i prosjektoppgaven er tatt med videre, men er da skrevet om for bruk med FreeRTOS, som nevnt i kapittel 3.1.1.

1.2 Motivasjon

Følgende avsnitt er hentet fra "Overvåking og kontroll av miljøvariabler i fritidsbolig og båt", prosjektoppgave høsten 2017 [1].

> I 2017 eksisterer det i Norge over 420 000 hytter, sommerhus eller liknende. Det tilsvarer omtrent 0,27 fritidsboliger per bolig i Norge [5]. Om man regner med at flere husholdninger har flere boliger, sitter man igjen med at omtrent 55% av norske husholdninger eier eller har tilgang til en fritidsbolig [6]. Nordmannen er glad i sin fritid, og til tross for noen år med lite økonomisk vekst det siste tiåret, har vi likevel stadig flere hytter og båter. De aller fleste av disse står urørt det aller meste av tiden. I 2007 brukte nordmannen fritidsboligen sin i gjennomsnitt 36 dager [6]. Samtidig som omtrent halvparten av norske husholdninger har en fritidsbolig, har også omtrent en fjerdedel fritidsbåt [7]. I følge TØIs rapport "Bruk av fritidsbåt i Norge" bruker kun 15% båten hele året. I vintermånedene november – mars brukes fritidsbåten i *gjennomsnitt* kun 4,78 dager i måneden [8].
>
> Mesteparten av tiden står altså norske hytter og fritidsbåter uten tilsyn. For å ta vare på disse er man nødt til å vinterpreservere, holde varmen slik at vann rør og gjennomføringer på båt ikke fryser til, eventuelt i tillegg til å besøke for å inspisere hvordan situasjonen er. I følge Finans Norge er en av de viktigste årsakene til forsikringsutbetalinger på norske hytter knyttet til vannskader, spesifikt frosne vannrør [9]. Andre viktige skader skyldes snøfall, innbrudd og brann. I og med at hytteeiere vanligvis er på hytta i kun kortere tid kan disse skadene forbli uoppdaget lenge og gi store konsekvenser. Liknende problematikk finnes selvsagt også for fritidsbåter, og årlig synker omkring 500 båter her til lands, de fleste på grunn av vanninntrenging, frostspreng, snøfall og liknende [10].

Som vi kan lese er det altså et økende behov for de fleste å kunne overvåke eiendommen sin når man ikke er tilstede, spesielt i vinterhalvåret når båter og hytter ofte står urørt. De fleste uhellene som er registrert kunne ha vært avverget om eierne hadde fått beskjed i god tid når for eksempel strømmen går og temperaturen synker. Det er altså et behov for et produkt hvor man kan samle overvåkingen av ikke bare temperatur og liknende, men også andre variabler man vil ha kontroll over. Dette kan eksempelvis være vannivå i en båt, gassalarmer, bevegelsesalarmer og luftfuktighet. Et slikt produkt må kunne være skalerbart til det sluttbrukeren trenger, og heller ikke medføre store månedlige kostnader.

Datatrafikk og SMS er i dag billig for norske forbrukere, slik at det bør være mulig å lage et system som minimerer data- og SMS-bruken på en slik måte at kostnadene holdes nede, uten større faste månedsavgifter. Selv med en kontantkort-løsning, uten rabbaterte datapakker og inkluderte SMS og ringeminutter, er det mulig å drifte et slikt system veldig billig. Mengen data som faktisk trengs å overføres mellom sentral og sluttbruker er tross alt stort sett veldig liten. Det er oftest snakk om enkle variabler, som endrer seg med veldig lave frekvenser. Unntaket er ved situasjoner hvor man for eksempel vil ha live tracking av større variabler, overføring av video, bilder og lyd. Dette er likevel noe den enkelte bruker, og videreutvikler av systemet enkelt burde kunne justere og tilrettelegge for, og er ikke et fokus i denne oppgaven.

1.3 Mål og avgrensning

Målene for denne oppgaven kan i hovedsak deles inn og avgrenses til to delmål. Ett aspekt av oppgaven tar for seg design og utvikling av trådløse lavenergi BLE-sensorer og enheter som kommuniserer med en sentral enhet, heretter ofte omtalt som *sentralen*. Den andre delen handler om å designe og utvikle et overordnet system som benytter seg av data som blir lastet opp til en SQL-database fra sentralen, og presentere disse på en egen app på brukerens Android-enhet.

1.3.1 BLE-enheter og sentral

For å gi et godt inntrykk av hvordan systemet kan fungere med ulike sensorer og andre typer enheter bør det designes og utvikles en prototyp på to ulike typer BLE enheter, en temperatursensor og et styrt relé. Disse vil fungere som eksempler på hvordan rammeverket rundt er bygget opp og hvordan dette kan omgjøres og tilpasses til ulike sensorer og andre typer enheter. For sensorer er det et hovedfokus på å minimere strømforbruket, da dette er en frittstående enhet. Dette i motsetning til en relé-enhet, som også bør bruke minimalt med strøm, men likevel har større toleranser ettersom denne i de fleste bruksområder vil være koblet til en større strømkilde (stor batteribank, 220V stikkontakt i veggen, etc.).

Basert på tidligere arbeid [1] skal det utvikles en sentral som mottar data fra enhetene og sender konfigurasjonsdata til disse. Sentralen skal også sende og motta informasjon til og fra sluttbrukeren både via data lagret på SQL-server og på SMS. Strømforbruket til sentralen er ikke hovedtema i denne oppgaven. Likt som med releet antas det at denne er koblet til en større strømkilde.

1.3.2 Android app

Det skal utvikles en egen app for Android som kan presentere data fra de ulike enhetene, samt ha mulighet for å konfigurere systemet. Appen skal hente data fra en online database og presentere disse i en historisk graf. Appen må generelt være lett å bruke, rask og responsiv.

Kapittel 2

Teori og forarbeid

2.1 Systemet og tilgjengelig kildekode ved prosjektstart

I løpet av prosjektoppgave høsten 2017 [1] ble det utviklet et SMS-grensesnitt for kommunikasjon mot maskinvaren. Det ble implementert et system for registrering og administrering av både brukere og ulike enheter. Det ble også utviklet et system for å oppdatere en SQL-database med sensor- og reléverdier. Kapittel 5 i prosjektoppgaven beskriver forslag til videre arbeid. En del av disse forslagene er det jobbet videre med i denne oppgaven. Det medfører blant annet at mye av kildekoden som ble skrevet høsten 2017 ikke direkte er brukt i den nye utgaven av systemet. Enkelte av programvaremodulene er beholdt, men systemet som helhet er skrevet om til å kjøre på to kjerner i et sanntids operativsystem, slik at det meste har måttet blitt skrevet om i løpet av prosessen. Mye av det overordnede designet og kravspesifikasjonen er likevel tatt med videre og danner grunnlaget for systemet slik det står ferdig i denne masteroppgaven. Maskinvaren for GPRS-kommunikasjon (SIM808) er lik i begge prosjektene, og programvaremodulen som ble skrevet for å kommunisere med denne er i hovedsak brukt som den er skrevet i den forrige oppgaven, bortsett fra at den er skrevet om til å fungere i et trådbasert operativsystem.

2.2 Kravspesifikasjon

For et programvareprosjekt i denne størrelsen er det hensiktsmessig å sette opp en oversikt over krav til systemet, og koble disse til spesifikke prioriteter og mål underveis. Noen av kravene til dette systemet er til dels basert på tidligere krav til systemet som ble utviklet til tidligere prosjektoppgave [1], men selvfølgelig utvidet og gjort om for den nye oppgaven. Vi deler opp kravspesifikasjonen i *funksjonelle krav* og *ikke-funksjonelle krav*. De funksjonelle kravene skal definere hva systemet er ment å oppnå og hvilke grunnleggende funksjoner systemet skal ha. De ikke-funksjonelle kravene er kvalitetskrav en bruker vil forvente av et slikt system. Alle krav er gitt en ID og tilhørende prioritet. I dette prosjektet er prioritetene satt til **Lav**, **Middels**, og **Høy**.

Tabell 2.1: Funksjonelle krav

ID	Beskrivelse	Prioritet
FK1	Systemet må kunne vise sensordata og historikk på appen	Høy
FK2	Systemet må kunne gi alarmer på appen	Medium
FK3	Systemet må kunne gi alarmer på SMS	Lav
FK4	Systemet må være konfigurerbart via appen	Høy
FK5	Systemet må ha støtte for at flere ulike enheter er koblet til samtidig	Høy
FK6	Systemet må kunne styre releer via appen (over SMS eller HTTP)	Høy

Tabell 2.2: Ikke-funksjonelle krav

ID	Beskrivelse	Prioritet
IFK1	Appen skal være lett og intuitiv å bruke	Høy
IFK2	Det skal være lett å legge til og fjerne enheter	Middels
IFK3	Skjermgrafikken og resten av appen skal ikke stoppe opp når data lastes inn	Middels
IFK4	Systemet må oppleves som kjapt og responsivt av brukeren	Middels
IFK5	Enhetene kan sorteres av brukeren på appen	Lav
IFK6	Innstillinger som gjøres på appen skal vedvare når den avsluttes	Middels

I løpet av prosjektperioden er det designet et system som skal kunne oppfylle de funksjonelle og ikke-funksjonelle kravene i tabell 2.1 og 2.2. Senere i rapporten følger mer detaljerte beskrivelser av ideer og design som senere ble implementert slik det er beskrevet i kapittel 3 og kapittel 4. Kapittel 6 oppsummerer hvilke krav som er oppfylt på slutten av prosjektperioden.

2.3 Komponenter

2.3.1 ESP32

For maskinvare til sentralen er det brukt en Espressif ESP32 SoC vist i figur 2.1, sammen med en SIM808 GPRS-modul beskrevet i 2.3.2. ESP32 er en kraftig videreføring/oppgradering av ESP8266, som ble brukt i det tidligere arbeidet. På grunn av størrelsen av prosjektet, og behovet for BLE og ikke minst å kunne håndtere flere ulike enheter samtidig, var det naturlig å oppgradere maskinvaren. ESP8266 har dessuten ikke innebygd Bluetooth/BLE, og ville ha hatt behov for en ekstern BLE-transceiver. ESP32 har integrert Bluetooth/BLE transceiver og to kjernes 160 MHz prosessor, hvor i dette prosjektet, den ene prosessoren i hovedsak er forbeholdt BLE-tjenester [11]. I tillegg til de to kjernene har ESP32 en egen innebygd ULP[1]-prosessor, for å håndtere blant annet timere og andre dyp-søvn-rutiner. Denne brukes ikke i oppgaven, ettersom det ikke har vært fokus på å minimere effektforbruket til den sentrale enheten. Dersom det ved videre arbeid er ønskelig å gjøre dette, vil ESP32 altså fortsatt være en god platform å jobbe videre med, blant annet på grunn av mulighetene ULP-en gir.

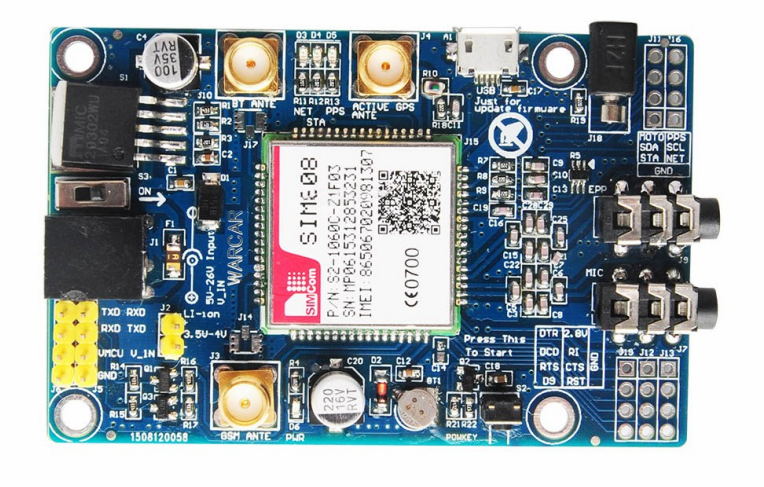
Figur 2.1: ESP32 Development Board [2]

[1] Ultra Low Power

2.3.2 SIM808

En Simcom SIM808-modul, vist i figur 2.2, er brukt for å kommunisere via GPRS. Kortet er en komplett Quad-Band 2G GSM/GPRS-modul, også med innebygd GPS-mottaker [12]. Kortet er det samme som er brukt i tidligere prosjektoppgave [1]. En del av planen i prosjektoppgaven var å muligens utvide systemet til å også motta GPS-data for å logge posisjon. Dette er ikke tatt med videre i denne oppgaven, men i og med at GSM/GPRS-grunnfunksjonene som trengs av modulen er likt, ble det besluttet å benytte samme kort og ikke skaffe et nytt.

Kommunikasjonen mellom ESP32 og SIM808-kortet skjer over UART. Protokollen er en oppgradert versjon av Hayes Command Set (AT-commands[2]) [13]. Protokollen som støttes av SIM800-serien benytter mange vanlige AT-kommandoer som benyttes i mange modem- og teleprodukter. I tillegg eksisterer mange andre kommandoer spesifikt for forskjellige typer moduler, blant annet Simcoms egne kommandoer [13]. Programvaremodulen som er skrevet for å kommunisere med SIM808-kortet er skrevet med enklest mulige kommandoer. Dermed kan kortet direkte byttes ut med et nærmest hvilken som helst GPRS-modul som kommuniserer med AT-kommandoer over en seriell UART-forbindelse, uten noen som helst endringer i kildekoden.



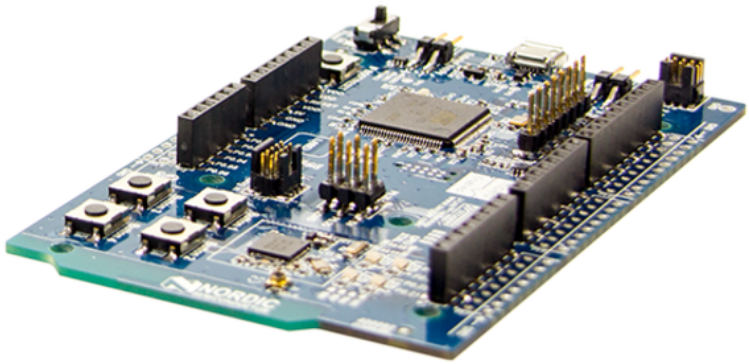
Figur 2.2: SIM808 Development Board [3]

[2] AT: ATTENTION-kommando. AT-kommandoen er brukt som et prefiks i seriell-kommunikasjonen. Etter prefikset følger andre parametre som definerer kommandoen.

2.3.3 nRF51

De trådløse BLE-sensorene er utviklet på et nRF51 DK[3] fra Nordic Semicondictors [4], vist i figur 2.3. Kortet støtter utvikling for nRF51822, nRF51824 og nRF51422 SoCs. nRF51 ble valgt over den nyere og noe mer kostbare nRF52, da dette prosjektet kun benytter seg av BLE, og ikke BLE+ANT. Systemet har heller ingen krav som har bruk for nRF52s kraftigere ARM Cortex-M4F prosessor. Om systemet derimot videre skulle støtte en meshkonfigurasjon, og ikke kun stjernekonfigurasjon slik det gjør i dag, kunne sensorene heller vært basert på den nyere nRF52-familien.

Det ble valgt å utvikle på Nordics utviklingskort fremfor å utvikle direkte på en mer barebone utgave av SoCen. Kortet har innebygd SEGGER debugger, som letter utviklingen av kortet betraktelig. Det har også egne terminaler for å måle strøm- og effektforbruket til nRF51822-chipen. Disse terminalene måler kun forbruket til chipen, ikke inkludert debugger og andre funksjoner som er inkludert i utviklingskortet. Når prosjektet ferdigstilles kan programvaren enkelt lastes opp på ferdig, tilpasset hardware, hvor kun nRF51 SoCen, nødvendig strømforsyning, og eventuell andre aktuelle periferienheter er tilstede.



Figur 2.3: nRF51 Development Kit [4]

[3]Development Kit

2.4 Utstyrsliste

Følgende er en komplett liste over utstyr og programvare som er brukt i forbindelse med denne oppgaven.

2.4.1 Maskinvare

- Espressif ESP32
- Simcom SIM808
- Nordic Semiconductor nRF51422 Development Kit

2.4.2 Programvare

- Visual Studio Code V1.22.1
- MSYS32 V2.8.1
- ESP-IDF V2.1
- µVision V5.24.2.0
- J-Link RTT Viewer V6.30b
- Android Studio V3.1.1

2.4.3 Annet utstyr

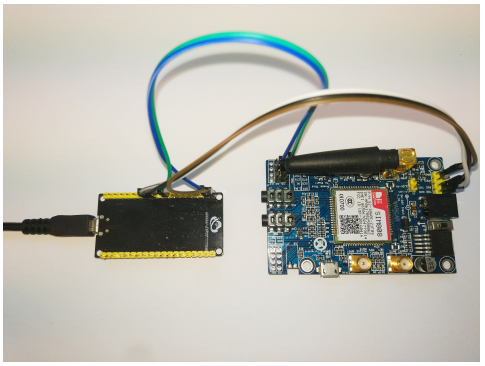
- Instrutek MAMY-60 Multimeter
- Xiaomi Mi5s Android smarttelefon

Kapittel 3

Sentral enhet og BLE sensorer

3.1 Design

Noe av designet og ideene til den sentrale enheten og de trådløse enhetene i denne oppgaven er basert på arbeid i tidligere prosjektoppgave [1]. Kapittel 3 og kapittel 5 i prosjektoppgaven beskriver det overordnede systemet, og hvordan dette ble designet med tanke på utvidelse til blant annet trådløse, lavenergi sensorer og andre enheter. I dette kapittelet skal vi se på hvordan dette designet er utvidet, hvordan de trådløse enhetene er designet og hvordan sentralen er endret for å passe inn i det nye designet. Figur 3.1 viser oppsettet av sentralen, med ESP32 og SIM808 koblet sammen.



Figur 3.1: Sentral enhet

3.1.1 Sentral enhet

Sentralen er oppbygd rundt en ESP32 og et SIM808 GPRS adapter-kort, som nevnt i 2.3.1. I korte trekk er sentralens rolle å håndtere de ulike trådløse enhetene, sende konfigurasjon til disse og motta blant annet sensordata. Dataene sendes så til en online database. Konfigurasjonsdata og enkelte styringsdata kan også hentes fra online database når sentralen får beskjed om det via en SMS-melding. Slik unngår sentralen å kontinuerlig måtte sjekke databasen for oppdateringer, noe som ville ha medført større datatrafikk over GPRS.

Som rammeverk for programmeringen av ESP32-en er det valgt å bruke produsenten Espressifs eget ESP-IDF[1] (Apache-lisens) [14]. ESP-IDF er et utviklinsrammeverk designet for å lette arbeidet med å utvikle IoT-applikasjoner med både nettverkstilkobling, Bluetooth/BLE, og effekthåndtering. Dette fasiliterer på en god måte bruk av begge kjernene og generell tråd- og oppgavebehandling. Systemet er basert på det åpne sanntidsoperativsystemet FreeRTOS (MIT-lisens), som for øvrig er en del av ESP-IDF. Programvaren til sentralen er modulbasert, hvor hver modul er uavhengig av hverandre. Figur 3.2 viser en noe forenklet modell over hvordan programvaren i sentralen er bygd opp. Kildekoden i sin helhet kan sees i vedlegg A.

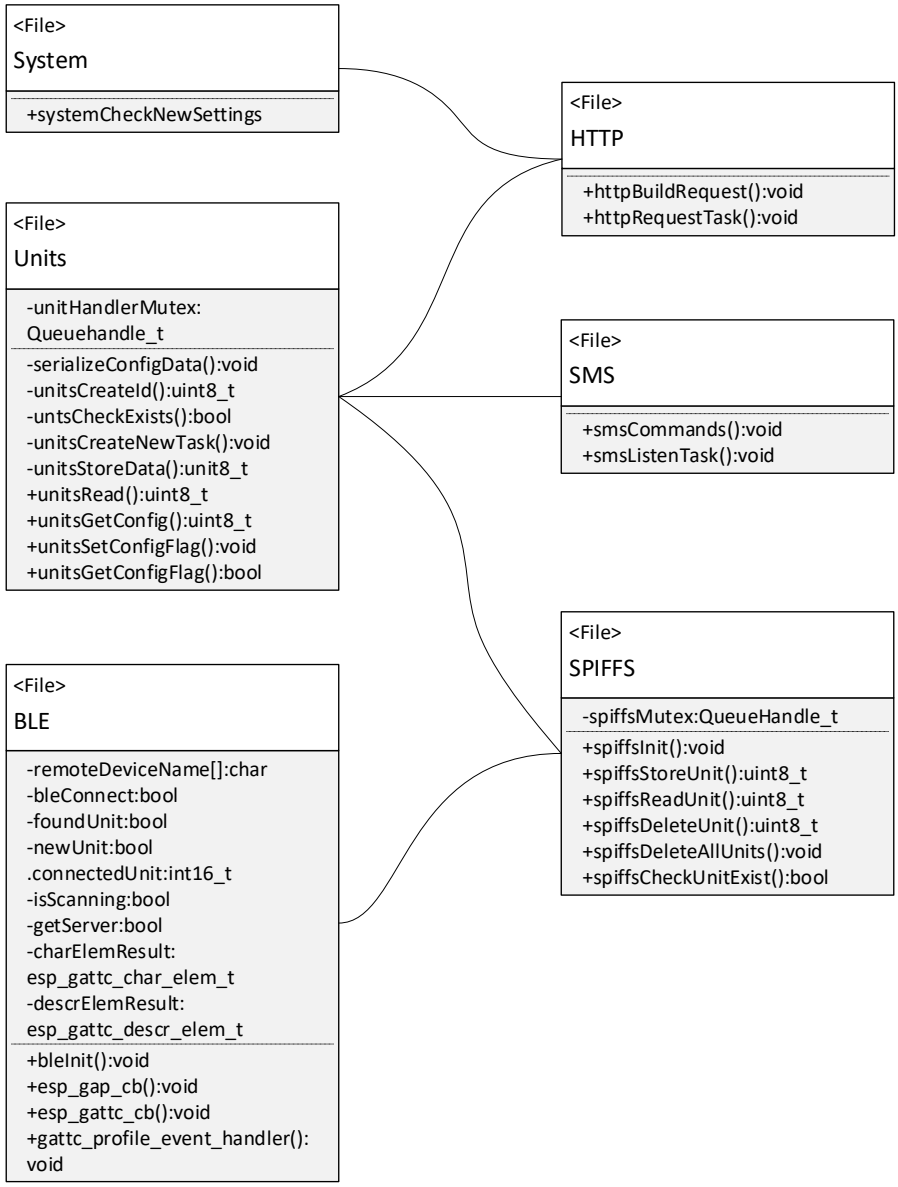
I programvaren til sentralen er det i hovedsak tre oppgaver som kjører samtidig:

- Enhetsbehandler: Håndterer behandlingen av de ulike enhetene
- BLE: Scanner etter BLE-enheter og leser/skriver til disse
- SMS: Lytter etter innkommende SMS-meldinger

ESP32 har to kjerner tilgjengelig. I konfigurasjonen til dette prosjektet er en av kjernene forbeholdt BLE-aktivitet, slik at BLE-tråden uansett aldri blir stående å vente på en annen tråd. Hver av oppgavene starter også egne tråder med oppgaver ved forskjellige hendelser. Blant annet vil enhetsbehandleren starte en nettverkstråd når en enhet har oppdaterte verdier. Den nye nettverkstråden vil så laste opp de nye verdiene til databasen før tråden avsluttes.

Nettverk via SIM808-enheten er implementert ved hjelp av en PPPOS-løsning[2], basert på Boris Lovosevics libGSM for ESP32 bibliotek (Public Domain / CC0) [15]. Løsningen benytter seg av lwIP[3], som er en veldig liten implementasjon av TCP/IP-protokollen, spesielt laget for innebygde datasystemer med redusert RAM-kapasitet og behov for en full TCP-stakk [16]. På toppen av disse modulene er det så implementert en egen HTTP-modul som tar for seg HTTP-kall til server. Alle nettverksfunksjoner vil kjøre på egne tråder, slik at resten av programmet ikke blir stående å vente på eventuell respons fra serveren. Det er tross alt snakk om data over GPRS, så hastigheten er ikke veldig høy.

[1] Espressif IoT Development Framework
[2] Point-to-Point Protocol Over Serial
[3] lightweight IP



Figur 3.2: Moduler i sentral enhet

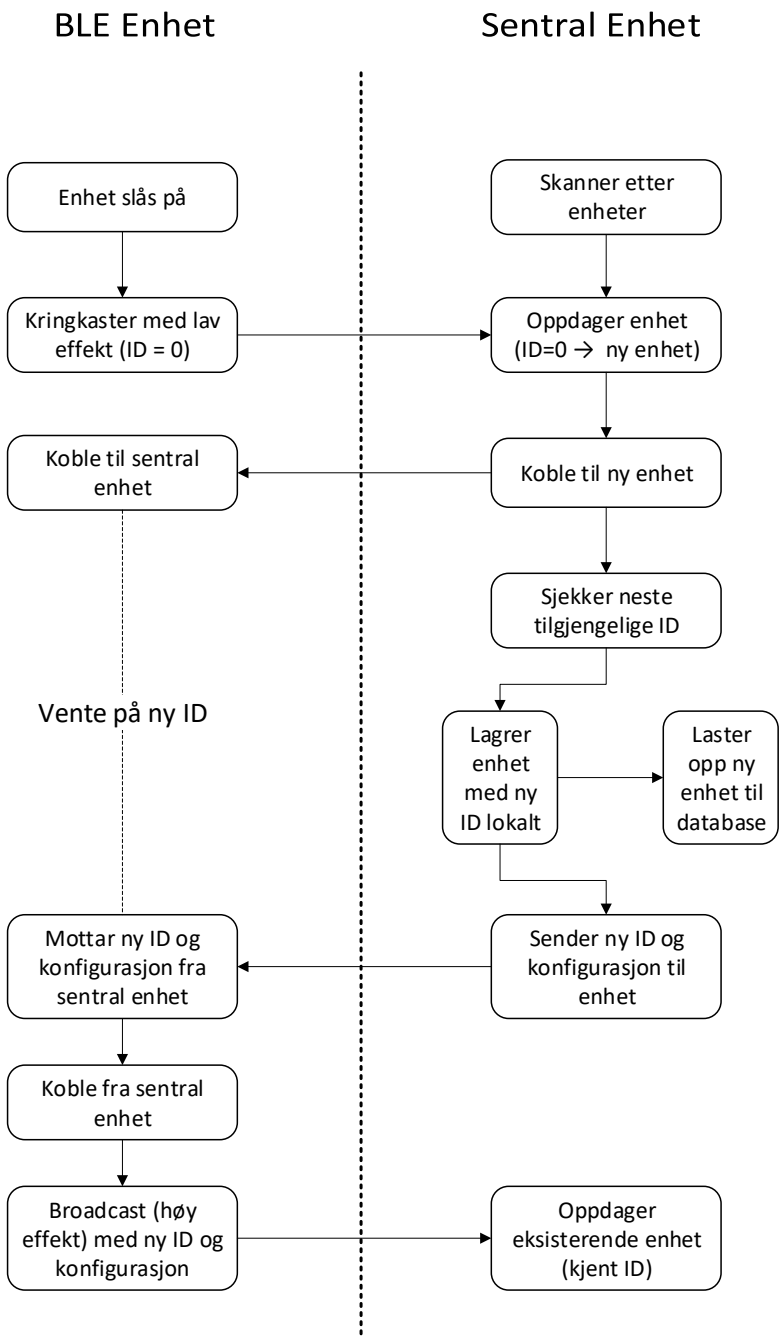
3.2 Automatisk tilkobling

I følge kravene *FK5* og *IFK2* i henholdsvis tabell 2.1 og tabell 2.2, må det tilrettelegges for at flere ulike BLE-enheter skal kunne kobles til systemet samtidig, og at det skal være enkelt å koble til disse. Et annet hensyn å ta er at hver enhet kun skal tilhøre én sentral, selv om flere sentraler kan være innen rekkevidde. Et eksempel på et slikt scenario kan være på en båtbrygge eller campingplass, hvor flere brukere har systemer som er i relativ nærhet til hverandre og BLE-signalene overlapper hverandre. På grunn av dette måtte det lages et rammeverk for tilkobling av enheter mot en spesifikk sentral. For å få en enklest mulig brukeropplevelse ble det bestemt at tilkoblingen av nye enheter skal skje automatisk, uten at brukeren for eksempel må skrive inn noe passord eller holde knapper inne på BLE-enheter for å pare de med sentralen. Ved å unngå dette, slipper man å være avhengig av ekstra knapper på enhetene, som ellers vil være overflødige. Dette skaper på en annen side noen utfordringer med sikkerheten, som adresseres videre i kapittel 7.4.3.

Prosedyren som er implementert er vist i figur 3.3. Alle enheter deler et navn, definert i kildekoden. Selve navnet er ikke viktig (det vil ikke leses av brukeren), men det forteller sentralen at en BLE-enhet som er kompatibel med systemet er funnet. Hver enhet har også et ID-nummer. Enhetens navn og ID er en del av advertisement-pakken som enheten kringkaster når den skrus på. Når enheten først skrus på vil den kringkaste denne pakken med lavest mulig effekt og med ID = 0. Med lav effekt må BLE-enheten og sentralen i praksis være nærmere enn omtrent en meter fra hverandre for at sentralen skal oppdage enheten. Sentralen oppdager da en ny kompatibel enhet med ID = 0. En enhet med ID = 0 forteller det sentrale systemet at enheten er ny og ikke tidligere konfigurert. Sentralen vil koble til enheten, lage en ny unik ID og sende denne. BLE-enheten vil så oppdatere sin egen ID og øke kringkastingseffekten til full styrke, slik at rekkevidden økes. Neste gang enheten kringkaster advertisement-pakken sin vil den gjøre dette med sin nye unike ID og for full effekt. Sentralen vil da gjenkjenne ID-en som en tidligere kjent enhet.

For brukeren sin del betyr dette at alt som trengs å gjøres for å legge til en ny enhet er å skru på en BLE-enhet i nærheten (ca 1 m) av sentralen, og å sørge for at kun én ny enhet er i nærheten om gangen. Straks enheten er registrert av sentralen vil informasjon om den nye enheten lastes opp på serveren og være tilgjengelig på appen. Her kan brukeren legge til et nytt beskrivende navn til enheten.

Som et utgangspunkt for kildekoden til enhetene er det benyttet eksempelkode fra Nordic Semiconductors [17], som inneholder et enkelt rammeverk med grunnleggende funksjoner for kringkasting og tilkobling av BLE-enheter. Dette er så utviklet videre til å oppfylle kravene i denne oppgaven.



Figur 3.3: Prosedyre for å legge til ny enhet

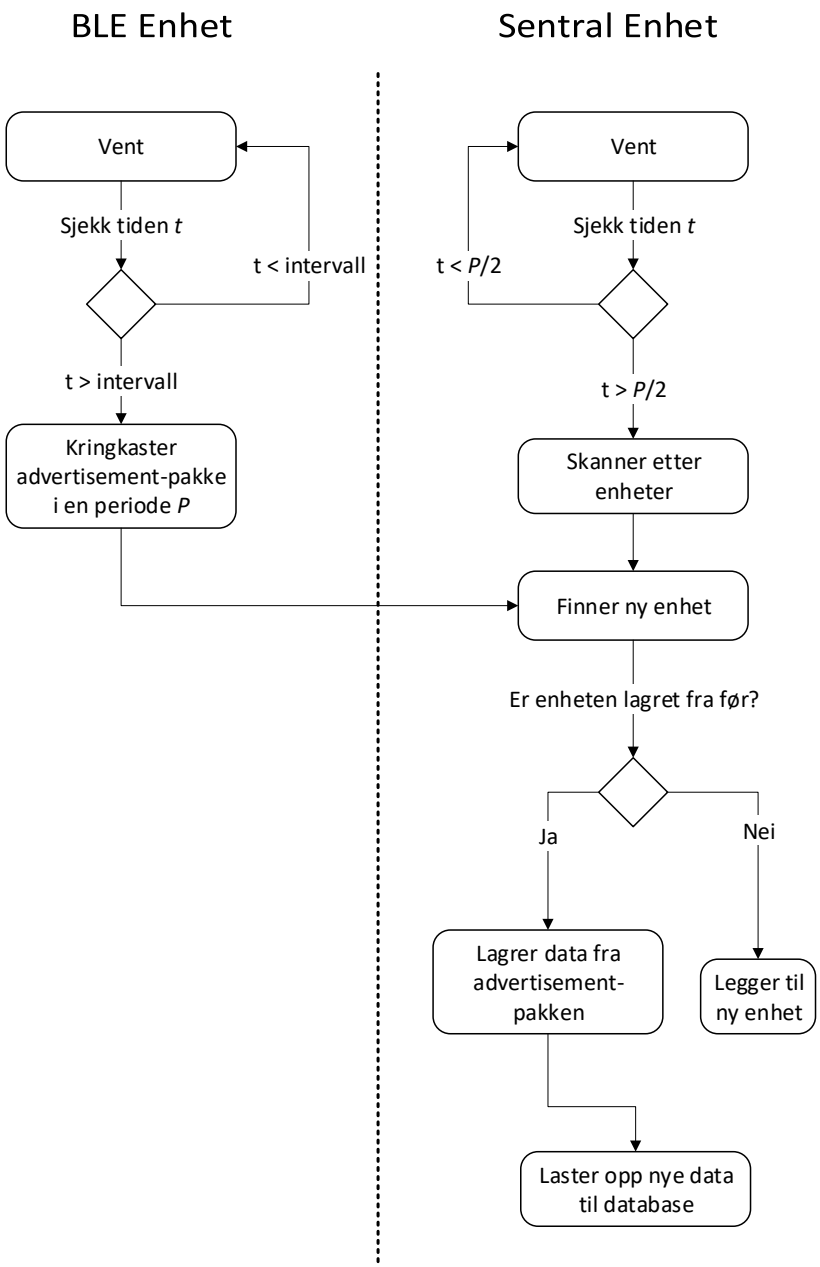
3.3 Overføring av data fra enhet til app

Når en BLE-enhet er lagt til og er kjent for systemet, skal data fra denne lastes opp til databasen, slik at disse er tilgjengelig på Android applikasjonen. I tilfelle det er en aktiv enhet, for eksempel et relé, skal det kunne skrives kommandoer til enheten. Derfor trengs et rammeverk for overføring av informasjon mellom enheten og databasen, med sentralen som mellomtjener.

3.3.1 Skanning og advertising

Når en ny enhet er oppdaget og lagt inn i minnet til den sentrale enheten vil den være der helt til brukeren sletter den fra appen. Dette fjerner enheten fra databasen, og sentralen kan så fjerne enheten fra lokalt minne. Enhetene og tilhørende metadata blir lagret på sentralens EEPROM, slik at informasjonen vil fortsette å være der ved en eventuell restart, eller til sentralen får beskjed fra appen om å slette én eller flere enheter.

Prosedyren for avlesing av enheter, slik det er forklart under, vises i figur 3.4. Slik systemet er laget så langt i denne oppgaven, vil sentralen og enhetene kommunisere med et fast tidsintervall. Dette intervallet er hardkodet i programvaren til både BLE-enhetene og sentralen. For testing er dette intervallet satt til 1 minutt. I kapittel 7.2.1 diskuteres det hvordan dette tidsintervallet kan endres. BLE-enhetene har også en fast periode P som er hvor lenge enheten kringkaster sin advertisement-pakke. Denne perioden P er også lagret i sentralen. Setralen skanner for BLE-enheter, men for å senke strømforbruket gjør den ikke det kontinuerlig. For å sikre at alle pakkene blir lest av sentralen er det derfor viktig at sentralen vet hvor lenge hver enhet kringkaster. Når sentralen vet at hver enhet kringkaster i P sekunder, kan sentralen scanne med et tidsintervall som er kortere enn P. I kapittel 3.4 diskuteres disse verdiene videre med tanke på batteritiden.



Figur 3.4: Prosedyre for å lese data fra en enhet

Når en BLE-enhet kringkaster sin adverisementpakke er denne altså i utgangspunktet garantert å bli mottatt av sentralen. Sentralen vil så avgjøre om denne enheten allerede eksisterer, og legge den til hvis den er ukjent, slik det er beskrevet i kapittel 3.2. Om enheten er kjent vil sentralen avgjøre hva slags enhet det er snakk om, basert på informasjon i enhetens advertisement-pakke. Dersom det er en sensor med data som skal logges lastes disse opp til databasen. Om enheten er et relé eller en annen type som venter på input vil sentralen sjekke i sitt lokale register om det er kommet anmodning om å endre verdien på enheten. Dersom det er det kan sentralen opprette en kobling til enheten og sende ny konfigurasjon eller input. Kun avlesing av sensorer, samt noe av systemet for skriving til releer er ferdig implementert i systemet ved prosjektets slutt. Kapittel 7 beskriver nærmere hvordan denne funksjonaliteten kan videreutvikles.

Data fra enhetene blir umiddelbart lastet opp til en online MySQL-database når de mottas av sentralen. Databasen inneholder historikk over alle enhetene som er knyttet til sentralen, helt til brukeren selv velger å fjerne informasjonen via et eget valg i appen. Når appen åpnes på telefonen vil all informasjon om enhetene lastes ned og en liste over tilknyttede enheter vises for brukeren. Brukeren kan åpne hver enkelt enhet, og avhengig av hvilken type enhet det er snakk om vil ulik informasjon vises. For en sensor vil historikk over måledata vises i en interaktiv graf. Siste måledata er også fremhevet, slik at brukeren raskt kan se hva siste avleste måling er. For releer vises historikk over hvilke tilstander de har hatt, samt mulighet for å skru av og på releet. Dette beskrives nærmere i kapittel 4.

3.4 Strømforbruk og optimalisering

De trådløse BLE-enhetene er ment å kunne drives av et enkelt CR2032 knappecellebatteri, og ideelt skal enhetene fungere i minimum ett år før batteriet må skiftes. Perioden på ett år er satt som et mål ut ifra de fleste tiltenkte bruksområdene til systemet. Det antas altså at et system vil ha tilsyn minst én gang i året. Om enhetene kan drives lengre enn ett år på et batteri er det selvfølgelig enda bedre.

Som nevnt tidligere er Nordic Semiconductors nRF51-serie valgt nettopp på grunn av sitt lave strømforbruk, og mulighetene chippen har for å gå i en dvalemodus når den ikke er i bruk. nRF51-serien har to strømsparende moduser, *System On*, og *System Off* [18]. System Off er systemets dypeste strømsparemodus, hvor kun deler av RAM-en er aktiv og resten av systemet er inaktivt. Det eneste som kan vekke systemet fra System Off er et eksternt interrupt-signal fra for eksempel et knappetrykk. Vi kan altså ikke bruke System Off i vårt system, ettersom enheten trenger å våkne basert på egen timer interrupt, ikke eksternt.

System On er en modus hvor CPU-en og alt periferiutstyr er funksjonelt, men kan gå i søvn-modus for å spare strøm. Dermed kan en av timerene vekke CPU-en ved faste intervall, slik vi har behov for i systemet vårt. På den måten brukes minst mulig strøm ved å la CPU-en sove mesteparten av tiden, og kun våkne når enheten skal kringkaste ny data, eller eventuelt sjekke for nye oppdateringer. I vår implementasjon settes det opp en timer, *RTC0*, som trigger et interruptsignal med et gitt tidsintervall, nevnt over i 3.3.1. I tiden når CPU-en sover er det i vårt system ikke noe annet periferiutstyr, slik som UART, I2C eller liknende i SoC-en aktivt, det er derfor kun deler av RAM-en og RTC0 som er aktivt og forbruker minimalt med strøm.

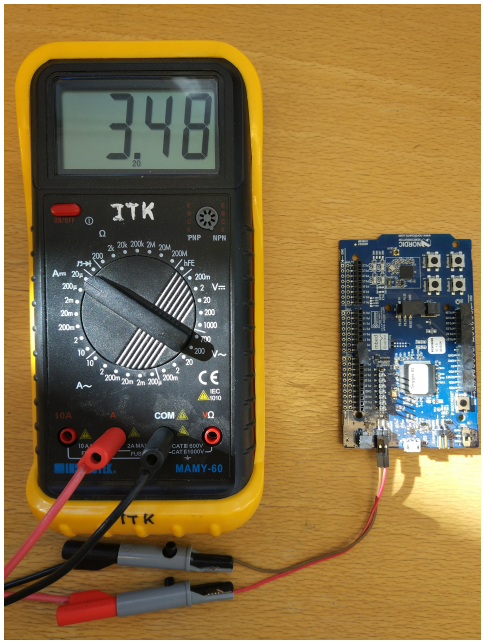
For å optimalisere batteritiden til systemet er det mulig å endre flere variabler. Den første, og kanskje mest åpenbare, er hvor ofte enheten skal våkne og kringaste sin advertisement-pakke. Dette er til en viss grad bestemt ut ifra hvilken type enhet det er snakk om og, som det nevnes senere i kapittel 7.2.1, hvor ofte brukeren ønsker oppdateringer ifra den enkelte enheten. Dette kan variere fra en temperatursensor i en kjeller hvor det ikke er antatt store temperatursvingninger, til for eksempel en luftkvalitets-sensor som forventer hyppigere endringer. Avhengig av hvor ofte enheten våkner, endres batteritiden betraktelig. Den andre variabelen man kan endre er hvor lenge enheten kringkaster pakken sin. Denne perioden må være lang nok til at sentralen klarer å fange den opp. En sentral som er tilknyttet mange enheter kan måtte lese flere enheter samtidig, dermed må enhetene vente, og kringkaste i lengre perioder. Dersom perioden er for kort risikerer man at sentralen ikke oppdager den (sentralen kan for eksempel være opptatt med å skrive til en annen enhet), og data fra enheten som kringkaster blir tapt. I tillegg til disse variablene har størrelsen på advertisement-pakken noe å si for strømforbruket når enheten kringkaster. En større advertisement-pakke vil forbruke mer strøm enn en mindre. Typisk vil en pakke med 10 bytes forbruke 20%-25% mindre strøm enn en full pakke på 31 byte[19].

3.4.1 Målinger

For å gjøre strømmålinger ble nRF51 utviklingskortet koblet mot et vanlig amperemeter for å måle gjennomsnittsstrøm både når enheten kringkastet, og når enheten var i dvale. Utviklingskortet har egne utganger for strømmålinger, som kan brukes om kortet forberedes for måling slik det beskrives i kortets dokumentasjon [20]. Ved å gjøre disse nødvendige endringene på kortet måler disse utgangene strømmen direkte fra knappecellebatteriet på baksiden av kortet til SoC-en. Det måles da altså ikke strøm igjennom debuggeren eller noe annet som finnes på kortet, og gir derfor et korrekt bilde av hvor mye strøm SoC-en trekker. Figur 3.5 viser det enkle måleoppsettet når enheten er i dvale. Resultatene i tabell 3.1 viser gjennomsnitt av 10 målinger, både for dvalemodus og kringkastingsmodus.

Tabell 3.1: Strømmålinger

Dvale	**Kringkasting**
$3.48\ \mu A$	$170\ \mu A$



Figur 3.5: Måleoppsett

Et estimert gjennomsnittlig strømforbruk av enheten kan gis av formelen 3.1. Videre kan man så regne ut estimert batteritid i dager ved å bruke formel 3.2. Ved å teste ulike verdier for hvor lenge enheten er i dvale- og kringkastingsmodus er det kommet frem til resultatene i tabell 3.3. I beregningen er det benyttet en batterikapasitet på $0.22Ah$ som er en relativt vanlig verdi for et CR2032 knappecellebatteri. Symboler er beskrevet i tabell 3.2.

$$I_{avg} = K \cdot 170\ \mu A + D \cdot 3.48\ \mu A \tag{3.1}$$

$$T = \frac{C}{I_{avg} \cdot 24} \tag{3.2}$$

Tabell 3.2: Symboler

Symbol	Betydning
I_{avg}	Gjennomsnittlig strømforbruk
K	Andel tid enheten er i kringkastingsmodus
D	Andel tid enheten er i dvalemodus
T	Tid i dager
C	Batterikapasiteten

Tabell 3.3: Estimert batteritid

Dvale	**Kringkasting**	D	K	I_{avg}	T
30 sek	10 sek	0.5	0.5	$86.75\mu A$	105 dager
45 sek	10 sek	0.74	0.25	$45.13\mu A$	203 dager
50 sek	10 sek	0.83	0.17	$31.25\mu A$	293 dager
1 min	10 sek	0.86	0.14	$27.29\mu A$	336 dager
1 min	5 sek	0.92	0.08	$16.31\mu A$	562 dager
2 min	10 sek	0.92	0.08	$16.31\mu A$	562 dager
3 min	10 sek	0.95	0.05	$12.26\mu A$	747 dager
10 min	10 sek	0.98	0.02	$6.23\mu A$	1471 dager
1 time	10 sek	0.997	0.003	$3.96\mu A$	2314 dager

Resultatene i tabell 3.3 viser for det første, ikke uventet, tydelig at batteritiden øker jo større D blir. Som det såvidt er nevnt tidligere har vi begrensede muligheter for å justere kringkastingstiden, da denne ikke må bli for lav. Hvis vi ønsker et intervall på ett minutt eller mindre ser vi at vi må senke kringkastingstiden til under 10 sekunder for at batteritiden skal være mer enn ett år. En såppass kort kringkastingstid vil kunne bety at enheten ikke rekker å bli oppdaget av sentralen, før kringkastingen er ferdig, for eksempel om sentralen er opptatt med å kommunisere med andre enheter. Dessuten trenger sentralen å scanne etter enheter oftere dersom kringkastingstiden til hver enhet er lav. Dette påvirker strømforbruket til den sentrale enheten. Sentralens strømforbruk er ikke tema i denne oppgaven, men det kan være aktuelt for videre arbeid. Den viktigste justeringsvariablen her blir altså dvaleintervallet mellom hver gang enheten kringkaster, noe som i hovedsak er styrt av funksjonen til enheten. Om vi setter krav for at enheten skal vare i minst ett år på batteritid, og at kringastingstiden skal være 10 sekund ser vi at dvaletiden må være i overkant av ett minutt, noe som er akseptabelt for for eksempel en temperatursensor i et vanlig rom med langsomme temperatursvingninger.

Resultatene gir imidlertid ikke nødvendigvis et helt riktig bilde av hvordan den reelle batteritiden er. Vi må ta hensyn til både batteriets egen utlading, og hvor godt enheten fungerer ettersom batteriet blir ladet ut og får både lavere spenning og kan levere mindre strøm. Tester andre har utført med liknende batteri [21] antyder et stort pakketap når spenningen synker og batteriet ikke lengre kan levere full effekt. Likevel er det i all hovedsak viktigst å senke gjennomsnittlig strømforbruk for å øke batteritiden, ettersom kortere og lavere strømtopper påvirker levetiden på et knappecellebatteri i mindre grad [22]. I tillegg må man ta hensyn til batteriets egen utlading, og hylletid som sådan. Det er ingen vits i å designe kretsen for en ekstremt høy batteritid dersom denne overstiger batteriets egen lagringstid. Knappecellebatterier leveres ofte med en lagringstid på 7-10 år.

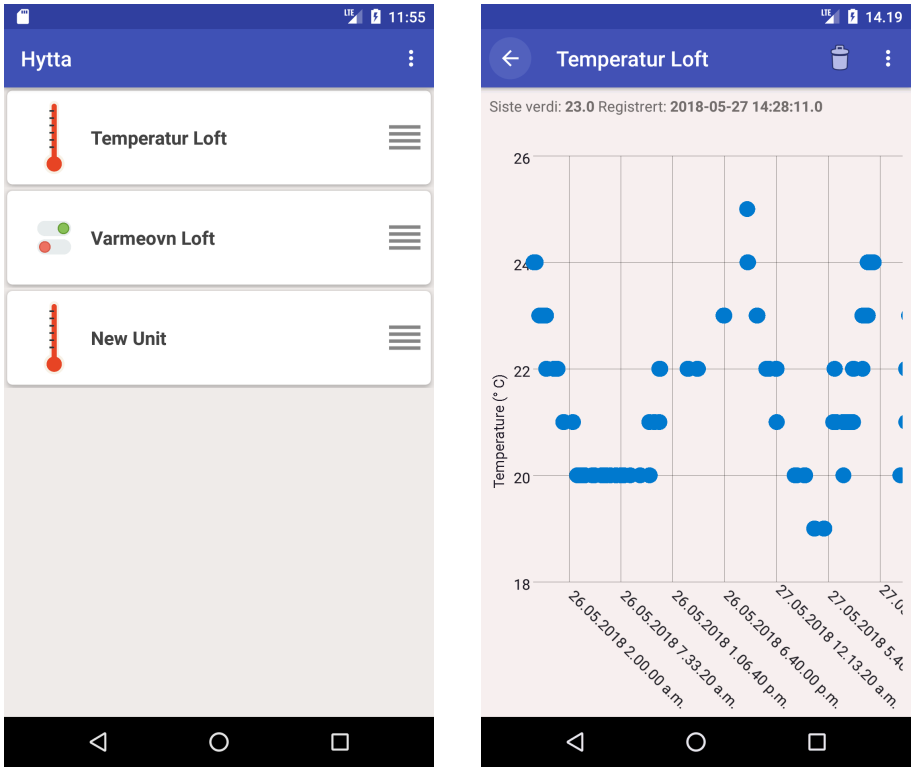
Kapittel 4

Android App

4.1 UI design

For å dekke kravene *IFK1*, *IFK5* må appen designes med brukervennlighet som mål. Det er valgt å benytte et "kort-basert" design, hvor hver enhet blir tildelt et "kort", som kan flyttes og dras rund på skjermen av brukeren, slik at han kan plassere det hvor han ønsker i listen over alle enhetene. Denne type layout er mye brukt og godt kjent for de fleste smarttelefon-brukere. Det antas derfor at navigeringen vil oppleves som lett å bruke for de fleste brukere. Ved å klikke på hver enhets kort åpnes et nytt vindu med informasjon om enheten og med muligheter for å gjøre endringer i konfigurasjonen, slette enheten etc. Dette layoutet er konstruert med Androids *CardView API* og *RecyclerView API* [23][24]. Sistnevnte er en metode for å framsitille data (her: enheter) fra et større datasett i en oversiktlig liste.

Når appen startes eller oppdateres blir listen fylt ut med enheter som er hentet fra tabellen *unitsConfig* i den online MySQL-databasen. Basert på hvilken type enhet det er, blir hver enhet tildelt et beskrivende symbol. Sammen med symbolet blir navnet til enheten plassert på et kort. Designet er vist i figur 4.1. En video av hvordan navigering i appen fungerer i praksis er vedlagt i vedlegg F. Det er i denne oppgaven ikke lagt særlig vekt på selve grafikken i appen, en del ting ser dermed noe røfft ut, og ikke nødvendligvis like pen som en ferdig kommersiell app ville ha sett ut. Likevel er UI-elementene som trengs på plass. Videre arbeid på appens UI nevnes i kapittel 7.



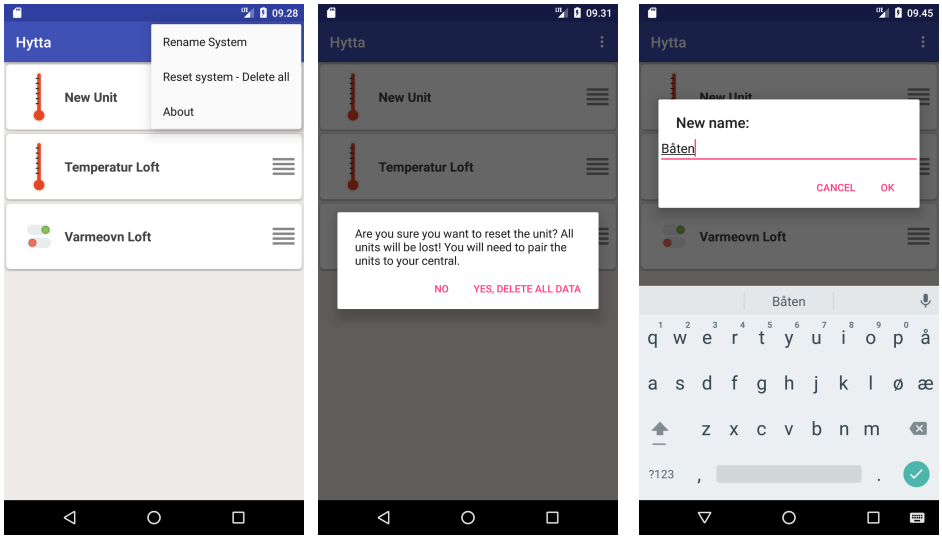
(a) Hovedbilde med kort-navigering

(b) Måledata fra en sensor

Figur 4.1: UI i appen

4.2 Konfigurering av systemet i appen

Som nevnt tidligere, og ifølge *FK4*, skal all konfigurering av systemet skje igjennom appen. Altså må det legges til funksjoner i appen for de konfigurasjonene som eksisterer i systemet i dag. På hovedskjermen er det i verktøylinjen lagt til menyvalg for å slette, samt å gi nytt navn til systemet. Ved å slette systemet menes å fjerne alle enhetene og dataene fra serveren. Etter alt er slettet vil appen sende en SMS til sentralen, slik at den kan oppdatere sitt interne minne med det som er oppført på databasen (ingenting, ettersom denne er tømt). Dersom sentralen er i nærheten av noen enheter som er i live må disse på nytt pares til sentralen, slik det er beskrevet i kapittel 3.2. Når en bruker trykker på menyvalg for å slette enheten, vil en egen meldingsdialog komme opp, hvor brukeren må bekrefte at han vil slette alle data. Menyvalgene og dialogen er vist i figur 4.2.



(a) Menyvalg **(b)** Sletting **(c)** Nytt navn

Figur 4.2: Menyvalg hovedskjerm

Menyvalget for å gi nytt navn til enheten vil gi en egen meldingsdialog, figur 4.2c, hvor brukeren kan skrive inn et nytt beskrivende navn på systemet, som for eksempel "Hytta", "Båten", eller "Campingvogna". Navnet blir oppdatert både i appen og på serveren. Slik prototypen er utviklet til nå, finnes ingen rammeverk for flere systemer, slik at det i praksis finnes kun ett system. Dermed har ikke valg av navn noen praktisk betydning, annet enn hva som vises på skjermen til brukeren. Om man videreutvikler systemet slik at en bruker kan registrere flere sentraler på samme app, vil det være nødvendig å skille disse med et unikt navn.

For hver enhet finnes også tilsvarende menyvalg når man går inn på den enkelte enhetenes underside ved å klikke på den. Her kan man endre enhetens navn på lik måte som hele systemet. Enhetens navn blir også oppdatert på både appen og i databasen. Her kan man også fjerne enheten fra systemet og slette all data fra enheten som er lagret i databasen. Brukeren vil også her få opp en meldingsdialoig for å bekrefte slettingen av data. Når enheten er fjernet vil brukeren måtte pare enheten på nytt for å bruke den.

4.2.1 Implementering og kildekode

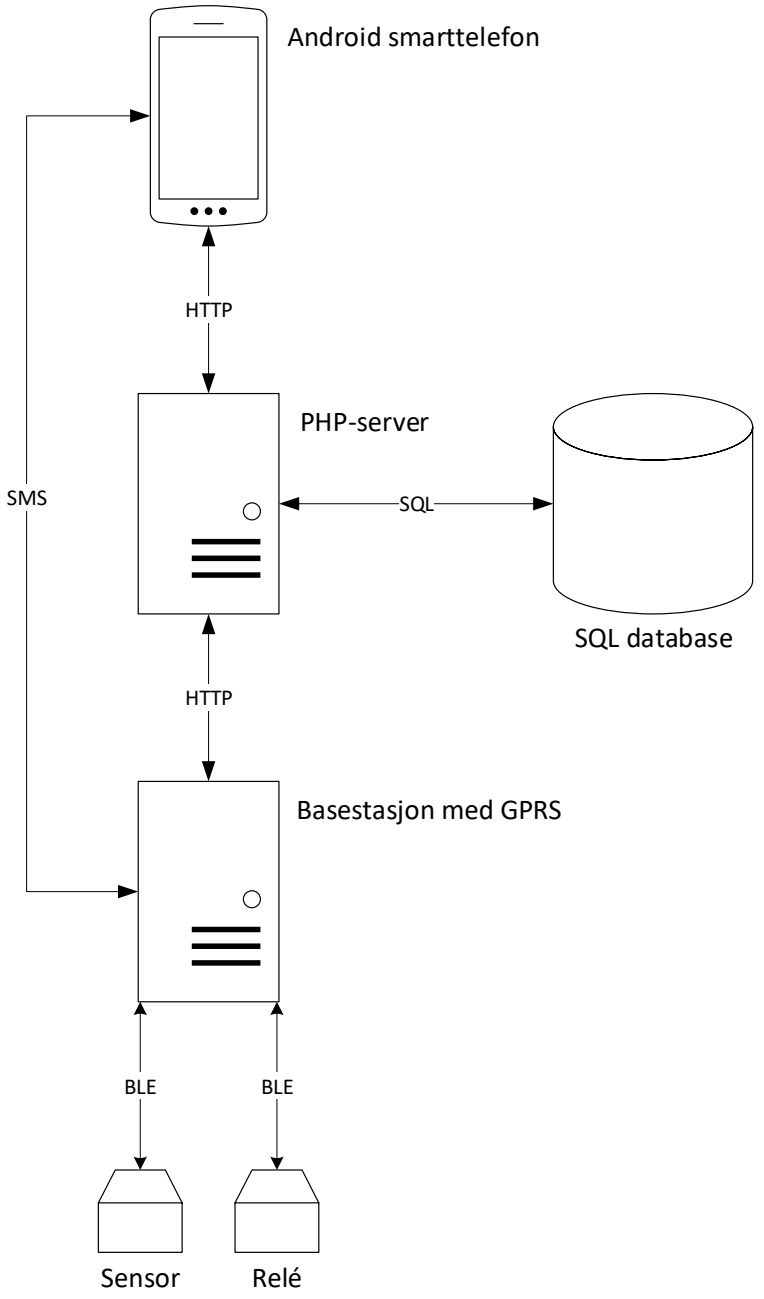
Android Studio er brukt for å utvikle appen til Android. Android Studio er et komplett IDE for utvikling og testing for Android. Hovedsaklig brukes programmeringsspråket Java for utvikling av apper, selv om det finnes muligheter for å bruke blant annet C og C++ ved bruk av Android NDK [25]. I denne oppgaven er alt programmert i Java. Android Studio leveres med en egen emulator for å teste apper i et virtuelt miljø. I tillegg til å teste på denne er appen som er utviklet også testet på en ekte Android mobiltelefon. Testingen har jevnt over gitt lik respons på både emulator og ekte smarttelefon.

Kildekoden til appen er hovedsaklig delt opp i to aktiviteter (*Activity* i Android). Disse aktivitetene er de to skjermbildene brukeren kan se, nemlig aktiviteten med oversikt over alle enhetene og aktiviteten til den enheten brukeren klikker på. I tillegg til disse to aktivitetene er det laget noen hjelpeklasser som blant annet håndterer utfylling av listen over enheter, og hjelpeklasser for håndtering av selve enhetene med tilhørende data. Grundigere dokumentasjon (JavaDoc) av koden og de enkelte klassene og funksjonene finnes i vedlegg D. Nettverksoperasjoner i Android bør ikke kjøre i samme tråd som resten av brukergrensesnittet, da dette kan resultere i at appen blir stående å vente på en nettverksoperasjon som kan ta lang tid. Derfor kjører alle nettverksoperasjoner i en egen tråd. Behandlingen av dette håndteres av et eget HTTP-hjelpebibliotek kalt Volley [26]. Dette biblioteket, utviklet av Android, skaper nye tråder for nettverksoperasjoner, og gjør det vesentlig enklere for utvikleren å lage nettverksfunksjoner uten å selv måtte opprette nye tråder (*AsyncTask* i Android) for hver operasjon.

Kapittel 5

Kommunikasjon

Figur 5.1 viser den overordnede kommunikasjonen i systemet, og hvilke protokoller som er brukt for overføring av informasjon. Figuren er omtrent lik den som ble presentert i prosjektoppgaven fra 2017 [1]. Systemet er designet slik at all informasjon vil logges til en online database. Denne vil også inneholde konfigurasjonsdata for systemet, både for appen og hele maskinvaresystemet. Når en bruker gjør en endring i appen vil dette lastes opp til databasen. Dersom denne endringen påvirker sentralen eller enhetene vil sentralen motta en SMS som varsler om ny konfigurasjon, og kan laste ned denne fra databasen.



Figur 5.1: Overordnet kommunikasjon i systemet

5.1 SMS

Et av hovedfokusene for tidligere prosjektoppgave [1] var SMS-kommunikasjon, mens det i denne oppgaven har vært fokus på å kommunisere via GPRS og internett. Likevel bruker systemet fortsatt SMS for å minimere datatrafikk over GPRS der det er nødvendig. En utfordring er nemlig at systemet skal gi en umiddelbar respons på brukerens kommandoer (jfr. *IFK4* og *FK5*). Altså må ting skje på systemet i rimelig kort tid etter at brukeren utfører en handling på appen (for eksempel endrer en enhet), og resultatet av dette lastes opp på serveren.

En løsning for å oppnå dette er å la sentralen kontinuerlig sjekke om det finnes nye oppdateringer på serveren. For å sikre god respons må denne sjekken utføres relativt ofte, noe som vil medføre et høyere databruk og kostnader for sluttbrukeren. Et annet alternativ er å kjøre en egen webserver på selve sentralen. Det vil sikre en god oppdateringsfrekvens, med et lavere dataforbruk, ettersom appen på brukersiden kan pinge sentralen. For å kunne kjøre en server på maskinvaren trengs en statisk IP-adresse, men med et vanlig SIM-kort har man en felles IP-adresse som er delt av mange brukere på mobilnettet. For denne løsningen må man altså ha en egen IP-adresse tildelt av mobiloperatøren, på egne SIM-kort beregnet for dette. Slike SIM-kort, med fast IP-adresse ment for produktutvikling gir ofte høyere månedspris, og høyere investering- og utviklingskostnader. Dette ville isåfall vært mot noe av oppgavens hensikt, å minimere kostnadene for brukeren. I et større videreutviklet kommersielt produkt vil denne muligheten kanskje på sikt være bedre.

For denne oppgaven ble det dermed besluttet å videreføre designet med SMS-kommunikasjon, og benytte dette på en slik måte at appen kan "pinge" sentralen når den trenger å hente en ny oppdatering fra serveren, eller gi kommandoer direkte i SMS-en. Hvis man antar at systemet er drevet med et enkelt kontantkortabbonement, er det i de fleste tilfeller er mer kostbart å bruke datatrafikk enn å sende en enkelt SMS fra brukerens telefon (de aller fleste har i dag ubegrenset SMS inkludert i sitt mobilabbonement), mens det er mer kostbart å sende en SMS fra kontantkortabonnementet. SMS-funksjonene fra prosjektoppgavens kildekode er i hovedsak ikke tatt med videre, men heller skrevet på nytt og implementert som en egen SMS-tråd i FreeRTOS på ESP32. SMS-tråden vil lytte etter innkommende meldinger og tolke disse. Om meldingen inneholder en kjent kommando vil den kunne utføre tjenesten. Dette er en viktig implementering fordi det muliggjør umiddelbar respons fra systemet uten at det trenger å sende HTTP-kall til serveren kontinuerlig.

Et annet funksjonsområde for SMS er muligheter for direkte alarmer til brukerens telefon, utenom appen. Enheter kan settes opp til å gi alermer om enkelte verdier kommer over eller under et visst nivå. I tillegg til at sentralen da rapporterer alarmer på serveren, kan det sendes en SMS slik at brukeren uansett får beskjed om hendelsen. Dette nevnes videre i kapittel 7.4.1.

5.2 Web-server

Web-løsningen som lagrer og formidler informasjonen mellom den sentrale enheten og Android er grunnleggende delt i to deler. En del er en SQL-server som inneholder all informasjonen om alle enhetene og systemet, og en PHP-serverer som serverer data til og fra databasen.

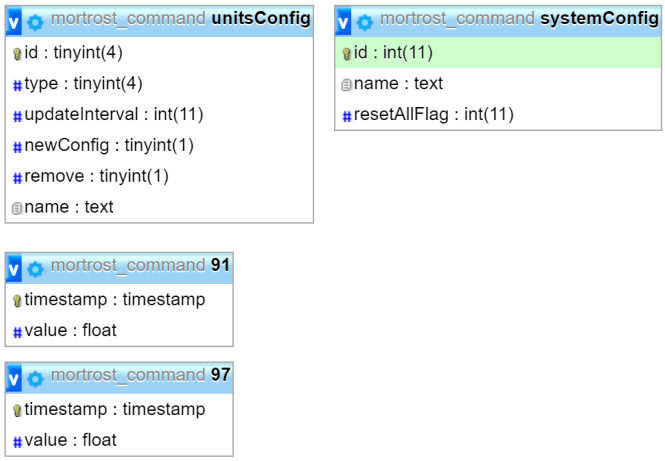
5.2.1 Valg av database

I tidlig fase av prosjektperioden ble flere typer databaser vurdert, og i hovedsak ble to typer databaser testet, Googles Firestore (kommersiell lisens) og MySQL (åpen kildekode GPL V2). MySQL er et SQL-basert databasesystem, og et av de mest brukte databasesystemene. Systemet er gratis, lett å bruke og gir høy ytelse, selv ved store databaser [27]. For oppgaven sin del er systemet også lett tilgjengelig, ettersom NTNU har en egen ferdig oppsatt MySQL-server som er tilgjengelig for studenter. Firestore er derimot Googles egen NoSQL skylagrings-løsning. Firebase er designet for å være fleksibelt og skalerbart, uten behov for "faste rammer" for datlagringen. Der SQL-baserte systemer lagrer all informasjon i egne forhåndsdefinerte *tabeller*, bruker Firestore heller et hierarkisk system med *dokumenter* sortert i *samlinger*. Et dokument kan ha en vilkårlig størrelse og inneholde ulikt antall variabler fra andre dokumenter i samme samling. Dermed kan databasen lettere endre seg dynamisk [28].

For dette systemte ble det valgt å bruke NTNUs MySQL-server for lagring av data. Dataene som skal lagres er lett kategoriserbare, hvor man kan ha én tabell for hver enhet, i tillegg til et par tabeller med konfigureringsdata. Slik systemet ser ut nå er det ikke behov for fleksibiliteten og skalerbarheten Firestore eller et annet NoSQL databasesystem kan tilby. Dessuten var det mulig å bruke om deler av databasesystemet som tidligere var utviklet for prosjektoppgaven [1].

5.2.2 MySQL database

Hvert system er tilknyttet en egen MySQL database. For tiden eksisterer kun én database, som er tilknyttet prototypen i denne oppgaven. I all hovedsak består databasen av informasjon om alle tilknyttede enheter. Hver enhet som blir koblet til får opprettet en egen tabell i databasen, med tittel lik ID-nummeret til enheten. For hver enhet lagres nye innlegg merket med tidsstempel og den aktuelle data som skal lagres. Ulike typer enheter kan ha ulike typer tabeller med forskjellige kolonner. I prototypen som er designet er det kun lagt inn én type tabell for en sensor eller relé, som kan benytte samme type tabell. Layout for hele databasen vises i figur 5.2.



Figur 5.2: SQL database layout, her med to enheter (ID = 91 og ID = 97)

I tillegg til tabeller for hver enkelt enhet er det lagt til to konfigurasjonstabeller. Én tabell inneholder generell informasjon om systemet, slik som navnet til hele systemet (e.g. "Hytta", "Båten", "Fjøset" etc.). Den andre konfigurasjonstabellen inneholder data om hver enkelt enhet i systemet. Dette kan være informasjon om navnet til enheten, og hvorvidt enheten er ny etc. Tabell 5.1 forklarer de ulike variablene i konfigurasjonstabellen til enheten.

Tabell 5.1: Variabler i unitsConfig

Variabel	Beskrivelse
id	Enhetens unike tall-ID
type	Hvilken type enhet
updateInterval	hvor ofte enheten skal kringaste
newConfig	Variabel som sier om enheten har fått ny konfigurasjon fra appen
remove	Variabel som sier om enheten skal fjernes fra systemet
name	Enhetens navn som bestemt av brukeren

5.2.3 Kall til databasen med PHP/HTTP

Databasen har begrensede tilganger. Derfor vil kommunikasjon direkte mellom ESP32-systemet og databasen være vanskelig, og dessuten gi begrensede muligeheter for utvidelser (e.g. sikkerhet og verifisering) senere i utviklingen. I stedet for å kommunisere direkte mot databasen med SQL-kall brukes derfor en "bro". En PHP-server er blitt satt opp med en rekke ulike PHP-script som kan hente informasjon fra og gjøre endringer i databasen. Som nevnt ovenfor muliggjør dette blant annet enkel utvidelse av systemet videre og ikke minst validering av ulike brukere og å begrense tilgangen til systemet. Mer om dette i kapittel 7.4.3.

Serveren er satt opp slik at hver operasjon som skal gjøres i systemet (legge til enhet, laste opp ny måling, gi nytt navn til enhet etc.) har sitt eget PHP-script. Scriptene kjøres med HTTP-kall sendt fra enten appen eller den sentrale enheten, alt etter hva som er intensjonen. I HTTP-kallene inkluderes den informasjonen scriptene trenger for å kjøre. For tiden er det som nevnt tidligere ikke lagt til en autentisering, men ved å bruke HTTP muliggjør man å senere implementere enten HTTP Basic Authentication eller den vesentlig sikrere HTTP Digest Access Authentication.

Vedlegg E inneholder alle PHP-scriptene som er i bruk i systemet. For testing og utvikling av prototypen er scriptene plassert på offentlig hjemmeområde på NTNUs servere. MySQL-databasen som brukes er også NTNU sin (`https://mysqladmin.stud.ntnu.no/`). Hvilken som helst PHP- eller SQL-server kan selvfølgelig brukes. SQL-serveren er meget skalerbar, og i praksis vil kun plassen tilgjengelig på serveren begrense hvor mye enhetsdata som kan lagres.

Kapittel 6

Resultater

Dette kapittelet tar for seg kravspesifikasjonen i tabell 2.1 og tabell 2.2, og hvorvidt disse kravene er oppfylt ved prosjektets slutt. Enkelte ting som ikke er fullført nevnes også videre i kapittel 7.

6.1 Evaluering av krav

FK1: Systemet må kunne vise sensordata og historikk på appen
Ved prosjektets slutt kan man installere appen på en Android smarttelefon, åpne appen og få umiddelbar oversikt over alle enheter som er tilkoblet systemet. Ved å klikke seg inn på en av enhetene får man informasjon om enhetens siste registrerte verdi, samt en graf som viser all data som er registrert så lang.

FK2/FK3: Systemet må kunne gi alarmer på appen/SMS
Appen gir ingen form for alarmer, det gis heller ingen alarmer på SMS. Dette har det ikke blitt tid til å implementere i løpet av prosjektperioden. Disse kravene har lavere prioritet enn andre, det er derfor fokusert på andre krav med høyere prioritet. I kildekoden til sentralen er det lagt til rette for at alarmer kan sendes på SMS ved hjelp av SMS-modulen. Alarmer beskrives nærmere i kapittel 7.4.1.

FK4: Systemet må være konfigurerbart via appen
I løpet av prosjektperioden er det lagt til de konfigurasjoner som trengs underveis. Det er for tiden mulighet til å gjøre følgende konfigurasjoner i appen:

- Skifte navn på systemet
- Tilbakestille hele systemet (slette all data)
- Skifte navn på individuelle enheter
- Slette individuelle enheter

Om man senere trenger nye konfigurasjonsmuligheter kan disse legges til ved følgende metode:

1. Lage et nytt PHP-script som utfører endringene som trengs på MySQL-databasen
2. Lage et menyvalg i appen som kjører et Volley HTTP-kall til PHP-scriptet
    (a) Sende SMS til sentralen om ny konfigurasjon dersom det er aktuelt og implementere endringene i kildekoden til sentralen og BLE-enhetene.

FK5: Systemet må ha støtte for at flere ulike enheter er koblet til samtidig
Sentralen kan være koblet til 255 ulike enheter samtidig, ettersom ID-en til enhetene er et 8 bit naturlig tall (0-255), hvor ID-en 0 er reservert ukjente, nye enheter. 255 enheter er nok tilstrekkelig for de fleste systemer, men om det er ønskelig med flere kan man utvide systemet til å håndtere 16 bit ID-er. Dersom man utvider ID-ene går dette på bekostning av advertisement-pakken til BLE-enhetene. Pakkene er begrenset til 31 byte, dessuten kreves mer strøm for å sende lengre pakker, så et lavere antall byte i advertisement-pakken er gunstig, slik det er nevnt tidligere i kapittel 3.4.

FK6: Systemet må kunne styre releer via appen (over SMS eller HTTP)
Denne funksjonen er ikke ferdig og fungerer ikke i appen. Det er lagt til rette for styring, ved at det kan sendes beskjed til sentralen både via SMS og HTTP når en bruker forsøker å styre et relé i appen, men så langt er det ikke lagt til funksjonalitet i sentralen som behandler disse forespørslene. Det som trengs videre er å registrere den nye statusen til relé-enheten, koble til enheten neste gang denne oppdages av sentralen, og skrive den nye statusen til enheten. Arbeid for å oppfylle kravet er altså påbegynt, men ikke ferdig.

IFK1: Appen skal være lett og intuitiv å bruke
Appen består av kun to skjermbilder, ett hovedbilde og ett bilde med informasjon om valgt enhet. Layoutet er valgt slik at det skal være kjent for de fleste som er vant til å bruke en smarttelefon. Appen har ikke mange knapper å trykke på, og for viktige valg, slik som å slette en enhet, vil brukeren få opp en bekreftelse-meny,slik at det ikke er lett å tilfeldigvis trykke feil.

IFK2: Det skal være lett å legge til og fjerne enheter
For å legge til en ny enhet trenger brukeren kun å skru på enheten (sette i batteri) og ha enheten i nærheten av sentralen. Enheten vil så dukke opp i brukerens app på telefonen. Det trengs ingen passord eller knappetrykk for å legge til enheter. Brukeren kan velge å gi nytt navn til enheten når den er lagt til i appen.

IFK3/IFK4: Skjermgrafikken og resten av appen skal ikke stoppe opp når data lastes inn, og systemet skal oppleves som kjapt og responsivt.
Appen er designet slik at all nettverksaktivitet foregår i bakgrunnstråder, og vil ikke stoppe skjermgrafikken når enheter og data lastes ned fra server. Noe krevende databehandling (sortering av enheter, sortering av enhetsdata etc.) på appen kjøres likevel på hovedtråden. For få enheter og mindre datamengder fungerer dette helt fint, men når databasene vokser med stadig flere målinger tar denne databehandlingen lengre tid, og man kan oppleve at appen går noe tregere. For å løse dette, bør enkelte deler av databehandlingen heller foregå i en egen oppgave (Android: *AsyncTask*). Det er likevel snakk om små forsinkelser.

IFK5/IFK6: Enhetene kan sorteres av brukeren på appen, og alle innstillinger skal vedvare når appen avsluttes
Brukeren kan sortere enhetene på start-skjermen. Disse endringene lagres lokalt på telefonen, og vil holdes fast også når appen startes på nytt. Andre innstillinger, som for eksempel enhetsnavn, lastes opp på databasen, og vil også vedvare når appen avsluttes.

6.2 Konklusjon av evaluering

Tabell 6.1 viser oversikten over alle kravene, og i hvilken grad de er oppfylt basert på oppsummeringen over. Merk at også de ikke-funksjonelle kravene er merket med godkjent eller ikke godkjent. Dette er noe upresist, da ikke-funksjonelle krav strengt tatt bør testes av noen utenforstående mennesker, for å få et godt bilde av hvordan systemet fungerer. Disse resultatene er dermed forfatterens egen oppfatning av hvor godt kravene er møtt. For videre arbeid bør kravene utsettes for grundigere testing. Det er også greit å merke seg at selv om tre krav ikke er oppfylt, er det påbegynt en del arbeid for å fullføre disse kravene, men på grunn av tidsmangel er arbeidet ikke ferdig. Dermed eksisterer det en del funksjonalitet knyttet til disse kravene i kildekoden som "venter på å bli brukt".

Tabell 6.1: Oppfylte krav

Krav	**Oppfylt**
FK1	Oppfylt
FK2	Ikke oppfylt
FK3	Ikke oppfylt
FK4	Oppfylt
FK5	Oppfylt
FK6	Ikke oppfylt
IFK1	Oppfylt
IFK2	Oppfylt
IFK3	Delvis oppfylt
IFK4	Delvis oppfylt
IFK5	Oppfylt
IFK6	Oppfylt

Kapittel 7

Videre arbeid

Som tidligere vist, er en del av kravene som er satt i forkant av denne oppgaven kun delvis møtt, og noen ikke fullført på grunn av manglende tid. Utenom oppgavens omfang finnes det i tillegg selvfølgelig mange forbedringer som bør gjøres på et slikt system før det kan fremstå som et ferdig produkt. Her følger en del forbedringer som forfatteren selv mener er naturlig å jobbe med i eventuelle videre versjoner av systemet.

7.1 Sentral enhet

7.1.1 Sjekk og oppdatering av systeminnstillinger

Når systemet starter opp lastes de siste innstillingene ned fra databasen. Dataene er ment å inneholde blant annet metadata om alle enhetene. Slik systemet er nå gjøres ingenting med disse dataene, og ingen av de lokalt lagrede enhetene blir endret ved oppstart. Dette resulterer i at det kan oppstå forskjeller mellom det som er lagret på databasen og det som er lagret lokalt i minnet til systemet. Dette kan igjen føre til problemer om for eksempel systemet prøver å laste opp data til en enhet som ikke finnes på serveren. Det nåværende systemet fungerer på en slik måte at dette ikke vil medføre noe annet enn en feilet nettverksoperasjon fra sentralen sin side, så vil systemet fortsette uforstyrret. Dette er ikke en katastrofal feil, men det medfører unødvendig datatrafikk og strømforbruk.

Et annet tilfelle der systemet må oppdateres er når enkelte endringer skjer i databasen. Om en bruker for eksempel sletter en enhet i appen vil denne endringen også skje i databasen. Appen sender da en SMS til den sentrale enheten om at en endring er skjedd på serveren. Sentralen mottar SMS-en men for tiden er det ikke lagt til noen respons til denne. Det som må gjøres videre er å lage en prosedyre for oppdatering av alle enhetene som er lagret. Denne prosedyren kan, enhet for enhet, sammenlikne med enhetene som finnes på serveren. En mulig bedre tilnærming vil være å slette alle enhetene som er lagret lokalt på sentralen og laste ned ny metadata for alle enhetene og legge til disse som nye enheter, ettersom data lagret i databasen alltid vil være oppdatert. Det bør vurderes hvilke av disse metodene som blant annet medfører lavest databruk.

7.1.2 Oppdatere enheter etter det som skjer på app/server

En annen funksjon som må implementeres på sentralen er hva som skjer når statusen til en enhet blir endret på appen. Når for eksempel et relé blir skrudd på på appen, vil denne endringen registreres på databasen. Ettersom sentralen ikke kontinuerlig sjekker databasen for endringer er man avhengig at appen sender en SMS til systemet om endringen som skal skje. Her kan appen sende en SMS til enheten med informasjon om hvilken enhet endringen gjelder og hva slags endring som må utføres. Så lenge samme informasjon utføres på databasen trenger ikke sentralen å koble til for å laste ned endringer online. Dette sparer datatrafikk.

7.2 BLE-enheter

7.2.1 Sette tidsintervall

For tiden er systemet slik at den sentrale enheten og BLE-enhetene har et hardkodet tidsintervall for hvor lenge BLE-enheter går i dvale mellom hver gang de kringkaster, og hvor lenge hver enhet kringkaster. Denne tiden må være kjent for både BLE-enheten og sentralen, så ingen informasjon går tapt, slik det er nevnt tidligere i kapittel 3.3.1. Ved videre utvikling bør dette tidsintervallet kunne settes av brukeren, slik at brukeren selv kan velge hvor ofte det kommer oppdateringer fra hver sensor. Dette påvirker selvfølgelig batteritiden, slik beskrevet i kapittel 3.4. For at brukeren skal kunne ta stilling til det kompromisset burde det utvikles en modell som sier noe om gjenværende batteritid basert på hvor ofte enheten kringkaster sin data. Batteristatusen kan da også kringkastes sammen med resten av advertisement-pakken og eventuelt gi en alarm til brukeren når den er kritisk, slik det er forklart i kapittel 7.4.1.

7.2.2 Tilrettelegging for styring av releer eller andre aktuatorer

Når en BLE-enhet blir oppdaget av den sentrale enheten, kan sentralen velge å koble til enheten, for eksempel om det er en ny ukjent enhet eller om det finnes ny konfigurasjon for BLE-enheten. Når de er tilkoblet kan de overføre data til hverandre. Datapakken som overføres er definert i kildekoden for både BLE-enheter og sentralen. Datapakken serialiseres for sending og konstrueres på nytt av BLE-enheten. For videre utvikling er det mulig å legge inn forskjellige data i disse pakkene, etter hva slags enheter og funksjoner som utvikles. Her er det altså mulig å sende informasjon til en relé-enhet om hvorvidt releet skal være åpent eller lukket, ved å kun sette et enkelt bit i pakken som sendes.

I BLE-enheten behandles disse dataene som mottas i funksjonen *on_write(...)*, og prosedyrer utføres basert på hva som er mottatt og hva slags type enhet det er snakk om. For videre arbeid er det derfor mulig å legge inn en prosedyre for styring av et relé basert på denne informasjonen. Alternativt til et relé kan man selvfølgelig styre en servo eller annen aktuator, også basert på en verdi som sendes i pakken.

7.2.3 Mer omfattende strømmålinger

Strømmålinger i denne oppgaven er kun utført med multimeter, det er derfor kun målt gjennomsnittlig strømforbruk over kortere perioder. I kapittel 3.4 vises det likevel at det med relativ sikkerhet kan forventes en levetid på over ett år for de fleste bruksområder. Skal man ha et mer nøyaktig estimat på batteritiden må batteristrømmen måles med oscilloskop, for å lettere se konsekvensene av å endre tidsintervall og liknende i koden. Et poeng for videre arbeid vil muligens være å på en eller annen måte å informere brukeren om forventet gjenværende batteritid på de enkelte enhetene. For at dette skal være mulig, må man ha mer nøyaktige måledata. Med det kan man utvikle en generell modell for å forutsi hvor lang levetid en enhet har ut ifra visse parametre brukeren kan sette (type enhet, oppdateringsintervall etc.).

7.3 Android app

Generelt for appen er det i denne oppgaven ikke lagt særlig vekt på det grafiske, utenom selve navigeringsflyten og appens generelle respons, og hvordan det føles å bruke den. Her er det selvfølgelig rom for et mer fengende brukergrensesnitt når det kommer til grafikk og ikoner. Ser man bort fra det rent estetiske, finnes det også en del funksjonelle forbedringer som det bør jobbes videre med, som det enten ikke har blitt tid til å ta tak i, eller som faller utenom oppgavens originale rammer.

7.3.1 Presentering av data

All data fra enhetene presenteres nå i en interaktiv graf på enhetens egen underside. Utenom denne grafen er det nå ikke mulighet for å se på dataene. Det betyr at om man skal se på sensordata fra en spesifikk tidsperiode er man nødt til å zoome og scrolle med fingrene for å finne frem til den riktige perioden. Her trengs altså et filter- og søkesystem hvor brukeren kan filtrere de data han vil ha fra grafen, for eksempel å finne perioder hvor temperaturen ligger under 0 grader. API-et som brukes for å tegne grafene, *GraphView*, har foreløpig ingen innebygde muligheter for å foreta en slik filtrering. En løsning er å gjøre filtreringen utenom GraphView API-et og skape et nytt undersett med de filtrerte dataene som så presenteres i en ny graf. I helt enkle tester gjort i denne oppgaven fungerer denne metoden tilfredstillende, men det må testes på større datasett for å se hva slags ytelse man kan få. Selv om metoden er testet er den ikke ferdig implementert i appen slik den er nå, på grunn av manglende tid.

I tillegg til å vise data fra enheten på enhetens underside i appen, bør det også vurderes å presentere aktuelle data på hovedsiden av appen. Hvert "kort" på forsiden kan presentere informasjon om siste måling, når denne er oppdatert, batteristatus etc. for å presentere brukeren for et umiddelbart overblikk over alle enhetene. Dette burde ikke være vanskelig å implementere, ettersom all data fra enhetene kan lastes ned med en gang appen lastes inn. Eventuelt kan man kun hente inn de data man trenger for å presentere på forsiden, for å senke innlastingtiden til appen. Dette er nok å foretrekke når databasene vokser med stadig mer historiske data.

7.4 Generelt

7.4.1 Alarmer

Den opprinnelige ideen som var grunnlaget for denne oppgaven inkluderer som sagt noe mer funksjonalitet enn det som er implementert her. Et eksempel på slik funksjonalitet er muligheten for at brukeren blir varslet når enkelte hendelser inntreffer. For brukeren bør det være mulig å stille inn en alarm for når verdier hos en sensor krysser en valgt terskelverdi. Dette må selvfølgelig være mulig å gjøre i appen. I kildekoden for sentralen er det lagt opp funksjoner for alarmhåndtering, men disse er ikke ferdig implementert. Ved hjelp av moduler som allerede eksisterer, kan disse funksjonene kan forholdsvis enkelt settes opp til å både sende SMS til brukeren når en alarm inntreffer, og varsle på appen via en endring i databasen. For at brukeren skal kunne oppfatte alarmene på appen som umiddelbare, må disse resultere i *push-varsler* på telefonen.

Slik appen er laget nå er det ikke mulighet til å motta push-varsler direkte fra serveren, uten at brukeren selv går inn i appen og manuelt henter data, noe som selvfølgelig virker mot sin hensikt spesielt når det gjelder alarmer. En vanlig løsning er å sende push-varsler til appen via Androids *Notification API* og *Push API*. Disse gjør det mulig for serveren å sende varsler direkte til riktig klient som kan vise push-varsler selv om applikasjonen er lukket eller i bakgrunnen på telefonen. I mange apper er det vanlig å benytte ferdig infrastruktur for dette, fremfor å utvikle det selv. Et vel brukt system er Googles eget *Firebase Cloud Messaging (FCM)*. FCM har god integrering med Android, og er designet til å fungere rundt Googles egen database-teknologi, Firebase. For denne appens tilfelle kan man da enten velge å flytte databasen fra et SQL-system over til et Firebase-system, eller kun lage en "mellomtjener" mellom hendelser på SQL-databasen (e.g. en alarm) og hendelse på en egen Firebase-database (e.g. "send push-notification").

7.4.2 Regulatorer

I tidligere prosjektoppgave [1] er det i designet nevnt muligheter for å implementere en regulator basert på verdier fra enkelte valgte enheter i appen. Tidlig i idéfasen for masteroppgaven ble det vurdert å implementere en slik regulator om det ble tid til det. Det er det ikke blitt, men det bør nevnes her at det fortsatt er en god idé som bør implementeres i fremtidige versjoner av systemet.

Som et eksempel på funksjonalitet bør det for brukeren være mulig å styre temperaturen vist i en gitt temperatur-sensor i appen ved å i appen "koble" denne til et relé som brukeren har knyttet til en varmeovn. Fra enhetens underside i appen må det da være mulig å stille set-temperatur, samt å skru selve regulatoren av og på. Selve reguleringen av temperatur avhenger selvfølgelig en del av ulike forhold slik som hvor stort rommet er og hvilken effekt varmekilden har. For fleksibilitetens skyld og med tanke på bruksområdet til appen burde det holde å implementere en enkel PID-regulator, hvor hver enkelt regulator kan til en viss grad kalibrere seg selv ved å se på stegresponsen til systemet. Det må selvfølgelig også vurderes hva slags krav systemt har til nøyaktighet, oversving og hurtigheten av reguleringen. For eksempel kan det være forskjellige krav til temperaturstyringen ved frostsikring i en båt, enn til styringen av luftfuktigheten i en kjeller.

7.4.3 Sikkerhet og flere brukere

I denne oppgaven er det ikke implementert noen som helst sikkerhet- eller personvernsfunksjonalitet. Det vil blant annet si at hvem som helst som har tilgang til serveren hvor PHP-filene ligger har mulighet til å benytte disse, lese eller endre data på databasen eller slette hele databasen. Dette er åpenbart ikke godt nok for et ferdig produkt, og videre må det implementeres autentisering på serveren slik at kun den enkelte bruker, eller rettere sagt appen til brukeren, og den aktuelle sentralen har tilgang til å endre på databasen. I utvidelsen av systemet må det implementeres et system for å håndtere flere brukere, hvor hver bruker er tilknyttet en egen database med brukernavn og passord. Denne brukeren må så tilknyttes en spesifikk sentral. Ettersom hver sentral allerede må ha sitt eget unike mobilnummer kan dette benyttes for å identifisere sentralen og knytte den mot en bestemt database.

Kommunikasjonen mot serveren, både fra sentralen og fra appen, kan relativt enkelt endres fra å benytte HTTP slik som i dag, til å kun sende kryptert trafikk, HTTPS. Dermed sendes ikke informasjon til og fra serveren i klartekst. Ettersom HTTP(S) brukes, kan Digest access authentication brukes for autentisering av informasjon sendt til og fra serveren [29]. Det er da mulig å autentisere brukerene (appen eller sentralen) som har kontakt med serveren, og dermed også autentisere den individuelle brukeren, om det er flere brukere av systemet.

Kommunikasjonen mellom BLE-enhetene og sentralen er også åpen, ukryptert, slik at "hvem som helst" som har kunnskap om protokollen som brukes har mulighet til å endre konfigurasjonen til enhetene, og i verste fall lage sin egen enhet som kan lure systemet. Dette kan ved første øyekast muligens ikke virke som en viktig detalj, men på denne måten kan en person potensielt styre enheter, og påvirke et helt system. Det er for eksempel mulig å sabotere et hus eller en hytte ved å skru av all varme på vinteren slik at vannrør fryser, eller om en bevegelses- eller posisjonssensor er koblet til systemet kan en person manipulere denne for å unngå å bli oppdaget ved innbrudd. En løsning på dette er å pare BLE-enhetene på en sikrere måte mot sentralen. Slik det gjøres nå, blir ikke enhetene ordentlig paret, de deler kun overfladisk informasjon med hverandre, slik at de senere kan "kjenne hverandre igjen". BLE-standarden bruker AES-CCM-kryptering, som ansees å være veldig sikkert. Selv om krypteringen i seg selv er sikker, må enhetene fortsatt overføre krypteringsnøkler med hverandre, i en prosess som ofte kan være utsatt for angrep. For overføringen av krypteringsnøkler brukes BLE-standardens parings-metode [30][31].

Ved bruk av BLE-standarden vil passord-verifisering for BLE-enheter ofte skje med at en enhet viser en sekssifret kode til brukeren, som så kan taste denne inn i en annen enhet. Med seks siffer blir én kode 20 bit. Enhetene verifiserer så koden ved å sende 20 meldinger med en kryptert datablokk med ett bit av koden i hver melding til hverandre. Dersom den sekssifrede koden er tilfeldig valgt hver gang, gir dette 1:1 000 000 sjanse for at en angriper kan gjette riktig paringskode for hvert forsøk. En utfordring med denne verifiseringen når det kommer til systemet vårt er at hverken BLE-enhetene eller sentralen har skjerm og knapper, slik at passordet er nødt til å være statisk. Dette forutsetter for det første at den statiske koden aldri er kjent av noen brukere. Dessuten gir det en potensiell angriper en mye større sjanse til å gjette koden. Med en statisk kode på 20 bit, trenger en angriper kun 20 paringsforsøk på å gjette koden, ettersom han kan få tilbakemelding på minst ett bit ved hvert paringsforsøk. Statisk-passord-systemet kan forbedres ved å for eksempel øke et tidsintervall for mulighet til paring mellom hvert mislykkede paringsforsøk, men angrep kan fortsatt skje over lengre tid [31].

For vårt system kan en løsning være å bruke en fysisk knapp på sentralen som må være holdt inne når en enhet skal pares. Dette brukt i sammenheng med en statisk paringskode vil gi tilstrekkelig sikkerhet, ettersom en angriper da nødvendigvis må kunne gjette riktig paringskode og forsøke å pare nøyaktig samtidig som en bruker holder inne knappen på sentralen, som vi antar er plassert utilgjengelig for angriperen.

Litteratur og kilder

[1] M. Rostad, "Overåking og kontroll av miljøvariabler i fritidsbolig og båt," Desember 2017.

[2] Espressif, "ESP32 Development Board." [Online]. Available: https://www.amazon.com/Makerfocus-NodeMcu-32S-Bluetooth-Development-NodeMCU-32S/dp/B01MDVNB87

[3] eProLabs, "SIM808 Development Board." [Online]. Available: https://www.eprolabs.com/wp-content/uploads/2016/04/WLC-0019_2.jpg

[4] Nordic Semiconductors, "nRF51 DK Overview and documentation," 2017. [Online]. Available: https://www.nordicsemi.com/eng/Products/nRF51-DK

[5] SSB, "Bygningsmassen," jan 2017. [Online]. Available: https://www.ssb.no/bygg-bolig-og-eiendom/statistikker/bygningsmasse/aar/2017-02-22#content

[6] M. Farstad, J. F. Rye, and R. Almås, "Fritidsboligfenomenet i Norge," 2008. [Online]. Available: https://www.regjeringen.no/globalassets/upload/krd/vedlegg/rega/fritidsboligfenomenet-i-norge---rapport.pdf

[7] KNBF, "Båtlivsundersøkelsen 2012," 2017. [Online]. Available: http://knbf.no/images/Presentasjoner/KNBF_-_Rapport_komplett.pdf

[8] A. H. Amundsen and T. Bjørnskau, "Bruk av fritidsbåt i Norge," 2017. [Online]. Available: https://www.toi.no/getfile.php?mmfileid=45470

[9] S. Neverdal, "Artikkel i Finans Norge," November 2017. [Online]. Available: https://www.finansnorge.no/aktuelt/nyheter/2017/11/milliarderstatninger-etter-vannskader/

[10] M. Rosbach, "Artikkel i Sunnmørsposten," Januar 2014. [Online]. Available: http://www.smp.no/forbruker/article9039363.ece

[11] Espressif, "ESP32 Datasheet V2.0," Desember 2017. [Online]. Available: http://espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[12] Simcom, “SIM808 Introduction and documentation,” 2017. [Online]. Available: http://simcomm2m.com/En/module/detail.aspx?id=137

[13] SIMCom, “SIM800 Series AT Command Manual V1.09,” 2015. [Online]. Available: https://www.elecrow.com/download/SIM800%20Series_AT%20Command%20Manual_V1.09.pdf

[14] Espressif, “ESP-IDF Programming Guide,” 2017. [Online]. Available: https://dl.espressif.com/doc/esp-idf/latest/

[15] B. Lovosevic, “LibGSM,” 2017. [Online]. Available: https://github.com/loboris/ESP32-PPPOS-EXAMPLE/tree/master/components/pppos

[16] A. Dunkels and L. Woestenberg, “lwIP,” 2017. [Online]. Available: http://lwip.wikia.com/wiki/LwIP_Wiki

[17] Nordic Playground, “nrf5-ble-tutorial-characteristic,” 2016. [Online]. Available: https://github.com/NordicPlayground/nrf5-ble-tutorial-characteristic

[18] Nordic Semiconductor, “nRF51 Series Reference Manual,” 2018. [Online]. Available: http://infocenter.nordicsemi.com/pdf/nRF51_RM_v3.0.pdf

[19] S. B. Sverrisson, “nRF51 Series Current Consumtion Guide,” 2015. [Online]. Available: https://devzone.nordicsemi.com/tutorials/b/hardware-and-layout/posts/nrf51-current-consumption-guide

[20] Nordic Semiconductor, “nRF51 Development Kit user Guide V1.1,” 2016. [Online]. Available: http://infocenter.nordicsemi.com/pdf/nRF51_DK_UG_v1.1.pdf

[21] Forum user: Akhil, “Nordic DevXone: "Significant packet loss at lower battery voltage",” 2016. [Online]. Available: https://devzone.nordicsemi.com/f/nordic-q-a/12345/significant-packet-loss-at-lower-battery-voltage

[22] M. Jensen, “Coin cells and peak current draw,” 2010. [Online]. Available: http://www.ti.com/lit/wp/swra349/swra349.pdf

[23] Android Developers, “Card View API,” 2018. [Online]. Available: https://developer.android.com/reference/android/support/v7/widget/CardView

[24] ——, “Recycler View API,” 2018. [Online]. Available: https://developer.android.com/reference/android/support/v7/widget/RecyclerView

[25] ——, “Android NDK,” 2018. [Online]. Available: https://developer.android.com/ndk/

[26] ——, “Volley,” 2018. [Online]. Available: https://developer.android.com/training/volley/

[27] Oracle Corporation, “MySQL 8.0 Reference Manual,” 2018. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/

[28] Google Developers, “Firestore Documentation,” 2018. [Online]. Available: https://firebase.google.com/docs/firestore/

[29] Wikipedia, “Digest access authentication,” 2018. [Online]. Available: https://en.wikipedia.org/wiki/Digest_access_authentication

[30] M. Bon, “A Basic Introduction to BLE Security,” 2016. [Online]. Available: https://eewiki.net/display/Wireless/A+Basic+Introduction+to+BLE+Security

[31] H. Chaskar, “IoT Security Using BLE Encryption,” 2017. [Online]. Available: https://www.networkcomputing.com/wireless-infrastructure/iot-security-using-ble-encryption/555332330

Vedlegg

Alle vedlegg ligger i en egen mappe i zip-fil med navn **vedlegg_master_rostad.zip**. Zip-filen finnes her (02.06.2018): `https://www.dropbox.com/s/kmyyhu5p8jltmfz/vedlegg_master_rostad.zip?dl=0`

A Kildekode til sentral

Denne mappen inneholder kildekoden til sentralen. For å kunne kompilere denne må et eget ESP-IDF-miljø settes opp. Dette beskrives godt i følgende lenke: `http://esp-idf.readthedocs.io/en/latest/get-started/windows-setup.html`

Mappen **ESP32-master** må deretter legges inn under brukermappen før koden kan kompileres og lastes opp på et ESP32-kort med kommandoen *make flash*. Dersom man ønsker å kun inspisere kildekoden i egen tekstbehandler finnes denne under mappen **.\ESP32-master\main**. Her er koden delt opp i moduler, som vist i figur 3.2.

B Kildekode til BLE-enheter

Denne mappen inneholder all kildekode for BLE-enhetene. Kildekoden befinner seg i filene **main.c**, **our_service.c** og **our_service.h**. For å lese kildekoden trengs kun en vanlig tekstbehandler.

Det er også mulig å åpne prosjektet i μVision. Prosjektfilen finnes i mappen **.\ble-units\pca10028\s130\arm5_no_packs**. For å kompilere prosjektet og laste dette opp på et nRF51 DK-kort må benyttede biblioteker linkes på nytt i prosjektet, da dette er avhengig av hvor prosjektet ligger plassert på disken.

I mappen **.\ble-units\hex** finnes også ferdig kompilert design, som er klart til å lastes opp på et nRF51 DK-kort.

C Kildekode til Android app

Denne mappen inneholder et helt Android Studio prosjekt med all kildekode for Android applikasjonen. Det skal være mulig å laste inn hele prosjektet i Android Studio, sist testet med versjon 3.1.1. Dersom man ønsker å kun se kildekoden i egen tekstbehandler finnes denne i mappen **.\Master\app\src\main\java\com\example\mortrost\dragdrop**

D Dokumentasjon til Android app

Denne mappen inneholder JavaDoc dokumentasjon for kildekoden til appen. For å lese åpnes filen **index.html** i en nettleser.

E PHP-filer

I denne mappen finnes alle PHP-script som kjøres på serveren. Disse kan åpnes i en vanlig tekstbehandler. Merk: Av sikkerhetsårsaker er passordet til databasen sladdet i filen **connect.php**.

F Video av appen

Videoen som er vedlagt viser de vanligste funksjonene til appen. En sirkel merker tastetrykk på skjermen. Videoen er tatt opp på en Xiaomi 5s Android smarttelefon, ved hjelp av appen *DU recorder*.

G Tidligere prosjektoppgave

Denne mappen inneholder en forberedende prosjektoppgave [1], skrevet av undertegnede høsten 2017 på NTNU.