NTNU
Kunnskap for en bedre verden

Kontroll og overvåkningsplattform Antibac

PROSJEKTOPPGAVE – IELET2002

GRUPPE 6

NTNU
Kunnskap for en bedre verden

KANDIDATER (ETTERNAVN, FORNAVN):			
Bonde, Håvard; + 4 Andre (Fjernet andres navn før jeg deler dette Online.)			
DATO: 17.12.2020	FAGKODE: IELET2001	GRUPPE (NAVN/NR.): Gruppe 6	ANT SIDER: 38

FAGLÆRER:

TITTEL:
Kontroll og overvåkningsplattform Antibac - Rapport

INNHOLD

Innhold

1. FORORD 4
2. INNLEDNING 5
3. BESKRIVELSE AV PRODUKTET 6
4. TEKNOLOGIER BRUKT, ESP32 7
   1. Hardware 7
   2. Software 9
5. SYSTEMOPPBYGNING/ARKITEKTUR 11
   1. ESP - koblingsskjema 11
   2. ESP - flytskjema 12
   3. Kommunikasjonskoder 13
   4. Kommunikasjon - flytskjema 14
      1. Ny klient kobler til server 14
      2. Klienter liste 15
      3. Sending av data ESP 16
      4. Vedlikeholdsmodus (Maintenance) 17
   5. Datatrafikk 18
   6. Nettside 19
   7. Server 22
   8. Database 24
   9. Raspberry Pi programvarer 26
6. BRUKERMANUAL 27
   1. Nettside 27
   2. Dispenser 28
7. VURDERING AV BRUKTE LØSNINGER 29
8. KONKLUSJON/ERFARING 33
9. REFERANSER 34
10. VEDLEGG 36
    Vedlegg 1: Utstyrsliste 36
    Vedlegg 2: Wireshark-analyse 37

1. FORORD

Vi ønsker å takke Girts Strazdins (faglærer i IELET2001 og førsteamanuensis ved Institutt for IKT og realfag, NTNU) for veiledning og oppfølging under hele prosjektarbeidet. Vi ønsker også å takke Adrian Heyerdahl (vitenskapelig assistent ved Institutt for elektroniske systemer, NTNU) for veiledning knyttet til de tekniske utfordringene rundt prosjektet.

2. INNLEDNING

Denne prosjektoppgaven er en del av emnet IELET2001 Datakommunikasjon ved elektroingeniørstudiet på NTNU og omfatter IoT, mikrokontrollerprogrammering, oppsett av server-klient-kommunikasjon, prosjektstyring og gruppesamarbeid. Datakommunikasjon er en sentral del av elektroingeniørstudiet, og har utviklet seg til å bli nærmest essensiell kunnskap for et stadig økende antall elektroingeniører innen diverse fagfelt.

Hensikten med prosjektet er å anvende vår kunnskap om datakommunikasjon, mikrokontrollerprogrammering og prosjektstyring. Målet er å ferdigstille en funksjonell prototype av en plattform som diverse småsystemer kan styres fra, og hvor brukere kan ha en oversikt over alle systemer. Plattformen skal bestå av en server og en nettside, samt et eller flere mikrokontrollersystemer som kommuniserer med serveren.

Rapporten tar i hovedsak for seg løsningen som gruppa har kommet frem til, men fokuserer også på teorien og metodene som er blitt brukt for å ferdigstille produktet.

3. BESKRIVELSE AV PRODUKTET

Covid-19 pandemien har ført til økt fokus på smittevern, derfor har vi valgt å lage en prototype for kontaktløs bruk av en pumpedispenser. Vi ønsker å fokusere på hånddesinfeksjon og derfor har modellen fått navnet Antibac-dispenser, men den kan også brukes for håndsåpe eller andre produkter. Produktets hovedfunksjon er å gi ut hånddesinfeksjon uten at brukeren må berøre flasken. Pumpingen styres av en bevegelsessensor. Den fungerer enkelt ved at man holder en hånd under dispensertuten, og så bruker man den andre armen til å aktivere sensoren. Da vil dispenseren gi ut en dose med hånddesinfeksjon, og dette skal være nok til begge hendene. Etter hvert pump vil det bli tatt målinger av nivået i dispenseren, som deretter sendes til en nettside, der man kan se nivået i flasken. På Nettsiden vil det også være mulighet for å stoppe modellen uten å skru den helt av, slik at man kan utføre vedlikehold. Prototypen er satt sammen av tre deler. En nettside, en server og en fysisk modell.

Den fysiske modellen består av pumpedispenseren, en ultralydmåler, en trykkmåler, en servomotor og en ESP32 microkontroller. Ultralydmåleren venter på at noe skal komme innenfor et satt område. Vi har valgt 10 til 30 cm fra sensoren. Hvis noe kommer innenfor dette området, vil servomotoren bevege seg 90 grader frem og tilbake en gang. På motoren er det festet et tau som trekker ned dispenserpumpen slik at den gir fra seg en dose. Etter hvert pump så leses det av en trykksensor, som er plassert under dispenseren. Denne måler nivået som er igjen i flasken og sender nivået via en ESP32 til en server, som videre lagrer dataen i en database.

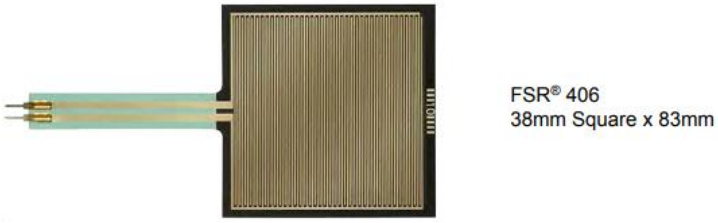
Serveren står for kommunikasjon mellom mikrokontroller, nettsideklienter og database. Som database har vi brukt Firebase. Dette er av typen dynamisk (noSQL) database. Tjenesten kommer fra Google og all data lagres i en nettsky. Serveren er satt opp på en Raspberry Pi. Server vil loggføre alle klienter som er koblet til. Nettsideklienter vil sorteres i eget rom: ‘website’, mens mikrokontrollerklienter vil sorteres i rom: ‘esp’. Dette gir mulighet for sortering av datatrafikk. Hver mikrokontrollerklient vil også bli tildelt et navn. Server og database jobber sammen på en måte som gjør at klienter vil lagres dynamisk. Dette gjør at man teoretisk sett kan ha så mange klienter tilkoblet som man ønsker. Data blir filtrert og sortert på en måte som gjør at man vet hvilken data som tilhører hvilken klient.

På nettsiden vil man se en liste over alle dispensere/klienter som er tilkoblet serveren. Her kan man velge klienten man ønsker lese data fra. Nettsiden har en logg der man ser en liste over målinger fra valgt klient. Her vises nivå, dato og klokkeslett for hver måling. Nettsiden har også en graf over nivået over tid, samt et stolpediagram som viser antall trykk på flasken de siste åtte dagene. Man kan bruke dette til å analysere trafikk. I tillegg har nettsiden muligheten til å stoppe/starte servoen til klienten. Hvis man trykker «stop» vil klienten gå i vedlikeholds-modus. Dispenserens servo vil da ikke kunne kjøre og nye målinger vil ikke bli tatt. Man kan derfor trygt utføre vedlikehold eller skifte flaske/fylle på innhold uten at det skaper noe søl eller går ut over loggføring av nivå.

4. TEKNOLOGIER BRUKT, ESP32

1. Hardware

FSR - Force Sensing Resistor



Figur 1: Force sensitive resistor for nivåmåling (Interlink Electronics, 2020)

FSR består av en tykk polymerfilm (PTF) som gir lavere resistans ved økt trykk mot overflaten. Denne trykksensitiviteten er optimalisert for bruk i automatisert elektronikk, medisinske systemer, industrielle regulatorer og roboter. Resistansverdi uten trykk over seg, er høyere enn 10MOhm. Aktuasjon ved trykk fra 0,2N opp til 20N. Kontinuerlige verdier innenfor gitt sensitivitetsområde. Sensoren tilkobles analog avlesning oppstilt i en spenningsdeler. (Interlink Electronics, 2020)

HC- SR04 Ultrasonic sensor
Sensor som måler avstand og registrerer blokkerende objekter.

Figur 2:HC-RS04 ultralydmåler (Components101, 2017)

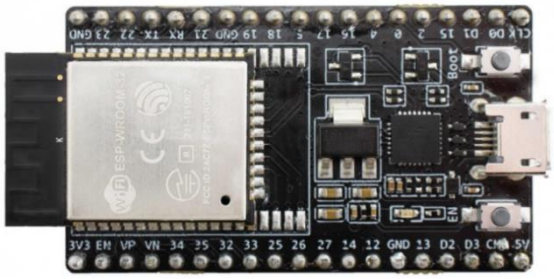
Ultrasonic har fire pins; inngangsspenning, jord, trigger og ekko. Anbefalt inngangsspenning er typisk +5V. Sensoren måler i praksis mellom 2 til 80 cm, med en nøyaktighet på +/- 3mm. Brukes ofte i applikasjoner som måler avstand, eller skal registrere et objekt. En innebygd ultralydsender avgir en ultralydbølge som reflekteres tilbake av objekter som kommer i veien. Den reflekterte ultralydbølgen fanges opp av en innebygd ultralydmottaker og avstanden er beregnet ut fra hastighet og tid. Sensoren bruker en lydbølgefrekvens på 40Hz. Det finnes mange bibliotek i flere kodespråk for denne sensoren, som utfører avstandsberegningene og tillater å stille inn range for det området som ønskes. (Components101, 2017)

Micro servo MG90S Metal Gear Servo

Figur 3 - MG90S Metal Gear Servo

Liten og lett servomotor med stor ytelse. Har tannhjul i metall for økt styrke og holdbarhet. Motoren kan rotere omtrent 180 grader, 90 grader i hver retning. Det finnes mange bibliotek tilgjengelige, og dette er en fin servomotor for nybegynnere og tar liten fysisk plass. Posisjon '0' er i midten, og den har mulighet til 90 grader rotasjon, til hver side. Dreiemoment fra 1,8 kg/cm ved 4,8V opp til 2,2kg/cm ved 6V. Rotasjonshastighet fra 60 graders rotasjon på 0,1s ved 4,8V til 0,08s ved 6V. (Datasheetspdf, 2014)

ESP32-DevkitC V4 utviklingskort



Figur 4 - ESP32-DevkitC V4

ESP32-DevkitC V4 er et utviklingskort (PCB) som er bygd opp rundt en ESP32 WROOM 32D-chip. Denne ESP32-chipen har følgende egenskaper: støtte for WiFi (2.4GHz) og Bluetooth, dobbel kjerne: Xtensa 32-bit LX6, en co-prosessor som forblir påslått i deep-sleep modus, og utganger med flere bruksmuligheter på en og samme I/O-pin. Disse egenskapene gir dermed mange muligheter for rask og pålitelig kommunikasjon, til og med trådløs, samt standby uten høyt energiforbruk. (Espressif Systems, 2020a)
Utviklingskortet kobles til enheten med programkoden, via en USB A- til en USB B-kabel for å laste opp koden. Denne tilkoblingen vil også fungere som spenningsforsyning. Om man ønsker så kan man i stedet bruke en ekstern spenningskilde via pins for 5V og jord, *eller* 3,3V og jord etter at koden er ferdig opplastet.
Enkelte pins brukes internt på utviklingskortet i kommunikasjonen mellom ESP32-chipen og SPI flash-minnet og man bør unngå å bruke disse til noen annet. Dette gjelder D0, D1, D2, D3, CMD og CLK. (GPIO's 6-11) (Espressif Systems, 2020b)
Ellers finnes utganger som har andre innebygde egenskaper slik som kapasitive berøringssensorer, Hall-effektsensorer, SD-korttilkobling, Ethernet, høyhastighets SPI, UART, I2S og I2C.
Bruksområdene for utviklingskortet er mange. Man kan oppnå enorme kommunikasjonsområder rent fysisk grunnet integreringen av WiFi, og kortere avstander gir mulighet for bruk av Bluetooth og Bluetooth LE. Strømforbruket i dvalemodus er mindre enn 5uA og utviklingskortet er derfor egnet til drift med batteri og i bærbare elektroniske innretninger.
(Espressif Systems, 2020c) (Espressif Systems, 2020d)

2. Software

Arduino C, Arduino Programming Language (APL)

Figur 5 - Arduino IDE. Skjermdump

Programmet som brukes til å skrive Arduino C-kode, og for å sende koden fra programmeringsenheten til mikrokontrolleren på utviklingskortet, kalles **Arduino IDE**. Dette er et gratis program med åpen kildekode og dermed er tilgjengelig for alle. Lastes ned fra Arduino sine nettsider.

Det mest brukte programmeringsspråket for Arduino utviklingskort, kalles **Arduino C** eller **Arduino Programming Language** (APL), og er basert på C og C++. Vi har brukt APL for å programmere utviklingskortet vi benytter som klient i socketkommunikasjonen. (Kjell & Company, 2017a)
Bruk av ESP32 med Arduino IDE krever en installering av et ekstra bibliotek for å få lastet opp koden, da denne mikrokontrolleren ikke ligger i grunnversjonen til IDE'en. (Randomnerdtutorials, 2016)

Arduino C kodeprogrammer kan deles inn i tre hoveddeler, struktur, verdier og funksjoner. Under verdier ligger variabler og konstanter. Strukturen består av to hovedfunksjoner, setup og loop. Setup kjøres kun én gang ved oppstart av programkoden. Her bestemmes startverdi for variabler som skal ha det, pinMode bestemmes og her kjøres oppstart av biblioteker og eventuelt seriellmonitor. Loop gjør som navnet lyder, kjører sitt innhold i kontinuerlig loop, så her legger man inn de funksjonene man ønsker at skal kjøre mer enn én gang/ kontinuerlig.

Datatyper i C-språk refererer til et omfattende system som brukes til å deklarere *variabler* og *funksjoner*. Variabeltypen avgjør hvor mye plass den tar i minnet og hvordan bit-mønsteret som lagres blir tolket. Variabler i C-språk holder et område som kalles skop. Et skop er *en del* av programkoden og det er tre ulike plasser hvor variabler kan deklareres: Inne i en funksjon, som lokal variabel. I definisjonen av funksjonsparameterne, som kalles formelle parametere. Utenfor alle funksjonene, som global variabel.
Operatorer er symboler som gir beskjed om å utføre spesifikke matematiske eller logiske beregninger. Innebygde operatorer utfører matematikk, sammenligning, boolsk logikk, bitvis logikk og sammensatt logikk.

Konstanter (konstante variabler) tar mindre plass i dynamisk minne enn variabler, konstanter er derfor bra å bruke der det er mulig. Konstanter lagres i flashminnet(programminnet), og alt som lagres der vil frigjøre plass i dynamisk minne (SRAM) til andre variabler. (Kjell & Company, 2017b)
C/C++ -mikrokontrollerbibliotekene utgis under LGPL-forskriften, kildekoden for Java-oppsett utgis under GPL-forskriften.

Kontrollbetingelser er følgende elementer; if betingelser, if-else betingelser, if- else- if betingelser, switch case betingelser, og betingelsesoperatoren '?' (som er den eneste treverdige spørreoperatoren i C-språk. Løkkebetingelser lar oss utføre en betingelse eller en gruppe betingelser flere ganger. I Arduino C finner vi *while løkke*, *do-while løkke, for løkke,*

og *nøstet løkke*. Setter man ingen sluttbetingelse vil løkken aldri ta slutt og programkoden vil forbli der, dette heter *uendelig løkke*.

For å strukturere programkoden i delkoder som utfører individuelle oppgaver og hendelser bruker man funksjoner. Disse brukes typisk ved behov for å utføre samme oppgave flere ganger i samme programkode. Denne inndelingen av programkode i funksjoner gjør feilsøking lettere, ettersom man kan teste hver funksjon enkeltvis og se akkurat hvor feilen ligger. Endringer gjøres dermed færre steder om man kun trenger endre elementer i én funksjon, og ikke gjennom hele programkoden. Inndeling med funksjoner gjør også programkoden mer lesbar og mer kompakt.

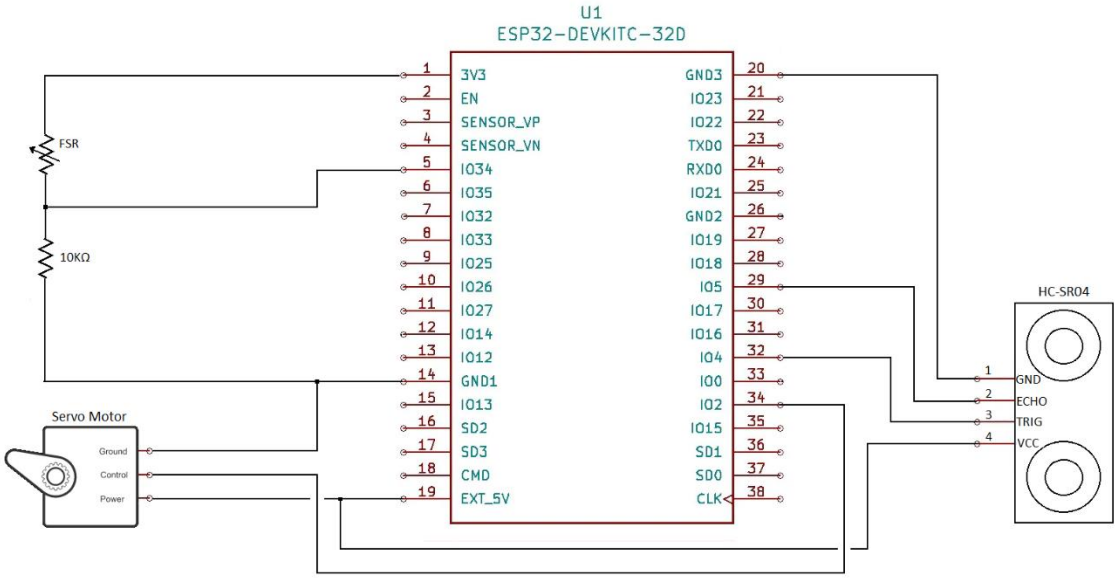
For å lagre tekst brukes (tekst)strenger. Disse kan benyttes til å vise tekst på en LCD-skjerm eller i IDE seriellmonitor. Strenger kan også lagre inngangsvariabler, eksempelvis om noen skriver bokstaver eller lignende på et tastatur som er tilknyttet utviklerkortet eller IDE'en. ***To typer strenger*** brukes i Arduino C-språk; ***String character array*** og ***String***. String character array er en *liste* med tegn; datatype **char**, dette tilsvarer strenger som brukes i C-språk. Arduino C sin datatype **String** lar oss bruke en *tekststreng som et objekt* i programkoden.

I innebygde C-språkbiblioteker i IDE'en finnes innebygde funksjoner for å manipulere *lister*, ellers kan man skrive egne funksjoner som gjør det. Bruk av 'strlen' og 'sizeof' vil gi *ulikt antall elementer* én og samme liste inneholder. ***Strlen*** teller ikke med den avsluttende '0' som avslutter strengen, i motsetning til ***sizeof*** som teller denne også. ***Sizeof*** gir derfor alltid én mer (den teller *hele* listen) enn ***strlen*** (som teller *kun strengen* inne i listen) når de teller opp i listen. Blir ikke den angitte listelengden overholdt vil man kunne få problemer å kjøre programkoden. Dette fordi en lang streng plassert i en for kort liste vil kunne gjøre at minnet overskrives av de overflødige elementene i strengen.

For andre datatyper enn char brukes også lister, betegnet Array. Liste (Array) er fortsatt en samlet gruppe elementer med hver sin indeksplassering. Lister krever plass i minnet og deklareringen av listen gjør at det reserveres en tilpasset lagringsmengde. Deklarasjon som *reserverer minne* kalles definisjon. Listestørrelsen må være et heltall større enn 0, og man kan bruke en løkkebetingelse til å plassere elementer i en liste.
(Tutorialspoint, 2020)

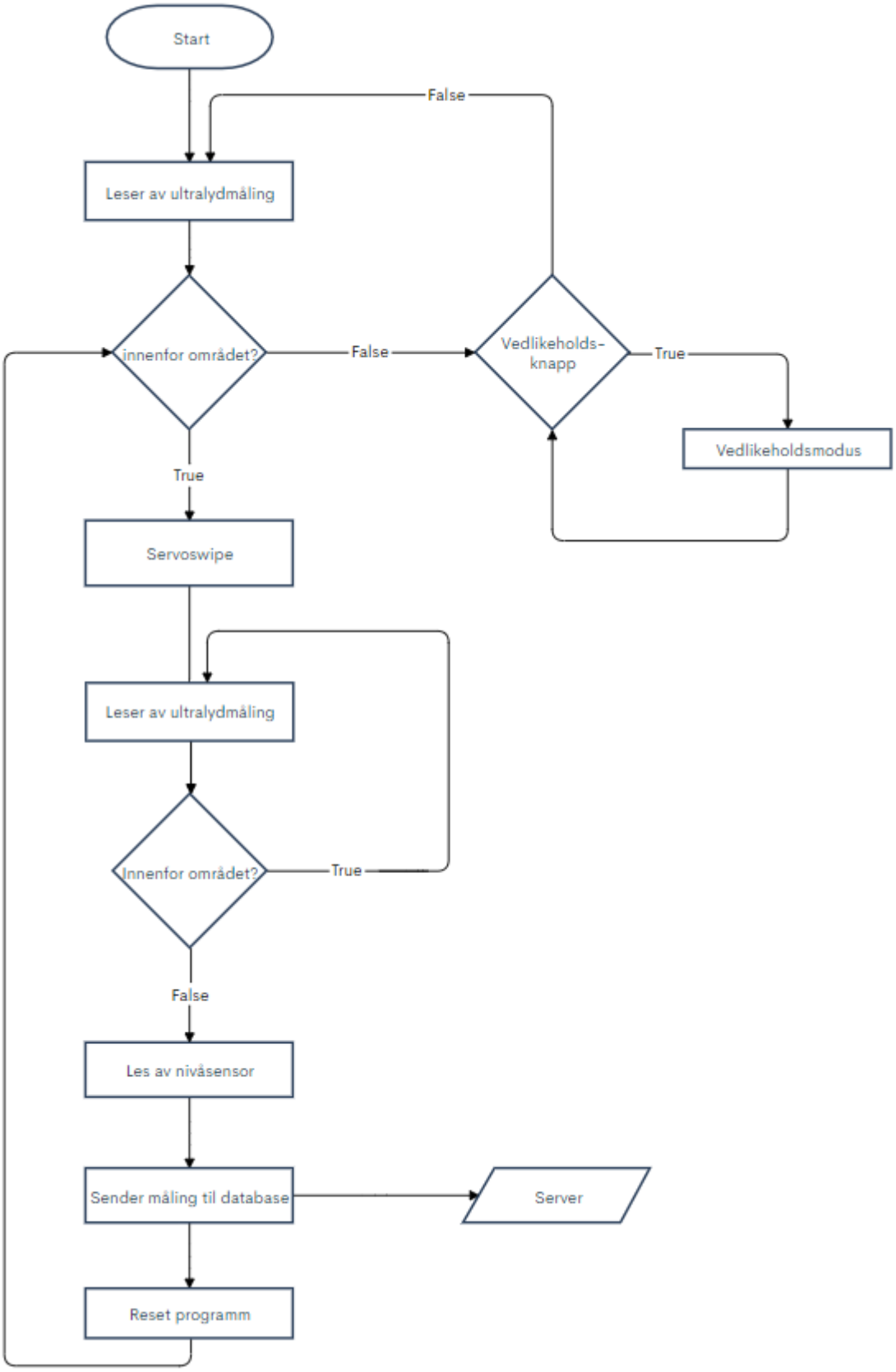
5. SYSTEMOPPBYGNING/ARKITEKTUR

1. ESP - koblingsskjema



Figur 6: Koblingsskjema ESP32 med utstyr

2. ESP - flytskjema



Figur 7: Flytskjema for ESP32 Dispenser programmet

3. Kommunikasjonskoder

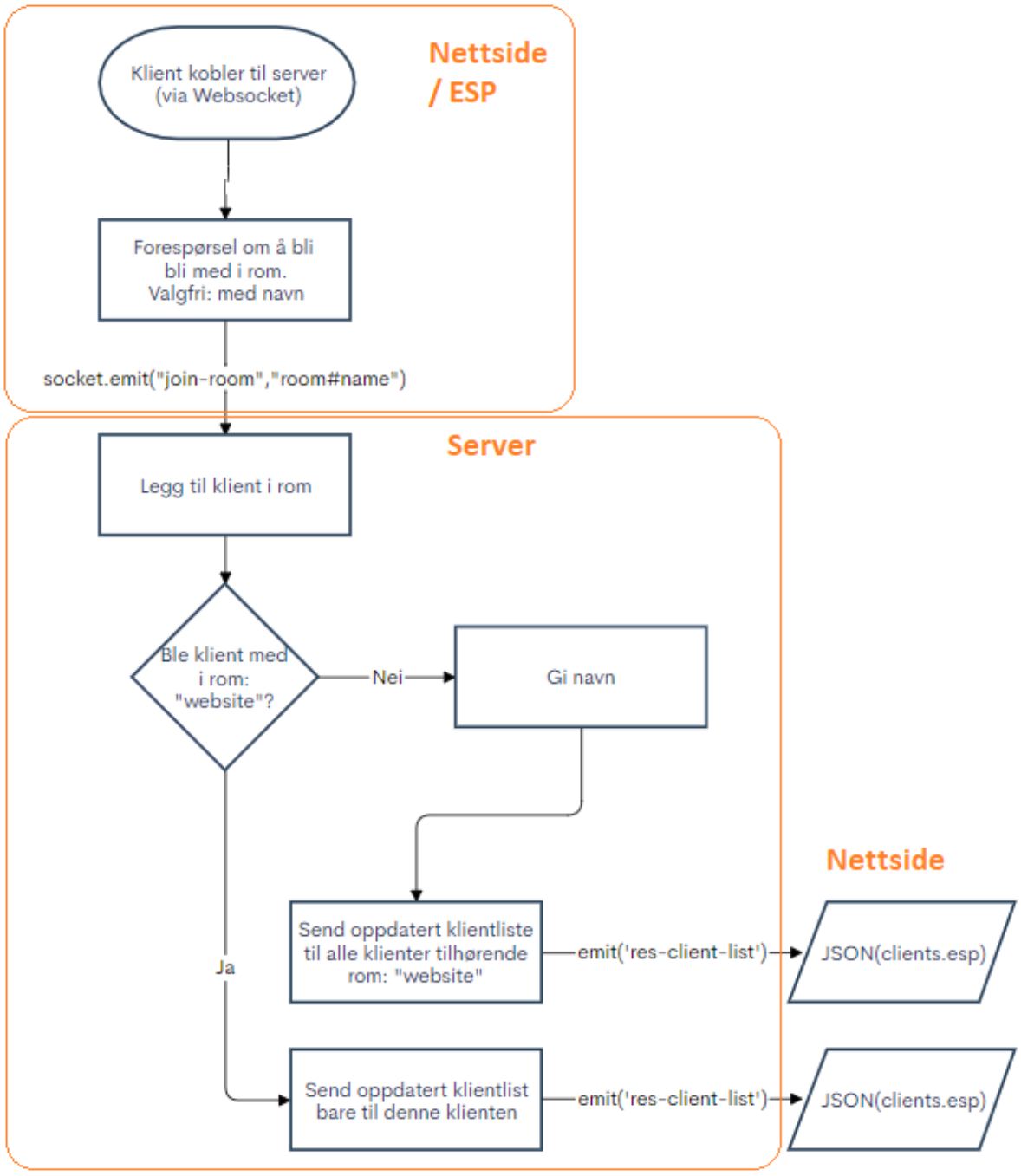
Liste over kommunikasjonskoder ('identifiers') brukt i websocket-kommunikasjon

Identifier	Fra	Til	Beskrivelse
Bootup			
«join-room»	Klient	Server	Forespør og bli med i rom. Rom blir brukt for å filtrere datatrafikk. Valgfri: legg med navn. Dette navnet vil vises på nettsiden. Data i format: «rom#navn» eller «rom»
«req-client-list»	Nettside	Server	Nettside spør server etter å oppdatere klientliste på forsiden. Denne forespørselen skjer ved opplasting av nettsiden.
«res-client-list»	Server	Nettside	Sender full liste over ESP-klienter koblet til systemet. Hvis klient har forespurt liste vil denne sendes kun til klient som spør. I tillegg vil denne listen bli sendt til alle klienter i rom «website» hver gang en ESP-klient kobler til eller fra server.
Ny datapakke (Live-data)			
«res-data»	ESP	Server	Hver gang servoen til dispenseren har kjørt en gang vil en ny datapakke sendes til server. Server vil lagre denne pakken til firebase.
«data->website»	Server	Nettside	Server har fått ny datapakke fra en klient gjennom «res-data». Dataen videresendes til alle nettsider slik at logg og grafer kan oppdateres real-time.
Data (Firebase)			
«req-data-full»	Nettside	Server	En nettside spør etter all data fra en spesifikk klient.
«res-data-log»	Server	Nettside (*)	Server responderer til «req-data-full». Sender full fra siste 3 dager. Maksimum 'X' antall indekser. Dette kan settes i config.js
«res-data-barchart»	Server	Nettside (*)	Server responderer til «req-data-full». Sender liste over antall trykk siste 8 dager. Inkludert i dag.
«res-data-linechart»	Server	Nettside (*)	Server responderer til «req-data-full». Sender data fra siste 24 timer.
Annet			
«maintenance»	Nettside	ESP	Stop/Start av servo. Knapper ligger på forsiden. Nettside sender til server hvilken klient som skal stoppes/startes. Server sender til gjeldende klient og ber den stoppe/starte servo.

Req: Står for request (forespørsel).
Res: Står for response (respons)
(*) Sender kun til klienten som forespurte data.

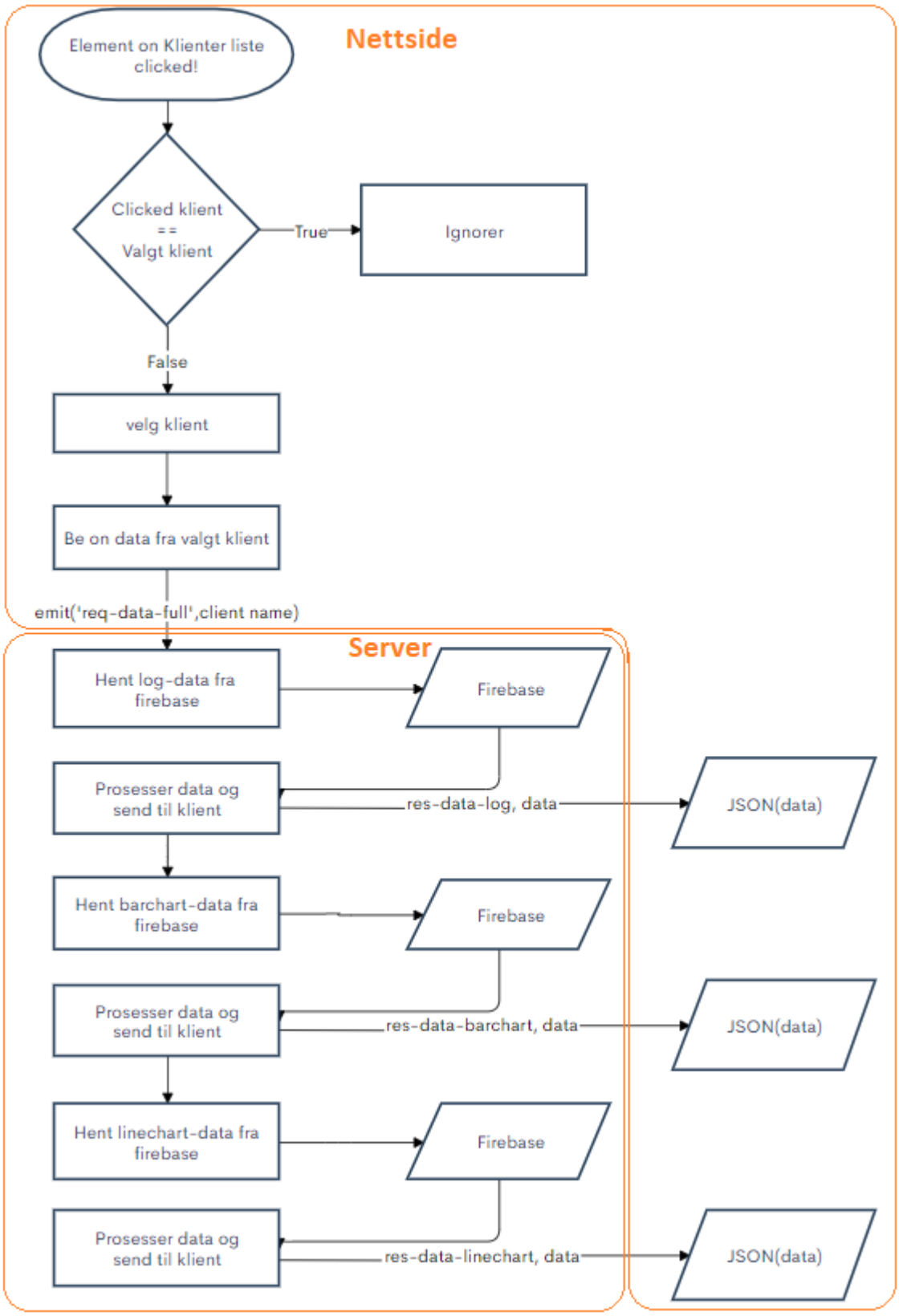
4. Kommunikasjon - flytskjema

1. Ny klient kobler til server



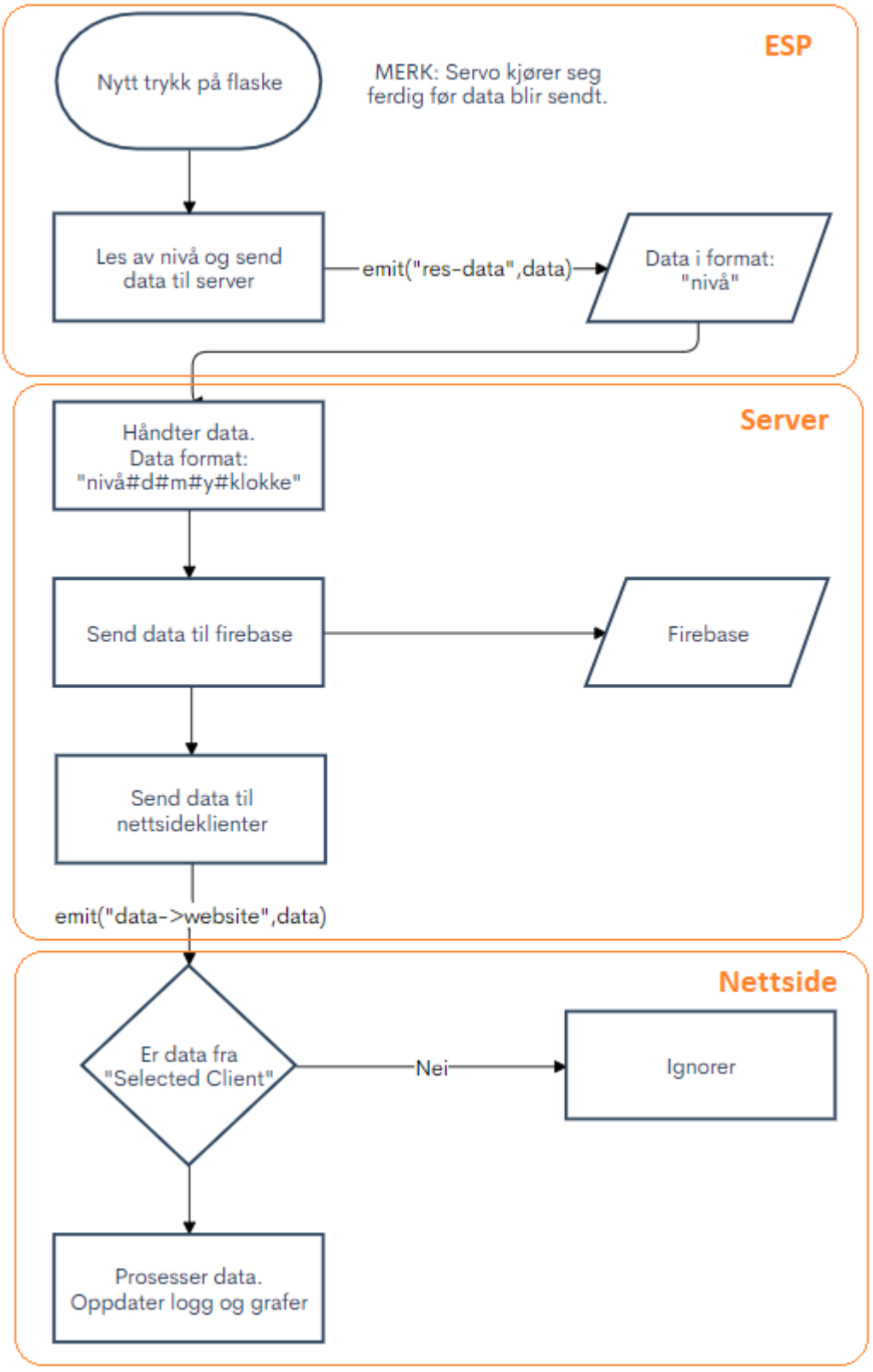
Figur 8: Flytskjema, ny klient kobler til server.

2. Klienter liste



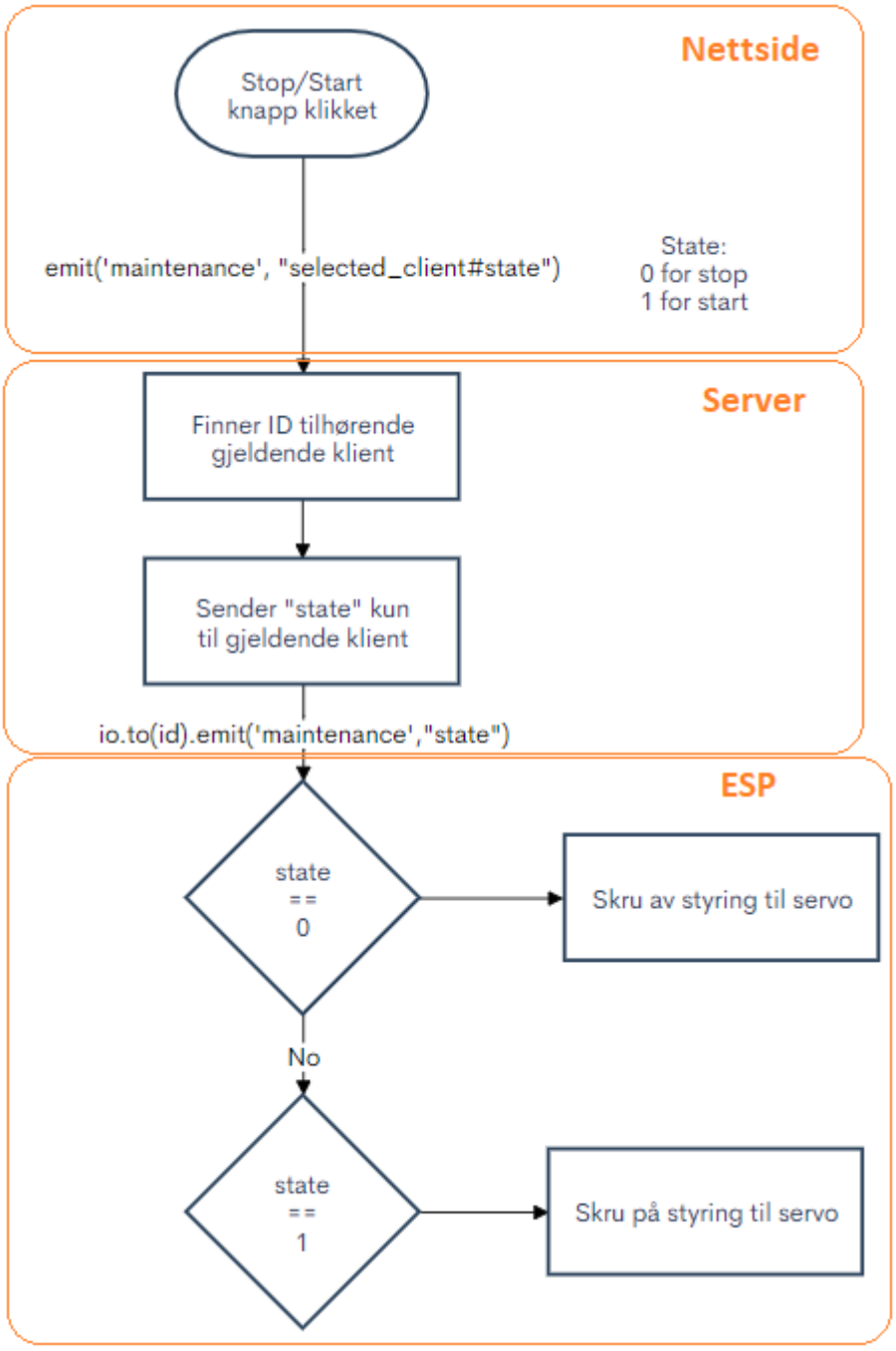
Figur 9: Flytskjema, klienter liste.

3. Sending av data ESP



Figur 10: Flytskjema, sending av data ESP.

4. Vedlikeholdsmodus (Maintenance)



Figur 11: Flytskjema, vedlikeholdsmodus

5. Datatrafikk

Rom
Alle klienter som kobler til server, bør også oppgi hvilket rom det ønsker å koble til.
Hvis klienten er av type ESP eller annen mikrokontroller, anbefales det og også oppgi navn.
Dette gjøres ved hjelp av kode: socket.emit(«join-room», «rom#navn»).
I Dispenser biblioteket gjøres dette automatisk ved funksjon: Dispenser::init().

Serveren aksepterer bare to forskjellige rom: «esp» og «website».
Rom blir brukt for å sortere datatrafikk og dermed begrense hvor mye data som blir sendt.
For eksempel ønsker vi kun å sende data tilhørende klientliste, logg og datavisusualisering kun til nettsider og ikke til mikrokontrollere.

Klientliste
Alle klienter som kobler til et rom, vil lagres i klientlisten på server.
Strukturen til listen er gjort på følgende måte:

```
var clients = {
  esp: [
    { clientId: "192068cjo20jicon", clientName: "Chuck Norris"},
    { clientId: "d10291090j0cnc0d", clientName: "Antiac-dispenser"},
  }],
  website: [
    "127068djo15jiuon", "d10292075a0cne2d", // ONLY CLIENTID
  ]
};
```

Figur 12: Objekt for å holde oversikt over tilkoblede klienter. Backend på server.

Merk at klienter i rom ESP har både ID og navn, mens nettsiden har bare navn.
ESP navn vises på klientlisten til nettsiden, og ikke ID. Dette er for å gjøre nettsiden mer brukervennlig.

Begrensninger
Enkelte løsninger har vist seg å ha en del begrensninger. Disse blir tatt opp under «Vurdering av brukte løsninger» senere i rapporten.

6. Nettside

Generelt
Valgte å lage nettsiden fra begynnelsen av i stedet for å bruke eksempelkode. Dette delvis fordi vi ønsker noe skreddersydd for dette prosjektet, og delvis fordi jeg ønsker å legge til JS, CSS og HTML i «toolbox'en».

Libraries og Frameworks
Til nettsiden er det brukt følgende Libraries:

1. JQUERY — Brukt til å velge elementer og til å utføre animasjoner.
   JQUERY gjør DOM (Document Object Model) lettere å jobbe med.

```
let element = document.getElementById('#myID'); // Uten JQUERY
let element = $('#myID');                       // Med JQUERY
```

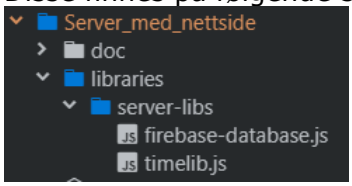
Figur 13: JQUERY eksempel

2. Bootstrap — Brukt for å gjøre CSS kode lettere. Dette rammeverket kommer med flere forhåndsdefinerte classes. Dette gjør styling av knapper, NAVBAR osv. mye lettere å gjennomføre.

3. chart.js — Er et bibliotek brukt for plotting av grafer. I dette tilfellet er det brukt til stolpediagram og Linechart.

4. SocketIo — Nettsiden må ha tilganger til en fil fra socketIo biblioteket for å koble seg til server via websocket.

Splitting av filer
Vi har splittet opp våre skreddersydde JS filer til 3 filer for å gjøre prosjektet lettere leselig. Disse finnes på følgende sted i direktivet:

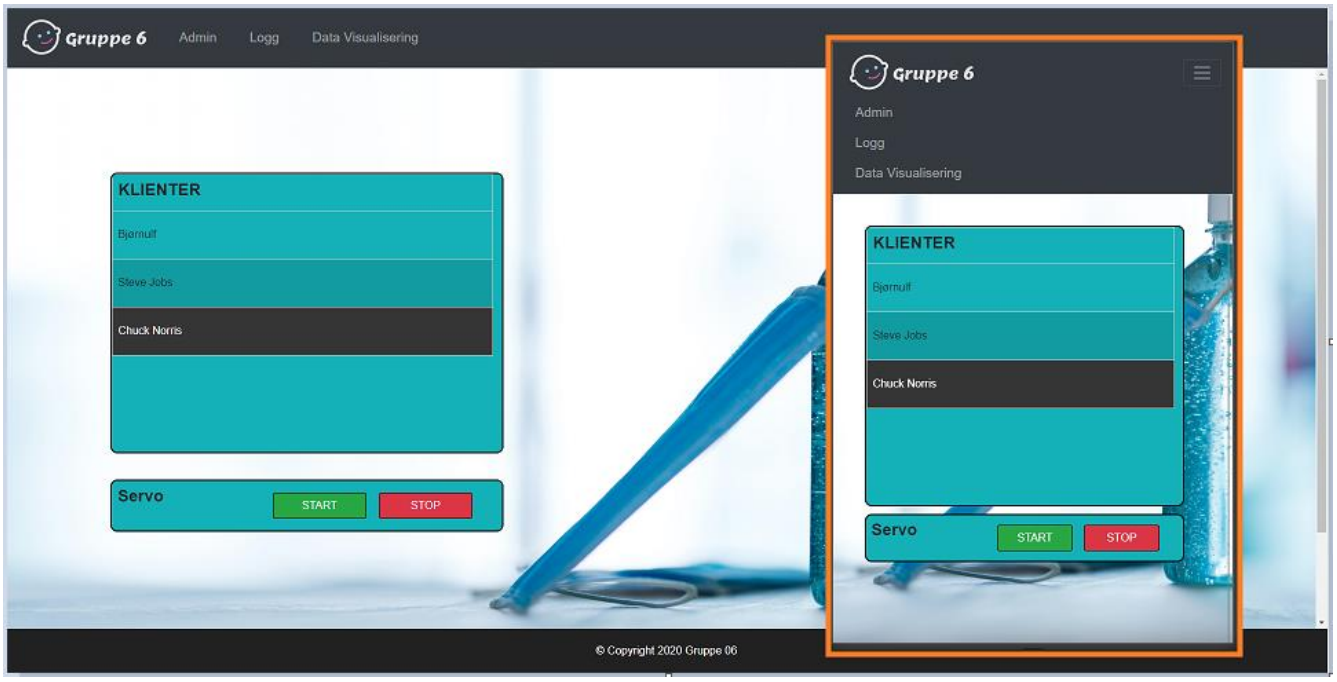


Figur 14: Path til egendefinerte libraries

Her er index.js hovedfila. Denne inkluderer alt som har med Websockets og kommunikasjon og gjøre.
Animations.js tar for seg animasjoner som navigasjonsbar m.m.
Plots.js tar for seg plotting av grafer. Disse finner man under [Data Visualisering] på navigasjonsbar.

Linking av filer og Libraries
All linking av filer og biblioteker er gjort direkte i index.html vha. script-tags.
Enkelte filer er hentet via CDN. Merk at disse er hentet fra forskjellige steder på internett. Dette gjør det lettere for programmerer å sette opp prosjektet. Men det har en ulempe: hvis stedet filen er hentet fra er nede, eller filen du prøver å hente er fjernet eller oppdatert kan dette føre til problemer. Det vil derfor være en bedre løsning permanent og importere disse inn i prosjekt-folderen i stedet. Dette er mer pålitelig.

«Responsive»
Nettsiden er kodet på en slik måte at den skal være relativt brukervennlig også på telefon. Blant annet vil navigasjonsbaren på toppen kollapse til en hamburgermeny ved bruk av telefon. Klientloggen vil forandre størrelse fra Desktop, til nettbrett, til telefon.

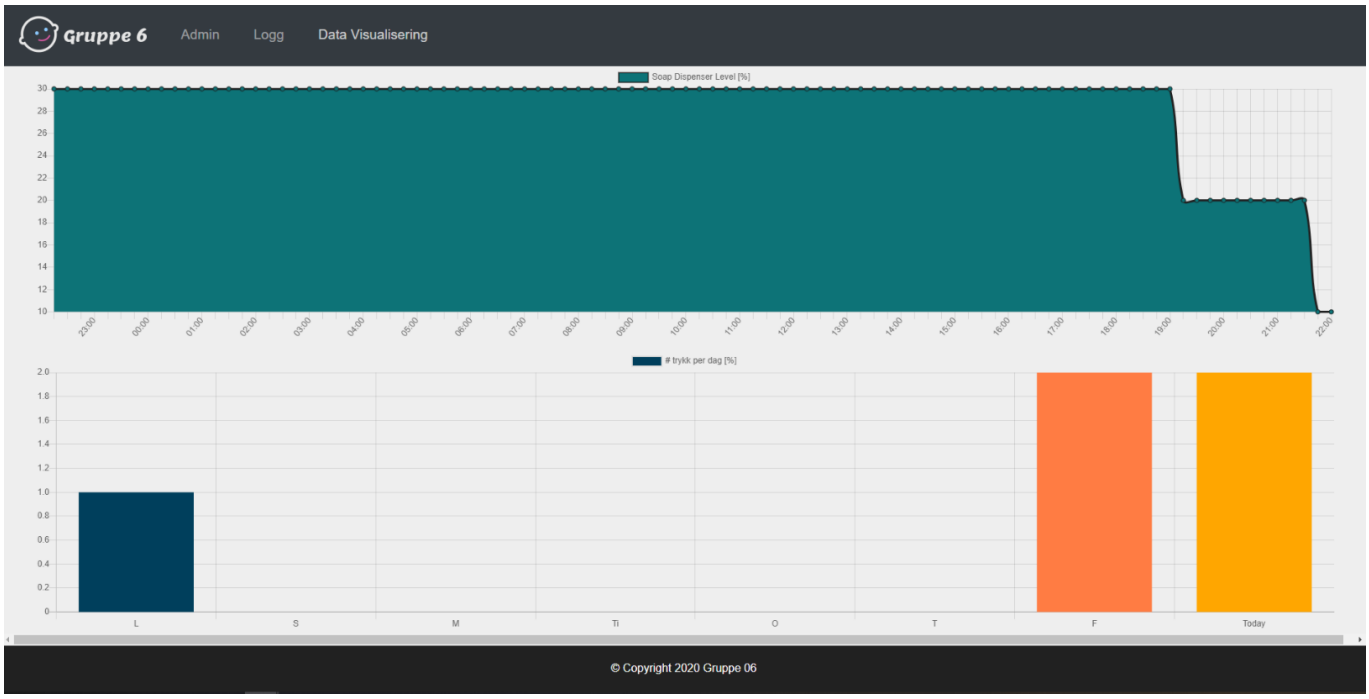


Figur 15: Forhåndsvisning av nettsiden. Her demonstrert hvordan forsiden vil se ut på desktop og mobil.

Forside
På forsiden har vi liste over klienter. Denne oppdateres ved innlasting av nettsiden

Data visualisering
Data visualiseringstaben består av to grafer.
1. Linegraf som viser nivå i Dispenser målt i prosent. X-aksen går 24 timer tilbake i tid.
2. Stolpediagram som viser antall trykk på dispenser hver dag, 8 dager tilbake i tid.

Grafene viser kun data fra valgt klient på forsiden. Man må derfor huske velge klient først for at data skal vises. X-aksen på grafene vil oppdateres på intervall. Linegraf vil oppdateres hvert 15 minutt. X-aksen på stolpediagrammet vil oppdateres hver dag klokken 00:00.



Figur 16: Datavisualisering på nettside

Logg
Når nettsiden spør etter data vil data fra de siste 4 dager bli sendt, men et maksimum av x-antall indekser. Max indekser kan stilles inn i config.js.
Loggen vil bli 'scrollable' hvis loggen blir høyere enn nettsidens høyde.

Figur 17: Nettside, Logg

Livedata
Logg, stolpediagram og linegraf vil alle oppdateres hver gang valgt klient på forsiden sender ny datapakke. Loggen vil pushe inn en ny øverst på lista og forkaste den siste, hvis loggen overskrider antall indekser.

Linegraf oppdateres siste y-verdien til siste punkt på kurven, uten å endre x-aksen (x-aksen endres på intervall).
Stolpediagram oppdateres ved å øke verdien på siste søyle med 1.
Alt annet av prosessering av data vil gjøres på forhånd på serversiden.

7. Server

Valg av programmeringsspråk
Vi valgte å bruke JS (Javascript) fremfor andre språk mest pga. at nettsiden blir gjort i JS. At både server og nettside er programmert i JS gjør at man enklere kan bruke mye av de samme bibliotekene på begge sider. I tillegg blir det lettere å bearbeide data på server og sende som JSON til nettside.

De aller fleste nettsider er kodet i JS. Dette gjør det lettere å finne biblioteker og ressurser. I tillegg er ofte filer og rammeverk som JQUERY og Bootstrap allerede lagret i cache på nettleser til bruker. Dette skyldes at svært mange nettsider bruker disse ressursene.

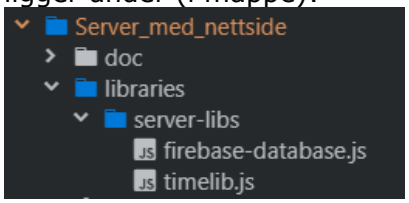
JS er et dynamisk språk. Det vil si at koden ikke kompileres og sjekkes for error.
Du kan faktisk åpne konsollen på Google Chrome og definere, og forandre variabler mens koden kjører. Dette gjør språket lettere å kode.

Moduler/Libraries
Bilde viser brukte moduler. Alle modulene er mye brukt og dokumentasjon finnes enkelt på internett.

```
const express = require('express');
const app     = express();
const http    = require('http').createServer(app);
const io      = require('socket.io')(http);
```

Figur 18: Brukte moduler

Splitting av filer
Filene tilhørende server er splittet for å gjøre prosjektet lettere å lese. Skreddersydde filer ligger under (i mappe):



Figur 19: Libraries, Path

Firebase-database.js inneholder alt som har med firebase å gjøre inkludert prosessering av data fra firebase.

I Timelib.js finner man alle funksjoner og variabler som har med tid å gjøre. Blant annet skal data som kommer fra ESP pakkes inn med tidsstempel. Dette blir gjort backend på server.

```
// Data inn:  "nivå"
// Data ut:   "nivå#year#month#date#klokkeslett"
```

Figur 20: Eksempel på datapakke oppbygning

Server.js
Dette er hovedfilen til serveren. Her importeres alle relevante moduler og egendefinerte biblioteker. Alt av websockets-kode tilhørende server ligger her. Inkludert en rekke funksjoner og variabler brukt til å holde kontroll på rom og klienter koblet til serveren. Ved kjøring av server er det denne filen node kjører.

Port forwarding
For å kunne jobbe på serveren uten å være direkte tilkoblet Raspberry-en eller være tilkoblet dens lokale nettverk, konfigurerte vi «port forwarding» fra den lokale ruteren til Raspberry-en. Dette tillater alle som har tilstrekkelig informasjon om nettverket og serveren å koble seg til serveren fra hvor som helst utenfor det lokale nettverket, via enn SSH-klient – eksempelvis WebStorm eller Putty.

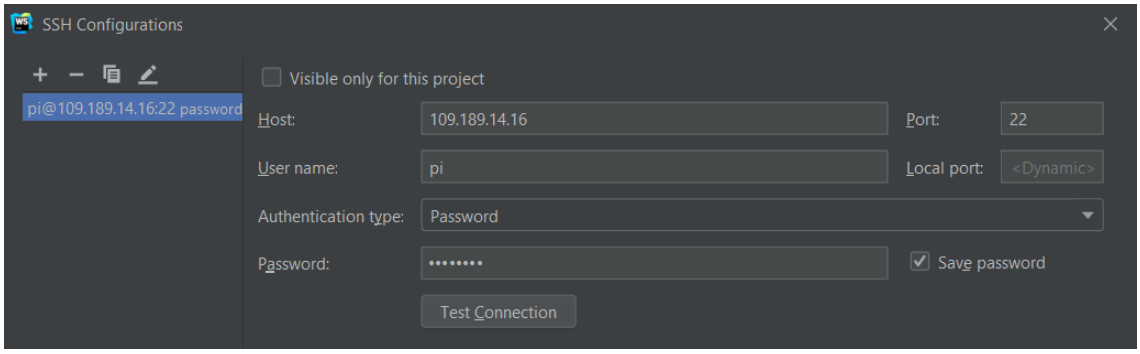
Nødvendigheten for port forwarding kommer av at lokale rutere ofte benytter seg av NAT (Network Address Translation). Dette konseptet dreier seg om at alle enheter på et lokalt nettverk benytter seg av én og samme globale IP-adresse – nettopp ruterens IP-adresse. I tillegg har hver enhet en lokal IP-adresse som kun er tilgjengelig på det lokale nettverket.

Ruteren må likevel klare å skille mellom de lokale enhetenes inngående og utgående trafikk til internettet. Derfor er hver enhet tilkoblet en unik virtuell port på ruteren.

Port forwardingen konfigureres på den lokale ruteren. Dette krever da naturlig nok administratortilgang på ruteren. Konfigurasjonen består i hovedsak av at man forteller ruteren å omdirigere, eller «forwarde», datagrammer/kommunikasjon videre til vår servers lokale IP-adresse ved forespørsel.

Enheten som ønsker å koble seg til serveren fra utsiden av dens lokale nettverk ber da om å koble seg til den bestemte porten. Til dette trenger den eksterne enheten både ruterens globale IP-adresse og portnummeret. Takket være oppsettet av port forwardingen, vet ruteren at den skal omdirigere kommunikasjonen videre til vår servers lokale IP-adresse når den får denne forespørselen.

I tillegg trenger brukeren brukernavn og passord til serveren.



Figur 21: SSH Configurations

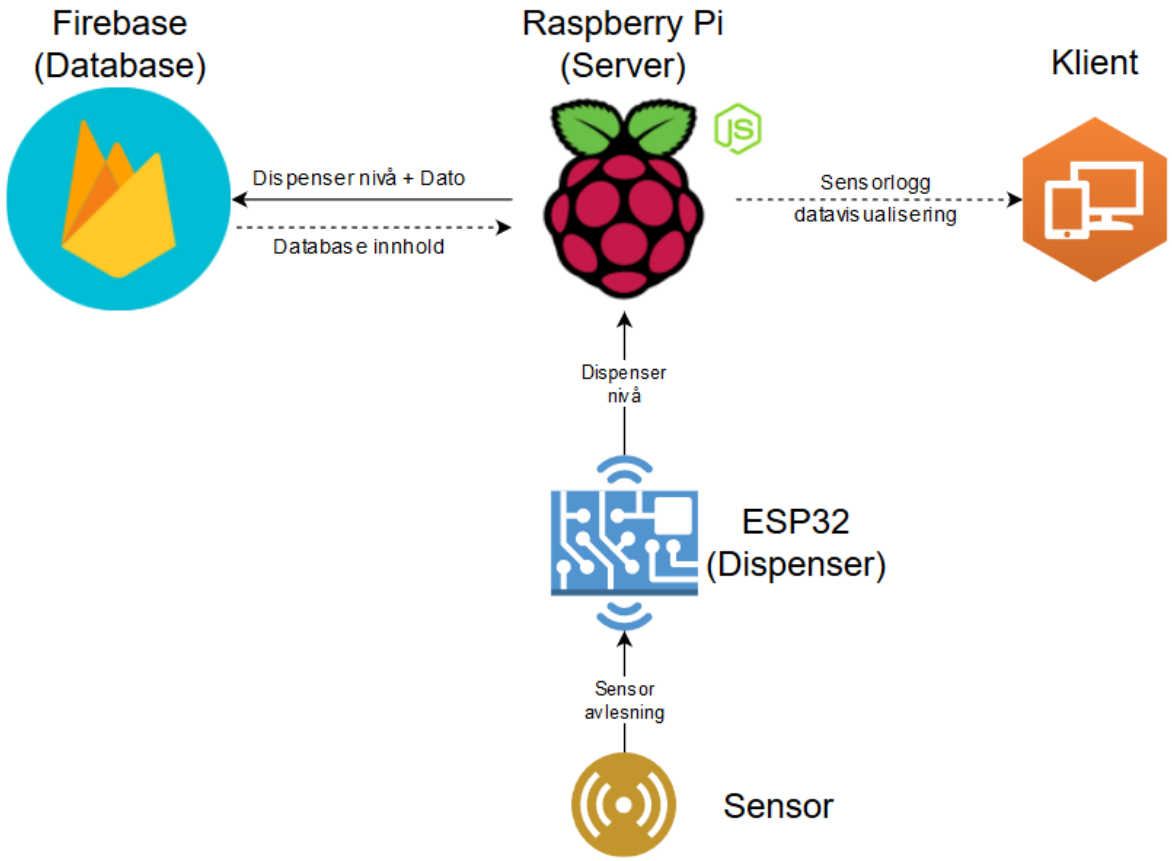
Ettersom konfigurasjonen av port forwarding innebærer at man ber ruteren om å omdirigere kommunikasjonen videre til en bestem IP-adresse, er det viktig at serveren settes til å ha en statisk IP-adresse, slik at denne ikke endres om serveren skulle få en omstart.

Kommunikasjonen mellom den eksterne brukeren og serveren benytter seg av SSH-protokollen (Secure Shell). Denne protokollen oppretter en sikker tunnel for kryptert kommunikasjon, og er godt egnet for å få tilgang til kommandolinjen til en annen datamaskin. For SSH-kommunikasjon brukes som regel port 22.

8. Database

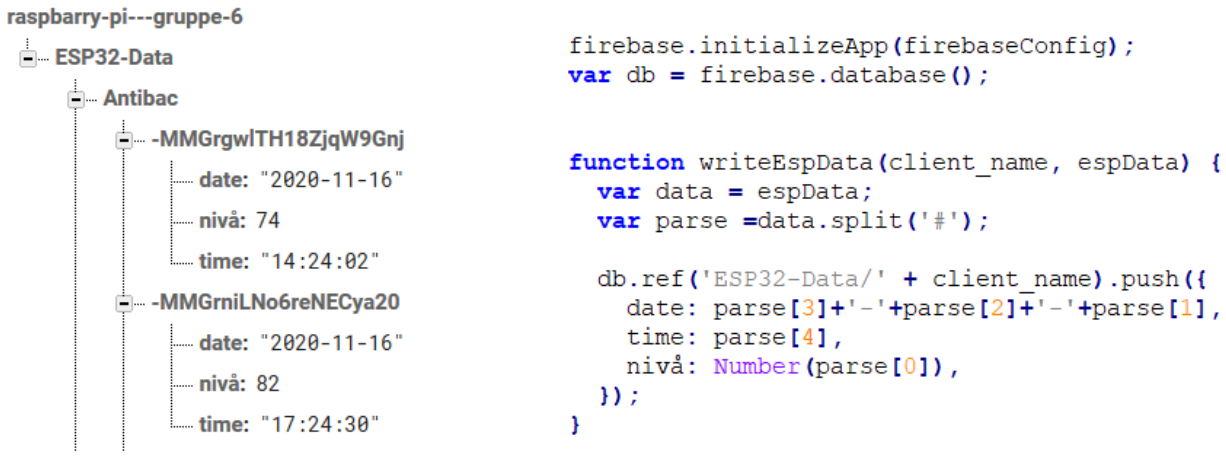
Valg av databasesystem
Vi anså det som nødvendig å kunne lagre sensordataen fra dispenseren. Det er mange kurante database-løsninger å velge mellom for å lagre data. Gruppen vurderte flere alternativer for å lagre data, slik som ren tekstfil, CSV, MYSQL og Firebase. Alle de nevnte løsningene har sine fordeler og ulemper.

Firebase ble valgt foran de andre alternativene da den har en rekke gunstige egenskaper for et slikt web-prosjekt. Firebase er mer brukervennlig enn tradisjonelle relasjonsdatabaser, og krever da mindre forkunnskaper. Tjenesten er skybasert, slik at gruppen ikke må sette opp en lokal database. Den har også fordelen over lokal fillagring med at den er nokså enkel å lese og skrive fra, og kan håndtere flere klienter samtidig. Tjenesten er også gratis for mindre prosjekter slik som denne prototypen. En ulempe med systemet sammenlignet med relasjonsdatabaser er at det er noe vanskeligere å søke i den lagrede dataen.



Figur 22: Database oversikt (tegnet i drawIO)

Firebase
Firebase var nokså enkel å implementere i løsningen vår, da den støtter node.js. Vi trengte bare å legge inn en ferdigpakket kodesnutt som innholdet API nøkkelen og annen prosjektinformasjon, samt å importere og iverksette firebase klassen. Da denne løsningen er skybasert var det ikke nødvendig med et lokalt oppsett på serveren.



Figur 23: Firebase struktur og kode for lagring av data

Da firebase løsningen er veldig anvendelig var det vanskelig å komme fram til en datastruktur. Firebase bruker en type trestruktur i motsetning til tradisjonelle databaser som bruker tabellstruktur. Vi valgte å bruke en flat struktur med navnet til dispenseren som rot for dataen. Med et slikt system kan vi enkelt skalere antallet dispensere.

For hver gang håndsensoren aktiveres vil dispenseren sende det gjenværende nivået i dispenseren til serveren. Serveren vil pakke denne dataen med gjeldene tid og dato for avlesningen, for så å sende dette til firebase. Dataene blir lagt inn i databasen med kommandoen «Push» over «Set», slik at hver avlesning får en unik ID. Vi valgte å la serveren være mellomledd for databasen og dispenseren for å unngå komplisert kode på mikrokontrolleren. En ulempe med løsningen er at mikrokontrolleren ikke lagrer dataen lokalt i tilfelle brudd i datakommunikasjonen.

Sikkerhet
Da denne løsningen er en prototype ble det ikke lagt vekt på datasikkerheten. Det kan være mulig for en klient å lese data den ikke skal ha tilgang til igjennom nettsiden. Det er derfor ikke lagret noen sensitive data i databasen, slik som brukerinfo. Det er ikke trygt å lagre slik brukerdata i samtidsdatabasen til Firebase. Dette er et punkt som burde bli tatt i betraktning ved eventuell videreutvikling av systemet.

9. Raspberry Pi programvarer

Fail2ban
Formålet med dette programmet er å hindre andre i å prøve å ta seg inn på serveren, ved å blokkere dem ute etter et spesifikt antall forsøk med å logge seg inn. Dette skjer ved at brannmuren blir informert, og kan derfor stoppe påloggingsforsøk fra mistenkelige IP-adresser. Vi blir da spart for å sjekke etter loggfiler for inntrengingsforsøk og deretter oppdatere brannmuren manuelt, for å forhindre dem i å ta seg inn til serveren. (Raspberry Pi, 2020)

IPtables - brannmur
Bestemmer hvilke porter det tillates nettverkstrafikk gjennom og hvilken det ikke tillates gjennom. I tillegg er det offisielt denne som avviser koblingsforsøk hvis en IP-adresse er bannlyst.

Vi har også lagt til noen portregler og det er disse portene som tillates:
- «sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT» **(SSH normal port)**
- «sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT» **(nettside)**
- «sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT» **(nettside kryptert)**
- «sudo iptables -A INPUT -p tcp --dport 2520 -j ACCEPT» **(Node tjener)**

Disse portene tillates ikke:
- «sudo iptables -P INPUT DROP» **(blokker alle inngående porter utenom de over)**
- «sudo iptables -P FORWARD DROP» **(blokker alle videresendende porter)**
- «sudo iptables -P OUTPUT ACCEPT» **(aksepter alle utgående porter)**
- «sudo iptables -A INPUT -m conntrack -m cpu --cpu 0 -j ACCEPT --ctstate RELATED,ESTABLISHED» **(gjør at oppkobling går raskere, samme med neste)**
- «sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT»

Siden brannmuren ikke oppdateres automatisk med bannlyste IP-adresser har vi skrevet inn en kommando:
- «sudo apt-get iptables-persistent»

Denne gjør da at vi slipper å skrive kommandoene på nytt hver gang vi tar omstart på raspberry pi-en, fordi pakken vil automatisk kjøre selv.

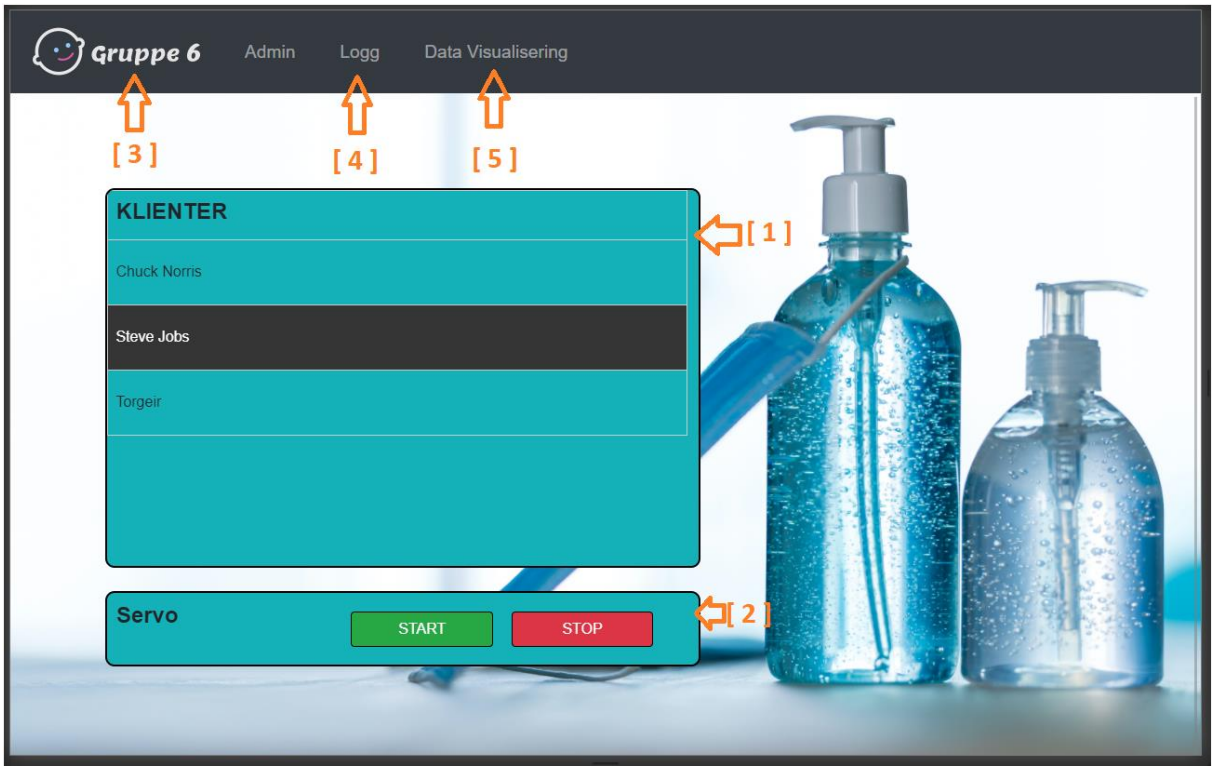
Senere la vi blant annet merke til at flere av kommandoene var skrevet opp flere ganger i filen, fordi vi trodde det hadde skjedd feil underveis, og hadde derfor skrevet inn kommandoene flere ganger. Dette fikk vi ordnet opp i ved å endre filen.

Node.js
Med Node.js har vi muligheten til å skrive/kjøre JavaScript utenfor en nettleser på tjener. Med denne muligheten kan man skape kommunikasjon med klienter.

NPM (Node Packet Manager)
NPM er en pakkeleder for JavaScript. Med denne vil vi få flere funksjoner som vil utvide funksjonalitetet til Node.js. Vi får også et kommandoverktøy som kan kjøres via terminal og vi kan da laste ned flere biblioteker til fordel for Node.js. Blant annet pakker som express, http og socket.io lastet ned gjennom NPM. Full liste over biblioteker lastet ned gjennom NPM finner man i fila «package.json».

6. BRUKERMANUAL

1. Nettside



Figur 24: Nettsidens forside vist i nettbrett-størrelse

[1]: Liste over dispensere koblet til server. Trykk på en klient i lista for å velge hvilken klient man ønsker å styre, og lese data fra. Det er viktig å merke seg at en klient må velges før man kan lese data fra [4] og [5].

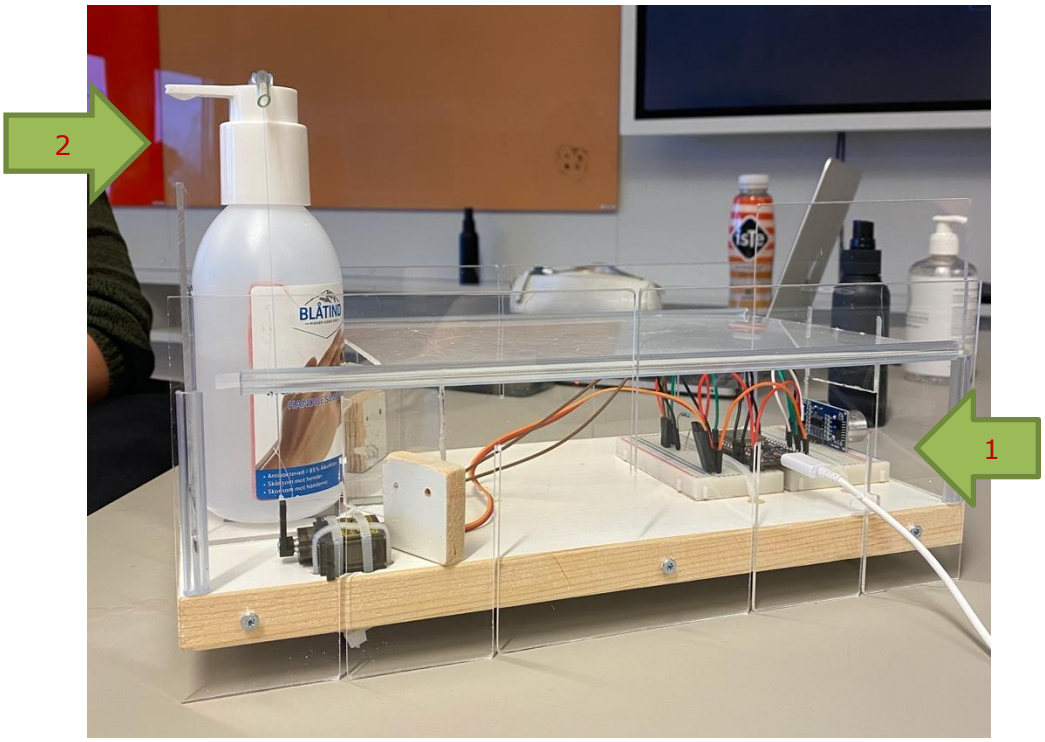
[2]: Denne listen popper fram etter man har valgt en klient. Her kan man velge å stoppe eller starte dispenseren til valgt klient. I stop vil ikke servo kjøre selv om bruker holder hånda foran dispenseren.

[3]: Trykk for å gå til forsiden.

[4]: Trykk for å se Logg. Her finner man en liste over nylige målinger.

[5]: Trykk for å se stolpediagram og Linegraf. Linegraf viser valgt klients nivå siste 24 timer.
stolpediagram viser antall trykk på flasken hver dag siste 8 dager. Man kan ut fra denne analysere trafikk. Begge grafer vil oppdatere x-aksen automatisk på intervall og y-aksen for hver ny måling.

2. Dispenser



Figur 25: Prosjektprototypen

[1]: Hold hånden under dispenseren for å ta imot antibac.

[2]: Før hånden foran ultralydsensor.

7. VURDERING AV BRUKTE LØSNINGER

ESP
Vi har valgt å bruke ESP32 microcontroller til å styre klienten. En av grunnene til dette er at den støtter Arduino C som er et språk hele gruppa er kjent med. Alternativt kunne vi brukt Arduino UNO som vi også har tilgang til. Den største fordelen med ESP32 er at den har innebygd wifi funksjon. Dette mangler Arduino UNO hvor man må gå via en pc med wifikort, dette er veldig tungvint og vi ønsker en modell som kan stå for seg selv. En annen fordel med ESP32 er at den er både billigere, har flere inn- og utganger, større flash memory, høyere hastighet (clock speed) og mindre i størrelse. For vårt formål er ESP32 en meget god løsning.

Den fysiske noden, ESP32- mikrokontrolleren ble programmert i APL (Arduino Programming Language) da gruppen hadde kjennskap til dette programmeringsspråket, og det skulle lages et C-bibliotek til prototypen. Biblioteket kunne man da skrive direkte i IDE-en til Arduino. Ved å benytte Arduino-IDE var vi også sikre på at koden ble kompilert før kjøring, dette gjør at man unngår uønskede feil ved kjøring. Ikke alle kodekjøringsprogrammer har denne funksjonaliteten.

Micropython som også er et objektorientert programmeringsspråk kunne vært brukt i stedet for APL, det finnes mye og lett tilgjengelig informasjon og eksempler på kombinasjonen ESP32 devkitC og Micropython. Micropython er bygd på C slik som APL, men gir tilgang på flere funksjoner. Det er mulig å bruke Python datatyper slik som dictionary med Micropython, dette er ikke mulig med et enklere språk slik som APL.

Gruppen hadde erfaring med APL i kombinasjon med ESP32 utviklerkortet, og kjent med at APL bygger på C og C++, statiske programmeringsspråk. Valget man tok ved å benytte APL og ikke eksempelvis Micropython kan diskuteres, det viser seg i etterkant at det var enklere å finne dokumentasjon på Micropython kombinert med ESP32, enn APL og ESP32. Micropython støtter i tillegg feilhåndtering, noe som kan være vanskelig med APL.

Gruppen opplevde noen ulemper ved bruk av klasser og metoder skrevet av andre utviklere. Dette problemet forsterkes med hvordan Arduino-utviklingsverktøyet behandler biblioteker. Når flere på gruppen jobber med samme programvare kan det være vanskelig å holde rede på hva som kommer fra våre, og hva som kommer fra eksterne bibliotek.

Samtidig finnes det tilgjengelige bibliotek for socket.io som man enkelt kan implementere i koden og den blir dermed kort og oversiktlig, selv med bruk av objektorienterte programmeringsspråk.

Flytkontroll
Gruppen har brukt verktøyet Git igjennom Github portalen for å kontrollere program-versjonen vår. Dette har gitt muligheten til å reversere ugunstige endringer i kodebasen til nettsiden og microkontrolleren. Dette verktøyet var også nyttig for å kunne programmere løsningen samtidig, og slik kunne dele opp arbeidet blant gruppen.

Websockets
I prosjektet ble det benyttet TCP-protokoll i kommunikasjonen mellom enhetene; ESP32-klient til/fra server, og nettsideklient til/fra server. Dette ble realisert ved bruk av socket.io, en websocketbasert protokoll. Denne gjør at man har en pålitelig dataoverføring, som sikrer at det ikke forekommer tap av data underveis i kommunikasjonen mellom enhetene ved godkjenning av mottatte datapakker for hver enkelt overført pakke, både til og fra server. Socket.io sin fulldupleks- overføring gir rask overføringshastighet.

Websocket-protokoll kan man kode fra bunnen av, men i socket.io- bibliotek ligger alle grunnfunksjoner innebygd og er meget arbeidsbesparende når disse kan importeres til kode. Man må uansett sette seg inn i funksjonaliteten med websockets for å kunne bruke socket.io sine bibliotek.

Socket.io er kompatibel med plattformen Node.js. Denne plattformen gjorde at vi kunne programmere klientene med hendelsesbasert kodestruktur. Hendelsene behandles asynkront gjennom Node.js av server, og sikrer dermed minimal forsinkelse i respons mellom server og klienter.

Server
Serveren ble kodet i JS og vi har brukt Node.js til å kjøre serveren. Bibliotekene: 'express', 'http' og 'socket.io' er installert gjennom NPM. Disse sørger for kommunikasjon mellom server og klient. Dette inkluderer håndtering av http get-requests og etablering av websockettilkobling.

Ved valg av programmeringsspråk til nettsiden er det naturlig å bruke JS. Dette da de aller fleste nettsider er skrevet i JS. På serveren kunne vi like gjerne brukt et annet språk, for eksempel Python. Ved å bruke JS både på nettsiden og backend på server, vil begge kommunisere på samme språk. Dette gjør det enklere å behandle filer backend, før man sender data til nettsideklient. Objekter kan da enkelt sendes gjennom JSON uten å måtte skrive om formatet først. I tillegg brukes samme moduler / biblioteker, som brukes på server, bli lastet opp til nettsideklienten.

JS er et dynamisk språk som ikke kompileres. Blant annet trenger man ikke spesifisere datatyper. Dette gjør at feilmeldinger sjeldent blir oppdaget av IDE. Dette øker sannsynligheten for krasj av server eller bugs under drift. Dette kan være en utfordring. I et forsøk på å forbedre situasjonen har vi brukt 'use strict' på toppen av mange av filene. Dette vil gjøre at IDE-en reagerer på enkelte syntax-error etc. I tillegg har vi brukt 'try & catch' for å unngå krasj.

Til ettertanke kunne vi programmert i TS (Type Script) i stedet for JS. TS er bygd på JS, men legger til strengere krav til syntax. En Typescript kompilator vil kompilere koden til vanlig JS. Forskjellen ved bruk av TS er altså at flere error vil bli tatt i kompilator. Programmering i rå JS krever mer testing i ettertid og er mer utsatt for feil.

Filtrering av datatrafikk
Vi har prøvd å minske belastningen på nettet. Dette ved å begrense størrelsen på enkelte pakker, og ved å filtrere trafikk til størst mulig grad. (Se kommunikasjonskoder).
For å filtrere trafikk har vi brukt to konsepter: rom, og klient-id (og navn).

Ved å bruke rom har vi splittet type klienter i to grupper: 'website' og 'esp'.
Rom ble brukt flere steder. Et eksempel er når en klient i rom 'esp' kobler til eller fra skal klientlisten hos alle nettsideklienter oppdateres. En ny datapakke vil da bli sendt til alle klienter i rom 'website'.

Alle ESP-klienter vil bli tildelt et navn, for eksempel Chuck Norris. Når en ESP-klient kobler til server vil navnet bli lagret i et objekt, sammen med en klient-id. Dette er den samme id-en websocket bruker. Dette brukes til å sende, eller forespørre data fra/til en spesifikk klient.

Filtrering av datatrafikk så ut til å fungere ved testing. Men det ble oppdaget en ting som er åpen for forbedring. Når en ny datapakke sendes fra en ESP-klient til server, vil server pakke inn datapakken med et tidsstempel og sende den videre til alle nettsideklienter. Dette er ikke et problem når det er få nettsider koblet til. Men hvis man ønsker å skalere opp systemet vil dette føre til unødvendig datatrafikk. Her ville det altså vær ønskelig å sende bare til de nettsideklienter som ønsker data fra gjeldende klient. Dette kan fikses ved å endre objektet 'clients', slik at hver nettsideklient lagres med både id og valgt klient navn. Flere ting må endres gjennom systemet og vi har derfor konkludert med at det ikke var verdt å gjøre da dette er en prototype.

```
var clients = {
  esp: [{
    clientId: "129009j0tjgno2"
    clientName: "Chuck Norris"
  }],
  website: [{
    clientId: "1209uj0hj0hg09"
    selectedClient: "Chuck Norris"
  }]
}
```

Figur 26: eksempel på fix

Målinger sendt fra ESP-klienter lagres på Firebase med et tidsstempel (klokkeslett og data). Dette tidsstempelet blir lagt til backend på server. Dette skjer av to grunner:
1. Vi ønsker at mesteparten av kalkulasjoner og dataprosessering foregår backend.
2. Vi unngår problemer der hvor klokkene til forskjellige klienter er ute av sync.

Serveren ble satt opp ved bruk av Raspberry Pi, selv om vi kunne brukt en VPS. Underveis i prosjektet ble det benyttet lokal webserver ved testing av kode og overføring til database frem til serveren var ferdig satt opp på RSPI'en. Krav til serveren var at denne skulle være tilgjengelig for kommunikasjon med klientene via WiFi, som RSPI har støtte for. Her kunne man brukt andre mikrokontrollere som støtter WiFi, eventuelt brukt utviklingskort som er kompatibelt med WiFi-shield.

RSPI som server var likevel det man ville prøve på. Det var en ukjent komponent for alle og interessant å finne ut mer om. I operativsystemet til RSPI er det i innebygd SSH, som gjorde den til et godt valg med tanke på sikkerhet i oppkoblingen mellom server og klient. RSPI egner seg svært godt til slike småprosjekter, som dette. Ved større prosjekt med svært mange klienter og datatrafikk vil det være aktuelt med noe kraftigere.

Sensorer
Utgangspunktet for valget av komponenter generelt var å bruke komponenter vi allerede hadde tilgang til, for å både se om man kunne klare seg med ting man hadde, og unngå stress ved å komme raskt i gang med alle elementer i prosjektet.
Til sammen i prosjektgruppa hadde vi da gode sensorer for å starte utviklingen av prosjektet allerede fra dag én, med den lysende idéen om å prototype en kontaktløs antibacdispenser.

Trykksensoren vi benyttet var meget følsom for om objektet den målte utøvde press innenfor et bestemt avgrenset område. Antibacflasken var større i diameter enn selve

sensoren så underlaget under flaska og over sensoren måtte tilpasses størrelsen på sensoren.

Til nivåmåling kunne man derfor brukt en annen sensor. Det kunne vært annen type trykksensor, eller eksempelvis en tranciever-variant oppstilt på siden av flasken om den kunne vært programmert til å måle blank væske i en blank flaske.

Selve programmeringen for trykksensoren var en enkel spenningsdeler. Dermed er den en godt egnet sensor og vil i tillegg være et nyttig element om prototypen skal brukes videre i opplæringsøyemed.

Ultralydsensoren fungerte utmerket til oppgaven med å registrere objekt slik vi ønsket. Den passet godt for de avstandene vi hadde behov for å måle, og det fantes bibliotek man kunne importere som gjorde det overkommelig å implementere det vi ønsket i koden på en grei måte. Det finnes avstandsmålere som er mindre i fysisk størrelse, det kan man ta med i vurderingen om størrelsen på prototypen ønskes nedskalert.

Servomotoren vi brukte var liten. Hadde man hatt tilgang på en sterkere motor kunne man brukt det, da vi ikke har maks omdreiningskraft når vi kun bruker 5V. Eventuelt kunne man brukt en ekstern spenningskilde på motoren for å få opp ytelsen om det var behov for det eller bruke to servomotorer, en på hver side av flasken. Dette merket vi også da vi byttet flaske, motoren klarte å pumpe ned på den første flasken, men ikke på den andre. For kombinasjonen servomotor og ESP32 finnes det et funksjonelt bibliotek som vi benyttet oss av ved programmering av motoren.

8. KONKLUSJON/ERFARING

Prosjektet har gitt oss et godt innblikk i hvordan elektroniske enheter kommuniserer, og hvordan internett er bygget opp. Vi har erfart at internettkommunikasjon krever samspill mellom flere komplekse systemer. Et slikt prosjekt drar nytte av allerede etablerte protokoller som tar seg av kommunikasjonen i de underliggende nettverkslag.

Ved å se på et helt systemoppsett har vi fått innblikk i mye mer av helheten og mange av elementene i datakommunikasjon, enn om vi kun skulle tatt for oss én protokoll. Vi har i dette prosjektet jobbet med både frontend og backend av et relativt enkelt system. Vi har på denne måten fått en større forståelse av helheten hvordan slike systemer er bygd opp.

Vi har igjennom prosjektet opparbeidet forståelse for hvordan man kan implementere websocket for kommunikasjon imellom forskjellige enheter. En kommunikasjonsanalyse igjennom Wireshark har gitt oss en forståelse om hvordan denne kommunikasjonen foregår i praksis.

Vi er svært fornøyde med å ha bygd opp majoriteten av prosjektet uten bruk av eksempelkode. Plattformen fungerer godt både på mobil, nettbrett og desktop. I tillegg har vi et dynamisk mobilt system som ikke nødvendigvis trenger en Raspberry for oppsett. Den serverbaserte prototypen er skalerbar, slik at den skal kunne håndtere flere dispensere og klienter.

9. REFERANSER

Components101 (2017) *HC-SR04 Ultrasonic Sensor*
Tilgjengelig fra: https://components101.com/ultrasonic-sensor-working-pinout-datasheet (Hentet: 15.november 2020)

Datasheetspdf (2014) *Gear Servo. MG90S Datasheet*
Tilgjengelig fra: https://datasheetspdf.com/pdf/1106582/ETC/MG90S/1 (Hentet: 15.november 2020)

Espressif Systems (2020a) *ESP32-WROOM-32D & ESP32-WROOM-32U Datasheet*
Tilgjengelig fra: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf (Hentet: 17.november 2020)

Espressif Systems (2020b) *ESP32-DevKitC V4 Getting Started Guide*
Tilgjengelig fra: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html (Hentet: 17.november 2020)

Espressif Systems (2020c) *Get Started*
Tilgjengelig fra: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html#introduction (Hentet: 17.november 2020)

Espressif Systems (2020b) *ESP32 Technical Reference Manual*
Tilgjengelig fra:
https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf (Hentet: 17.november 2020)

Interlink Electronics (2020) *hånddesinfeksjonFSR ® 400 Series Data Sheet*
Tilgjengelig fra:
https://cdn2.hubspot.net/hubfs/3899023/Interlinkelectronics%20November2017/Docs/Datasheet_FSR.pdf (Hentet: 15.november 2020)

Kjell & Company (2017a) *Hva er Arduino og Genuino?*
Tilgjengelig fra: https://www.kjell.com/no/kunnskap/hvordan-virker-det/arduino/introduksjon/hva-er-arduino-og-genuino (Hentet: 17.november 2020)

Kjell & Company (2017b) *Variabler og strenger*
Tilgjengelig fra: https://www.kjell.com/no/kunnskap/hvordan-virker-det/arduino/grunnleggende-programmering/variabler-og-strenger (Hentet: 17.november 2020)

Randomnerdtutorials (2016) *Installing the ESP32 Board in Arduino IDE (Windows, Mac OS X, Linux)*
Tilgjengelig fra: https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/ (Hentet: 17.november 2020b)

Raspberry Pi (2020) *Securing your Raspberry Pi.*
Tilgjengelig fra: https://www.raspberrypi.org/documentation/configuration/security.md
(Hentet: 23.oktober 20)

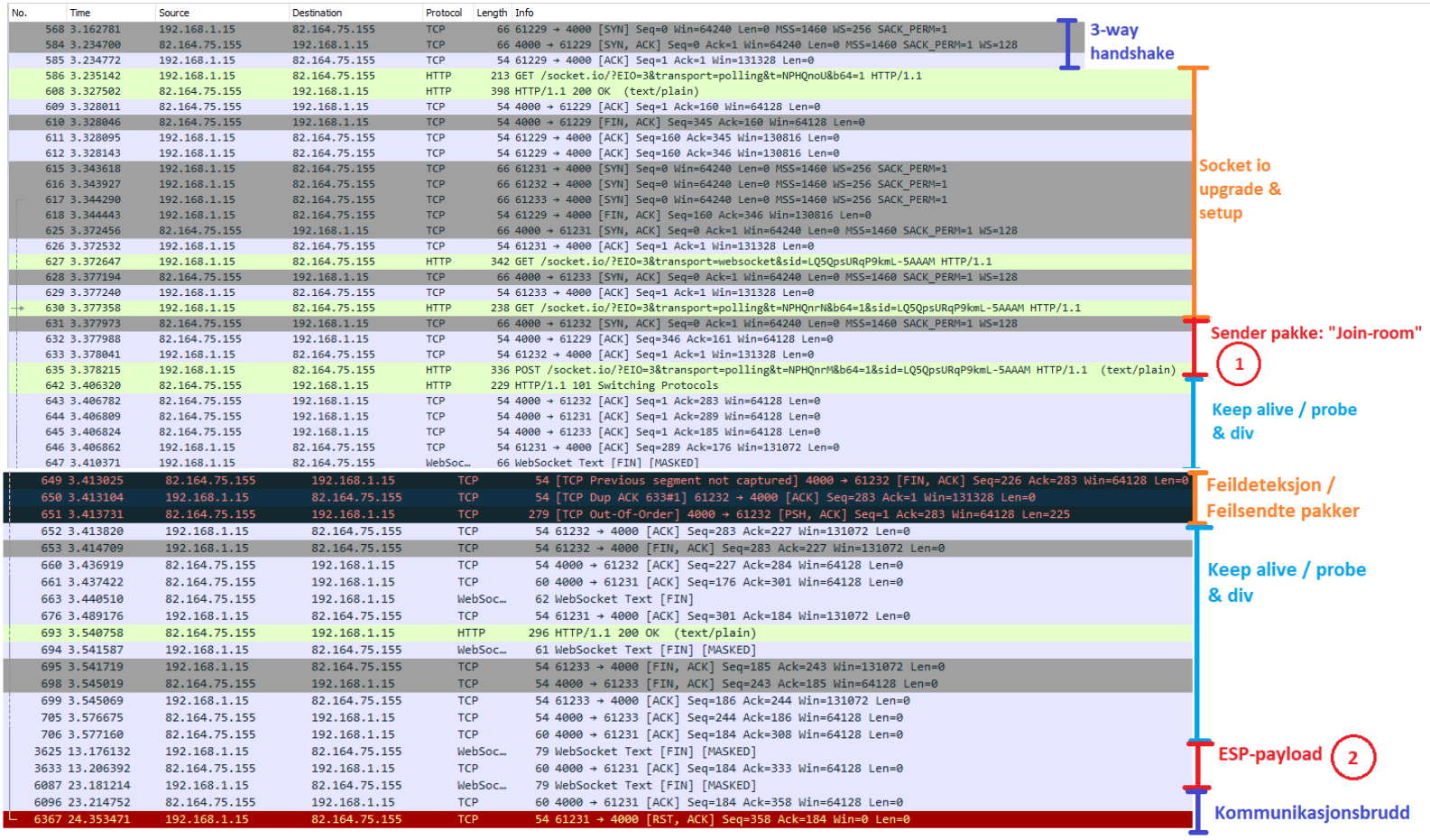
Tutorialspoint (2020) *Arduino - Quick Guide*
Tilgjengelig fra: https://www.tutorialspoint.com/arduino/arduino_quick_guide.htm (Hentet: 17.november 2020)

10. VEDLEGG

Vedlegg 1: Utstyrsliste

Type utstyr	Spesifikasjoner	Antall
Mikrokontroller	ESP32 DEVKIT	1
Servomotor	SG90 Micro-servo motor	1
Ultralydsensor	HC-SR04 Ultrasonic Sensor	1
FSR	FSR 406 Force sensitive resistor	1
Resistor	10k Ohm resistor	1
Breadboard		2
Raspberry pi	3B	1

Vedlegg 2: Wireshark-analyse



Figur 27: Wireshark analyse

Gruppen har brukt protokollen websocket for datakommunikasjon i de fleste ledd av prosjektet. Gruppen har derfor brukt wireshark for å analysere pakke-strømmen mellom ESP klient og server, for å oppnå forståelse for hvordan websocket håndterer datakommunikasjonen. Dette ble utført både på klient siden og på server siden, men siden kommunikasjonen er den samme har vi valgt å bare vise klient siden.

Vi observerte blant annet at mye data ble sent under oppsett av kommunikasjonen etter «3-way handshake». Denne kommunikasjonen inneholdte data for oppsett av websocket, slik som «keep-alive» og ping intervall. Det ble også observert hvordan websocket håndterer «keep-alive» med såkalte probe HTTP-pakker.

```
96:0{"sid":"LQ5QpsURqP9kmL-5AAAM","upgrades":["websocket"],"pingInterval":
25000,"pingTimeout":5000}2:40
```

Figur 28: Websocket Setup

Når kommunikasjonen var etablert kunne vi se de sendte datapakkene fra klienten. Da disse pakkene inneholder relativt lite data ble de sendt igjennom bare en TCP-pakke. Da datastrømmen ikke er kryptert kunne vi lese av innholdet i wireshark, som vist på figuren over. Denne overføringen sender tallet 80 fra klienten, som er nivået på dispenseren i prosent. Denne overføringen blir sendt hver gang sensoren registrerer en bevegelse.

Websocketprotokollen er en mye brukt protokoll som støttes av de fleste nettlesere. Protokollen er «statefull» og krever ingen «polling» som vil si at server kan selv pushe pakker til klient uten at klient har spurt om data. Dette var akkurat det vi så etter. Vi valgte derfor å bruke websockets.

Fordeler og ulemper med Websockets (generelt)
Websockets er bygd på http-protokollen. Kanskje den største fordelen med websockets er at kommunikasjonen holdes åpen. For at dette skal skje må sendes en del pakker frem og tilbake. Disse går på intervaller. Hos oss er intervallet 25 sekunder. Det vil si at pakker sendes mellom klient og server, som fører til mer trafikk.

En annen fordel med websockets er at klienten ikke behøver å spørre server om informasjon. Serveren kan selv «pushe» pakker direkte til klient.

Da websockets er bygd på http og TCP er forbindelsen «statefull». TCP-protokollen holder orden på pakkene i form av sekvensnummer, samt sender [SYN] og [ACK] pakker for å verifisere at pakkene er kommer frem. Dette fører igjen til mer datatrafikk.

Det at websockets er bygd på http og TCP gjør den brannmurvennlig. Dette da brannmuren kjenner igjen trafikken som sendes og vil som oftest ikke prøve å sperre den.

Websockets bruker noe mindre overhead enn http. Dette er med på å minske størrelsen på noe datatrafikk.

Kort oppsummert sørges websockets for pålitelig overføring av data vha. TCP. I tillegg vil tilkoblingen holdes åpen. Ulempen med dette er at websockets vil sende pakker frem og tilbake for å holde tilkoblingen åpen selv om klienten ikke har noen pakker å sende.