Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for elektroniske systemer

Pragaash Mohan
Sondre Ravnås Vik
Jesper Rombo Mejlænder Oddaker
Andreas Glomsrud

Serverbasert køsystem med adgangskontroll

Bacheloroppgave i Elektronikk
Veileder: Olav Alexander Myrvang
Mai 2021

NTNU
Kunnskap for en bedre verden

Pragaash Mohan
Sondre Ravnås Vik
Jesper Rombo Mejlænder Oddaker
Andreas Glomsrud

Serverbasert køsystem med adgangskontroll

Bacheloroppgave i Elektronikk
Veileder: Olav Alexander Myrvang
Mai 2021

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for elektroniske systemer

NTNU
Kunnskap for en bedre verden

NTNU

Institutt for elektroniske systemer
Institutt for elkraft
Institutt for teknisk kybernetikk

Bacheloroppgave

Oppgavens tittel: Serverbasert køsystem med adgangskontroll	**Gitt dato:** 14.01.2021 **Innlevingsdato:** 20.mai 2021 **Gradering** **[X] åpent** **[ ] lukket** **[ ] åpent fra** _____
Project title: Server-based queueing system with access control	**Antall sider/bilag** 90
Gruppedeltakere: Jesper Rombo Mejlænder Oddaker Sonde Ravnås Vik Andreas Glomsrud Pragaash Mohan	**Veileder internt:** Olav Alexander Myrvang 40338101 olav.myrvang@ntnu.no
Studieretning: Elektronikk	**Prosjektnummer:** 57
Oppdragsgiver: NTNU IES	**Kontaktperson hos oppdragsgiver:** Ingulf Helland ingulf@ntnu.no 73594214 / 92615557

Sammendrag

Institutt for elektroniske systemer (IES) ønsker å bygge et kø- og adgangssystem på elektronikklaboratoriet. Denne skal være oppkoblet til kanalsystemet over labplassen, og skal tilby studentene en enkel måte for innlogging, samt registrering i kø.

Prototypen som ble utviklet implementerer en Raspberry Pi 4B, touchskjerm, navigasjonsknapper og RFID-leser. Rammeverket som blir benyttet er en kombinasjon av programmeringsspråk som Python, PHP og JavaScript. Dette har resultert i et sammensatt system som tilbyr en web-basert løsning for innlogging, registrering og administrering av køen. Studentene benytter seg av tildelt adgangskort til pålogging, og kan navigere systemet ved hjelp av både knapper og touch-skjerm. Et kretskortdesign er også utviklet for fremtidig montering i kanal.

Gitt oppgavens avgrensninger så konkluder vi med at prosjektet var vellykket. Gruppen presenterer et levedyktig konsept med godt potensial for videre utvikling.

Stikkord:

RFID, Raspberry Pi, Webutvikling, Touchskjerm, Køsystem

Summary

The Department of Electronic Systems (IES) wishes to build a queue- and admission-based system in their electronics laboratory. This must be connected to the cable channel mounted above the lab seat and will offer students a simple way of logging in and requesting assistance.

The developed prototype implements a Raspberry Pi 4B, touchscreen, navigational buttons, and RFID-reader. The framework used is a combination of different programming languages, such as Python, PHP, and JavaScript. This has resulted in a combined system which offers a web-based solution for log-in, registering, and administration of the queue. Students make use of their assigned access cards to log in and can navigate the system with the use of both buttons and touchscreen. A circuit board design has also been developed for future assembly in the cable channel.

The group has deemed the project a success given the assignment's constraints. We have presented a viable proof-of-concept with good potential for future improvements.

Keywords:

RFID, Raspberry Pi, Web development, Touchscreen, Queuing system

Forord

I dette bachelorprosjektet har 4 elektroingeniørstudenter, med spesialisering i elektronikk, ved NTNU utviklet en prototype for et serverbasert køsystem med adgangskontroll. Prototypen legger et godt grunnlag for å effektivisere læringsprosessen ved elektronikklaboratoriet på NTNU.

Motivasjonen bak oppgaven kommer fra opplevelser og ergrelse ved lang ventetid og lav effektivitet ved utførelse av laboppgaver. Videre ønskes det å utvide systemet for adgangskontroll av utstyret som er på labben. Det er lagt opp for installasjon av dette kontroll- og køsystemet ved tidligere oppgraderinger av lab.

Studentene ønsker å takke følgende personer for støtte og veiledning gjennom prosjektet:

- Oppdragsgiver Ingulf Helland, ved Institutt for Elektroniske Systemer, NTNU Trondheim
- Veileder Olav Aleksander Myrvang, ved Institutt for Elektroniske Systemer, NTNU Trondheim

Kildekoden i sin helhet kan bli lastet ned her:

https://github.com/pragaashm/bachelor57-QueueSystem

Andreas Glomsrud | Jesper Oddaker | Pragaash Mohan | Sondre Ravnås Vik

Innholdsliste

Figurer

Tabeller

Ordliste

ASC	Stigende rekkefølge
Baud	En enhet for modulasjonshastiget gitt digitale signaler
Boolsk	Variabel med kun to tilstander, eksempelvis høy/lav og sant/usant
Breadboard	Koblingsbrett for testing av elektronikk
Chromium-nettleser	Open-source nettleser laget av Google
CMOS	Felles betegnelse for kretser med felteffekttransistorer av P- og N-kanal i kombinasjon
ENUM	Enumerated Type
ER-diagram	Entity Relationship Diagram
Event Listener	Funksjon som venter på en handling
Female header	Et elektrisk støpsel
Full dupleks	Informasjonsstrømming mellom to enheter parallelt i begge retninger
GNU/Linux	En gruppe open-source operativsystem
Halv dupleks	Informasjonsstrømming mellom to enheter i begge retninger, ikke samtidig
Integer	Datatype med heltall som verdi
Kapasitiv	Evnen til å lagre elektrisk ladning
Modul	En mindre del av et fullstendig system
MySQL Injection	Uønsket ondsinnet endring av databaseverdier via kodeinjeksjon
MySQL Query	Instruksjoner for databaseendringer
MySQL Trigger	Lagret programkode som aktiveres når tabeller oppdateres i databasen
Open Source	Programvare hvor koden er åpen for redigering og visning fra alle
Periferienhet	Utstyr som kobles til, og kommuniserer med en datamaskin

PLS	Programmerbar logisk styring
Read Only	Data som er ikke modifiserbar
Resistiv	Evnen til å motstå elektrisk strøm
RJ45	Tilkoblingsterminering for nettverkskabler
Seriell kommunikasjon	Informasjonsoverføring en bit om gangen
Session	Identifikator for ansvar i køsystemet
Simpleks	Enveiskommunikasjon mellom to enheter
Standoff	Skrue for å skape mellomrom mellom monterte PCB kort
String	Datatype som kan inneholde tekst eller binær data
Studass	Studentassistent
Ubuntu	Linux-distribusjon

1 Innledning

1.1 Prosjektets bakgrunn

Institutt for elektroniske systemer (IES) er underlagt NTNU, som underviser og forsker på elektroniske systemer samt videreutvikler dette. I forbindelse med dette har IES også ansvaret for undervisningslaboratorier, som elektrolaboratoriet. Det er primært brukt av 1. og 2.års studenter og har en kapasitet på ca. 260 studenter fordelt på 88 labplasser. Hver labplass representerer en investering på ca. 300 000 NOK i utstyr og infrastruktur, og krever årlig ca. 300-600 timer av teknisk personell for drift, forbedring samt videreutvikling.

I 2010-2012 samt 2017-2018 gjennomgikk laboratoriet oppgradering av infrastruktur, innredning og IT-systemer. I tillegg til dette ble det planlagt fremtidige utvidelser, som forberedt på mulig køsystem. Grunnet laboratoriets utforming samt tidvis lav kapasitet av studentassistenter kan det oppstå forsinkelser og uheldige ventetider for studentene. Dette kan resultere i en skjev fordeling av hvem som får hjelp, og kan forverre erfaringen for både studenter og assistenter samt redusere læringsutbytte. I denne forbindelsen er det ønskelig å anvende et køsystem som skal være parallelt med PLS-systemet på hver labplass. Dette skal sikre rettferdig adgang til studasshjelp samt bedre studentenes opplevelses på elektrolaboratoriet.

1.2 Problemstilling

> Elektrolaben i Høgskoleringen 3 er forberedt for smart-styring av lab-plasser. Denne smartstyringen er forberedt for at hver student / gruppe logger på lab-plassen med adgangskortet sitt. Videre er det og tiltenkt LCD-skjerm med køsystem for studasshjelp. Her kan det være store muligheter for en student/studentgruppe som ønsker å være kreative. Her kan det inngå både design av kretskort, mikrokontrollersystem med grafisk grensesnitt mot LCD, serverprogrammering, RFID-leser design, og bussystemer for sammenkobling. Oppgaven avgrenses i forhold til studentenes interesser og ferdigheter.
>
> *Problemstilling fra Overingeniør Ingulf Helland, ved Institutt for Elektroniske Systemer, NTNU Trondheim*

1.3 Tekniske rammer og avgrensinger

Tekniske rammer

Hver arbeidsplass har en INKA101 el-kanal installert. Denne er oppkoblet over en PC-skjerm og er tiltenkt kø- og styresystemet. Arbeidsplassen er utstyrt med en 40-leder-flatkabel, hvorav 10-15 ledere er reservert PLS-styresystemet. I tillegg er den utstyrt med en CAT-6 nettverkskabel samt en 12-15V DC forsyningskabel. Styresystemet hadde som opprinnelig plan å tilby aktivering av labplassens funksjon, uavhengig av andre plasser. Køsystem og adgangskontroll var planlagt som ekstrafunksjon på labplassnivå. Laboratoriet er utstyrt med to Linux-baserte servere. Kø- og adgangssystem vil være adskilt fra driftssystemet grunnet sikkerhet. Se vedlegg U. for tegning av labplass.

Avgrensinger

Grunnet størrelsen på oppgaven samt manglende forkunnskaper, blir det satt avgrensninger i prosjektet for å fokusere på implementasjon av de viktigste funksjonene. Det er lite sannsynlig at gruppen evner å levere en robust løsning gitt tidsrommet. Vi vil derfor kun utvikle et produkt som demonstrerer de valgte konseptene. Gitt tidsrommet valgte vi å utelate design for implementasjon a PLS-system. Fokuset lå på en fungerende prototype for et køsystem, bygd på Raspberry Pi med LCD-touchskjerm, knapper, RFID-leser og kommunikasjon over en lokal server. Hovedfokus blir satt på funksjon- og systemdesign. Dette blir begrenset til funksjonene vist i tabell 1.1. Fysisk implementasjon, som kretskort blir derfor nedprioritert. Designet av prototypen skal likevel legge til rette for framtidig utvidelse og implementasjon av bus-systemet.

Innlogging og registrering av bruker
Implementasjon og avlesing av RFID-kort
Design og implementasjon av plattform
Rammeverk for sammenkobling og kommunikasjon av lokalserver og nettside
Implementasjon for input via knapper og touchskjerm
Filtrering og administrering av kø

Tabell 1.1 Planlagte implementasjoner

1.4 Budsjett

Budsjettet for installasjon av det nye køsystemet er på 100 000 NOK for alle deler. Arbeid er ikke talt som en utgiftspost. Maksimalkostnaden per enhet kan ikke overskride 1000 NOK.

1.5 Rapportens oppbygning

Gruppen har delt opp rapporten som følgende. Kapittel 1 tar for seg bakgrunnen for prosjektet, problemstilling samt avgrensinger gruppen har gjort. Kapittel 2, 3, 4 og 5 omhandler henholdsvis implementering av maskinvare, oppsett av plattform, adgangskontroll samt nettside og kommunikasjon. Her gjennomgår vi relevant teori, metode, resultat og drøfting hver for seg. Kapittel 6 legger frem det sammensatte resultatet av alle komponentene. Kapittel 7 drøfter prosjektets helhet i form av endelig system og arbeidsprosess samt forbedring og videre utvikling. Vi konkluderer oppgaven i kapittel 8.

2 Maskinvare

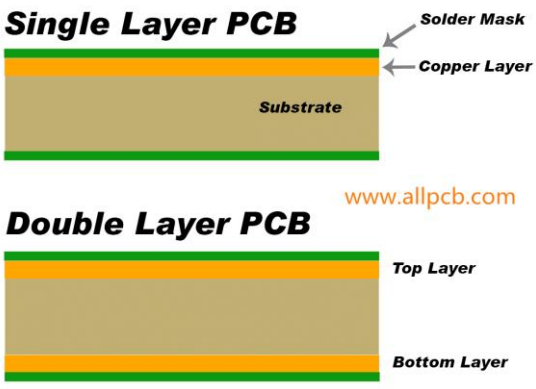
Dette kapittelet omhandler delen av prosjektet som er tilknyttet maskinvaren til køsystemet. Det vil diskutere de ulike komponentene benyttet til å utvikle systemet, hvordan de er satt sammen, og hva de brukes til. Det vil gjennomgå relevant teori knyttet til de ulike komponentene, metoden for sammensetting av systemet, sluttresultatet, og diskusjon av utfordringer, åpenbaringer og alternative metoder som ble vurdert gjennom utviklingen.

Problemstillingen presiserer bruk av en skjerm på hver av lab-plassene som skal benyttes til brukergrensesnitt for studentene, samt muligheten til å bruke NTNUs adgangskort for registrering og pålogging. Diverse maskinvare må brukes for å kunne realisere disse funksjonene, samtidig som en sentralmodul styrer disse og kommuniserer med serveren og hovedterminalen til studentassistentene. Gruppen har bestemt at dette skal styres av en Raspberry Pi som er koblet til en LCD-touch-skjerm, en RFID-leser, og navigeringsknapper, montert til et tilpasset kretskort designet av gruppen.

2.1 Teori

2.1.1 Printed circuit board

Printed circuit board, forkortet PCB og ofte bare kalt kretskort på norsk, er et brett laget for å sette sammen elektriske kretser ved hjelp av innebygde baner og monterte komponenter. Et kretskort består av ett eller flere tynne lag med kobber som er blitt etset til å danne monteringspunkter bundet sammen av elektrisk ledende baner. Kobberplatene er holdt sammen av et substrat ofte laget av fiberglass og er dekket av en loddemaske på overflaten for å isolere lederne. Komponenter er så loddet fast til disse monteringspunktene for å danne en fullstendig elektrisk krets. Kretskort er en såpass grunnleggende komponent at alt av moderne elektriske apparater, med unntak av de minste og enkleste av dem, benytter seg av disse for å fungere [59].

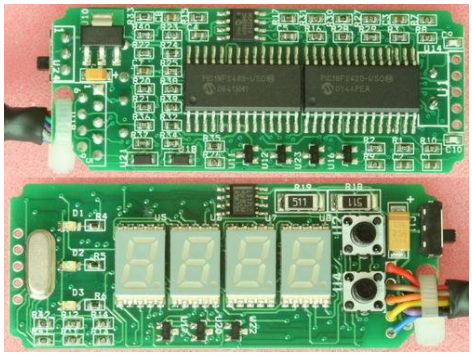


Figur 2.1 Illustrasjon av grensesnittet til et kretskort [2]

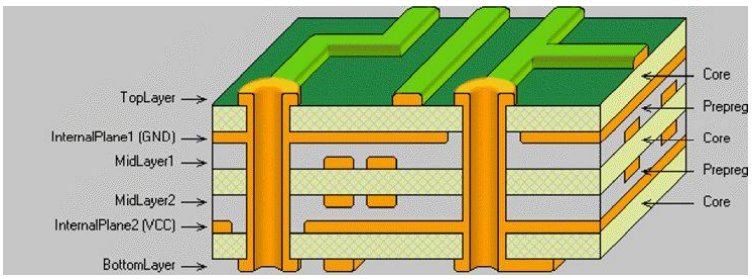
Kretskort kan skilles i tre forskjellige grunnleggende typer: ensidig, tosidig, og flerlags. Et ensidig kort er et kretskort med ett kobberlag. De har derfor elektrisk ledende baner kun på én side, og er de billigste kortene å produsere. Slike kort blir i stor grad kun brukt til å sette sammen de enkleste kretsene, og kommer sjeldent frem i moderne forbrukerelektronikk [3].

Figur 2.2 Ensidig kretskort med komponenter på en side og baner på andre side [1]

Et tosidig kort, derimot, har ett kobberlag på hver side. Dette åpner muligheten for en større krets montert på et kort av samme størrelse. I utgangspunktet er det ingen elektrisk kobling mellom de to lagene. En via er nødvendig for å oppnå dette, som fungerer som en kobberbelagt tunnel mellom de to lagene. Tosidige kort er i dag den mest vanlige typen som finnes i vanlig forbrukerelektronikk siden de har en fin balanse mellom kompleksitet og produksjonskostnader [3].

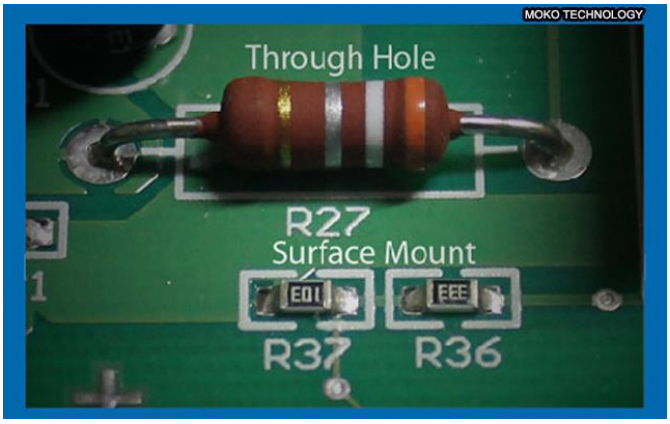

Figur 2.3 Tosidig kretskort med komponenter på begge sider [25]

Flerlagskort er kort med tre eller flere kobberlag, to på utsiden slik som i tosidige kort, og ett eller flere lag på innsiden av kortet mellom de to ytterlagene. Slike kort gjør til rede for mye mer komplekse kretser ettersom mye av arealene som vanligvis ville blitt tatt opp av baner mellom komponentene nå kan bygges inn i kortet i stedet. For at forbindelse mellom ytterlagene og de indre lagene skal oppnås på samme måte som ved tosidige kort, benyttes viaer, bare her går ikke alltid hullene hele veien gjennom kortet. Mens slike kort kan ha en større komponenttetthet, er en viktig ulempe at testing og reparasjon av slike kort fort blir vanskelig [3].



Figur 2.4 Grensesnitt av flerlagskort med to interne kobberlag [18]

For montering av komponenter finnes det to forskjellige metoder som blir brukt. De tidligste kretskortene brukte det som kalles through-hole teknologi. Through-hole montering innebærer at komponentene har ledere, eller pinner. Disse tres gjennom hull i kortene, som i stor grad fungerer på samme måte som viaer. Slike komponenter er ofte store i størrelsen, men har den fordelen at dersom de monteres på tosidige kort vil de automatisk ha kontakt med begge sider av kortet uten behovet for ytterlige viaer. De kan også enkelt loddes fast for hånd. Selv om denne teknologien i stor grad er utdatert har den fortsatt utbrett bruk i moderne elektronikk, spesielt i konstruksjon av prototyper og til bruk ved undervisning av grunnleggende elektronikk. For mer moderne elektronikk blir ofte surface-mount teknologi brukt. Surface-mount komponenter, som navnet tilsier, er komponenter montert på overflaten av kretskortet. Slike komponenter er langt mindre i størrelse, noe som tillater tettere plassering og produksjon av mindre kretskort for den samme kretsen. De minste surface-mount-komponentene kan være så små som en halv millimeter i lengden, og blir ofte montert av roboter i stedet for mennesker [37].



Figur 2.5 Through-hole- og surface-mount-varianter [37]

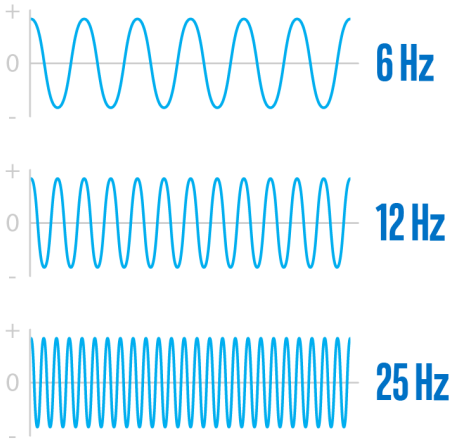
2.1.2 Datamaskin

For at systemet skal kunne fungere som en sammenhengende enhet kreves det en datamaskin i sentrum. En datamaskin er en maskin bygget opp av en rekke elektroniske komponenter som kan programmeres til å automatisk utføre logiske operasjoner og aritmetikk. En fullstendig datamaskin består av maskinvare, operativsystem, og periferienheter. Datamaskiner benyttes i en stor variasjon av elektronikk, som PCer, mobiltelefoner, spillkonsoller, og en rekke andre smart-apparater [36].

Det finnes en stor variasjon av typer datamaskiner etter hva det brukes til, og hvilke funksjoner de har innebygd. En vanlig type datamaskin brukt i elektriske produkter med lav kompleksitet er SBC-er. En SBC, kort for Single-Board Computer, er en fullstendig datamaskin montert på et enkelt kretskort. Slike datamaskiner blir ofte brukt til å produsere


Figur 2.6 Raspberry Pi SBC [4]

demonstrasjoner eller utviklingssystemer, men de kan også benyttes til å styre enkle elektriske apparater. En SBC er ofte bygget opp av en eller flere SoC-er, System on a Chip, integrerte kretser som integrerer flere av et systems komponenter inn i en enkelt krets. En slik krets inneholder nesten alltid en prosessor, minne, inngangs- og utgangsporter, og sekundær lagring [61].

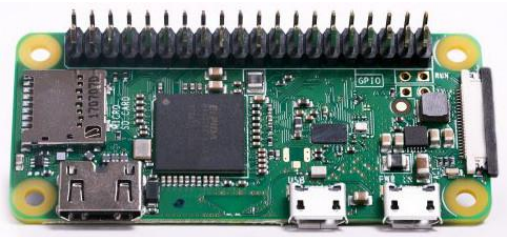


Figur 2.7 Prosessorhastighet målt i Hz [26]

I datateknologi regnes prosessoren som den sentrale komponenten i en datamaskin. Det er den som står for den største andelen av beregninger og behandling av data, og kommuniserer konstant med resten av komponentene som utgjør datamaskinens helhet. Prosessorer er klokkedrevne, register-baserte, og er bygget opp av en digital integrert krets. Prosessorer opererer på binær data og bruker dette til å kommunisere med de andre komponentene i systemet. De inneholder både kombinatorisk logikk og sekvensiell digital logikk. Hastigheten til en prosessor måles i hertz som beskriver antallet pulser klokken til prosessoren kan produsere per sekund, som regel oppgitt i gigahertz (milliarder hertz). Prosessoren kan utføre instrukser for hver gang den leser av en puls fra klokken, og avhengig av prosessoren kan den utføre en eller flere instruksjoner per puls [27].

Minne i en datamaskin regnes som midlertidig lagring av informasjon som skal brukes umiddelbart av maskinen, og blir noen ganger kalt primær lagring. Minne har lav lagringskapasitet, men opererer på en mye større hastighet enn konvensjonell lagring som harddisker. Slike harddisker er det som ofte kalles sekundær lagring og har mye større lagringskapasitet enn datamaskinens minne, men de bruker merkbart lengre tid til å hente frem denne lagrede informasjonen når det skal brukes [15].

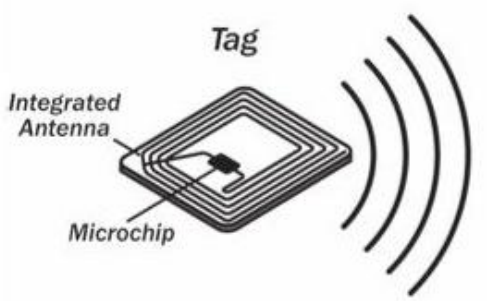
Ved mer tradisjonelle datamaskiner blir det tatt i bruk standardiserte grensesnitt som USB-porter for å koble maskinen til diverse periferienheter. Mens det finnes flere tilfeller hvor SBC-er også benytter seg av slike porter, er det mer vanlig at det er montert pinner koblet til SoC-en. Disse kalles GPIO pinner, kort for general-purpose input/output, og er pinner som kan brukes som innganger og utganger for informasjon og instrukser. Disse pinnene kobles til diverse


Figur 2.8 Raspberry Pi Zero med 40 pin GPIO header på toppen [6]

periferienheter og moduler, og benytter seg av programvare for å styre disse enhetene. For å gjøre dette kan pinnene konfigureres til flere ulike funksjoner. De mest grunnleggende av disse er å lese/skrive digitale verdier. Siden digitale verdier består av 0 og 1 har pinnene ved denne funksjonen to tilstander; høy og lav. Høy og lav er en beskrivelse av spenningsnivået over pinnene. Lav er nesten alltid 0V, og høy varierer som regel mellom 3.3V og 5V. Å lese innebærer at pinnen er koblet til en spenningskilde, og er programmert til å måle spenningsnivået over kilden. Dersom den leser et nivå rundt 0V vil datamaskinen tolke dette som den digitale verdien 0. Dersom den leser et nivå rundt 3.3V eller 5V, etter hva som er definert av den enkelte maskinen, vil den tolke dette som den digitale verdien 1. Å skrive innebærer at pinnen selv fungerer som en spenningskilde. Dersom maskinen er programmert til å skrive den digitale verdien 0 til pinnen vil den fungere som en 0V spenningskilde, og som en 3.3V/5V spenningskilde dersom den skal skrive verdien 1. Videre kan de også blant annet brukes til styring av I^2C-enheter, referert i kapittel 4.1.2. Det finnes også pinner med faste funksjoner som ikke kan konfigureres, som spenningskilder og jording [53].

2.1.3 RFID

RFID, kort for radiofrekvensidentifikasjon, er en måte å spore, lagre, og hente data gjennom brikker som er innebygd i, eller festet til et elektronisk produkt. Et RFID-system består av en sender og en mottaker. Senderen, ofte kalt en RFID-leser, avgir en elektromagnetisk puls i form av svake radiosignaler som fanges opp av mottakerens antenner. Dette aktiverer mottakeren, som er brikken innebygd i enheten, som deretter gir fra seg et svar på de mottatte signalene. Svaret er som regel informasjon innebygd i brikken som blir tolket av RFID-leseren. Fordelen med RFID-brikker, fremfor en mer tradisjonell strekkode, er at det ikke trenger å være noen visuell kontakt mellom brikken og avleseren [56].



Figur 2.9 Enkel illustrasjon av en RFID-brikke [35]

RFID-brikker kan deles inn i to hovedkategorier: passive brikker og aktive brikker. Passive brikker kalles dette fordi de ikke har en egen spenningskilde. I stedet bruker de energien fra det mottatte signalet for å sende et svakt radiosignal tilbake som svar. I store deler av forbrukerelektronikk er det denne teknologien som blir benyttet, siden det er enkelt, billig, og kan lett masseproduseres. Mest kjent er bruken deres i for eksempel adgangskort og adgangsbrikker. Aktive brikker, derimot, er koblet til sin egen spenningskilde, ofte et batteri, og har derfor evnen til å svare med et langt sterkere signal enn passive brikker. Dette fører til at de kan registreres av en RFID-leser over store avstander, i noen tilfeller over flere hundre meter. Enkelte aktive brikker kringkaster signalet sitt kontinuerlig og kalles derfor ofte fyr, fra det engelske ordet «beacon». Maskinvaren i en aktiv brikke er mer kompleks enn i en passiv brikke, og de er derfor både større i volum og dyrere å produsere [5].

Figur 2.10 Bankkort med innebygd RFID-brikke [16]

Den viktigste egenskapen ved en RFID-brikke er det at hver av dem har sin egen unike ID. Denne ID-en benyttes i en lang rekke bruksområder. Blant disse er adgangskontroll en av de mest vanlige. RFID-brikker kan brukes som nøkler, både fysisk, som ved inngangsdøren til en bygning eller rom, og digitalt for å få tilgang til en database eller annen form for programvare [56].

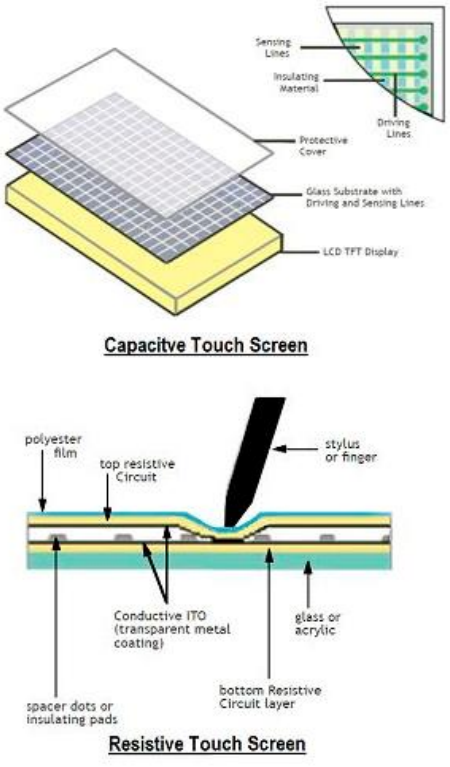
2.1.4 LCD

LCD, kort for Liquid-Crystal Display (flytende-krystall-skjerm), er en type flatpanelskjerm som er laget for å produsere bilder ved hjelp av flytende krystaller og en egenskap ved lys kalt polarisering. LCD er en av flere teknologier utviklet for å vise digitale bilder på en skjerm, og er sammen med OLED-skjermer en av de mest brukte i moderne elektriske apparater. Til tross for at LCD-teknologi er relativt utdatert, når det sammenlignes med nyere OLED skjermer, finnes det utbredt bruk i apparater hvor bildekvalitet og responstid ikke er en essensiell egenskap. LCD-er er billige å produsere og er

Figur 2.11 En enkel LCD som kan vise 2x16 symboler om gangen [64]

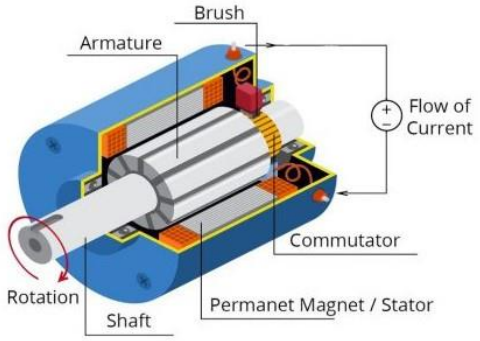
derfor svært godt egnet til små skjermer som kun skal vise enkle bilder og interaktive menyer [17].

Utenom framstilling av digitale bilder kan LCD-er også utstyres med diverse andre egenskaper for utvidet funksjonalitet. En av disse er touch-teknologi for å skape det som kalles en berøringsskjerm. En berøringsskjerm tillater en bruker å avgi kommandoer eller sende signaler til skjermen uten behovet for eksterne periferienheter. I stedet kan brukeren berøre skjermen direkte og oppnå et tilsvarende resultat. Slike skjermer skilles inn i to hovedkategorier. Den første kategorien er skjermer med resistiv touch. Resistiv touch er en metode for å føle berøring på en skjerm som innebærer bruken av flere tynne lag separert med et tynt lag luft. To av disse lagene er gjennomsiktige og dekket av et resistivt materiale, hvor det ene har loddrette, strømledende koblinger og det andre vannrette koblinger, slik at det utgjør et stort rutenett. Når det legges trykk på disse lagene, får koblingene kontakt med hverandre. Det ene laget har en spenning over seg, og det andre laget måler denne spenningen. Når disse får kontakt dannes det en spenningsdeler mellom dem. Ved å fort bytte mellom lagene kan posisjonen til det påførte trykket avleses. Den andre kategorien er skjermer med kapasitiv touch. Kapasitiv touch er en metode for å føle berøring på en skjerm som innebærer bruken av et panel som består av en elektrisk isolator, som regel glass, og en elektrisk leder. Siden menneskekroppen også er en elektrisk leder vil berøring av dette panelet forårsake en forstyrrelse av det elektrostatiske feltet til panelet, som så kan måles som en endring i kapasitansen. Ulike typer kapasitive touchskjermer bruker forskjellige metoder for å måle posisjonen til berøringen. Felles for alle er at de benytter seg av endringen i kapasitansen for å oppnå dette [33].



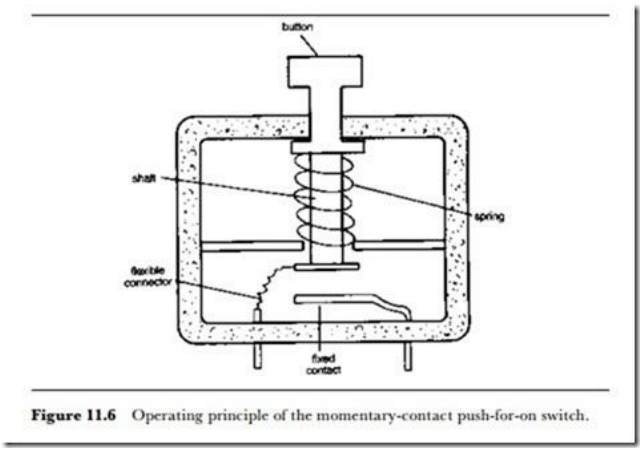
Figur 2.12 Illustrasjon av de to typene berøringsskjerm [32]

2.1.5 Elektromekanikk



Figur 2.13 En enkel motor styrt av et elektrisk signal [60]

Elektromekanikk er sammensetningen av elektrisk- og mekanisk ingeniørvitenskap og dreier seg hovedsakelig om samspillet mellom elektriske og mekaniske systemer. I elektronikkens verden handler elektromekanikk i stor grad om elektriske komponenter med bevegelige deler. Tradisjonelle komponenter er statiske i form, og det eneste som er i bevegelse er elektronene i lederne. Ved elektromekanikk, derimot, er deler av selve komponenten også i bevegelse. En elektromekanisk komponent har hovedsakelig to definisjoner: enten er det en enhet drevet av et elektrisk signal som fører til mekanisk bevegelse, eller en enhet drevet av mekanisk bevegelse som fører til et elektrisk signal. Dette kan være alt fra komplekse elektrisk drevne motorer bestående av flere titalls bevegelige deler, til enkle brytere der komponenten kun utfører én lineær bevegelse for å generere et elektrisk signal [59].



Figur 2.14 Illustrasjon av grensesnittet til en enkel trykk-knapp [34]

Slike elektromekaniske brytere finnes i alle mulige typer forbrukerelektronikk og kommer ofte i formen av trykk-knapper. En trykk-knapp er en komponent bestående av to ledere separert med et lite luftrom som for seg selv fungerer som en åpen krets. Festet til enden av den ene lederen er en fjær. Når det legges en viss mengde trykk på denne fjæren blir lederne skjøvet mot hverandre til de får kontakt og danner en lukket krets slik at et elektrisk signal kan passere gjennom dem. Når trykket lettes på, vil fjæren strekkes ut igjen og lederne vil ikke lenger være i kontakt med hverandre. Slike knapper brukes i stor grad for å styre andre elektriske komponenter ved å kontrollere når et elektrisk signal skal sendes, og hvor lenge det skal vare [20].

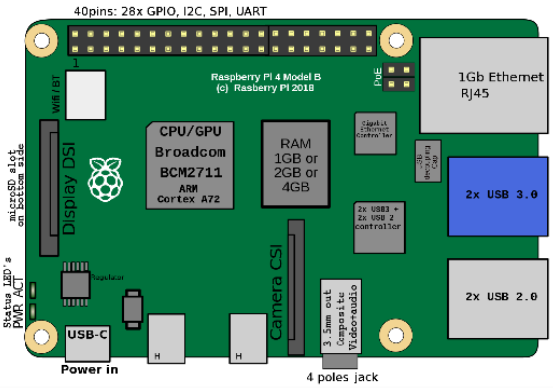
2.2 Metode

For å kunne konstruere køsystemet beskrevet i oppgaveteksten, trengs det en rekke forskjellige elektroniske komponenter. På brukersiden av systemet er det behov for en skjerm, som gruppen har bestemt at skal være en berøringsskjerm, hvor menyen til køsystemet vises. Et sett med knapper som alternativ til styring av menyen, og en RFID-leser slik at studentene kan bruke sine NTNU-adgangskort til å få tilgang til systemet er også benyttet. På utviklersiden av systemet trengs det en form for datamaskin som kan programmeres til å styre periferienhetene på brukersiden. Den må også kunne kommunisere med serveren til NTNU for å hente nødvendig informasjon om studenten og faget de er meldt inn i. Til slutt skal periferienhetene kobles til datamaskinen via et tilpasset kretskort laget for å lette på bruken av løse ledninger.

2.2.1 Utstyr

Datamaskin – Raspberry Pi 4 Model B

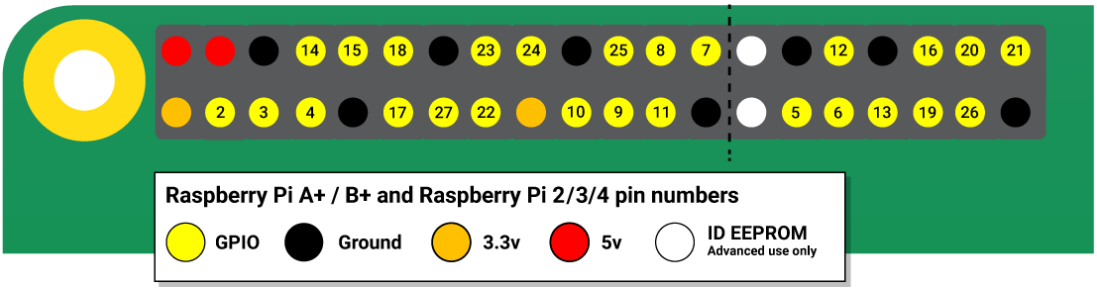
Datamaskinen benyttet i dette prosjektet er en Raspberry Pi 4 Model B, en SBC produsert av Raspberry Pi Foundation. Raspberry Pi-datamaskinene er en rekke open-source-enheter laget for å fremme undervisningen av grunnleggende datateknologi på skoler og i utviklende land [52]. De er små og enkle i konstruksjonen, og egner seg godt til prosjekter og elektroniske produkter som ikke trenger en spesielt kraftig datamaskin for å fungere. Raspberry Pi 4 Model B er bygget opp av en Broadcom BCM2711 SoC. BCM2711 består av en 1.5 GHz Quad-core Cortex-A72 prosessor, har tilgang til opptil 8 GB minne, er koblet til 40 inngangs- og utgangsporter, hvorav 28 er GPIO pinner, fire USB-porter og en Ethernet-port [55].



Figur 2.15 Illustrasjon av Raspberry Pi 4 Model B [30]

Per i dag er Model 4B den raskeste Raspberry Pi-enheten på markedet, 50% raskere enn sin forgjenger Model 3B+. Maskinen benytter seg av varmespredningsteknologi, noe som tillater enheten å operere under økt press uten risiko for overoppheting. Koblet til prosessoren er 2x20 tilkoblingspunkter med variert funksjonalitet. To av disse er koblet direkte til

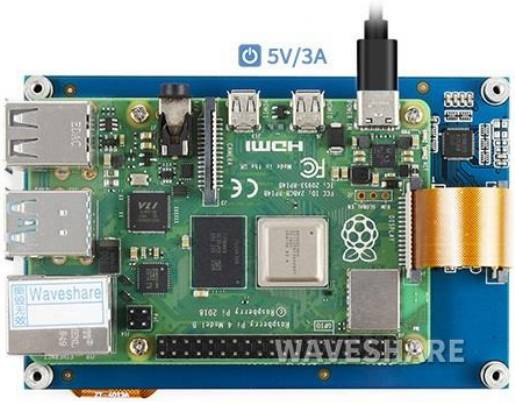
spenningsforsyningen og fungerer som 5V spenningskilder. Videre er det også to andre pinner som fungerer som spenningskilder, i dette tilfellet på en redusert 3.3V for bruk ved komponenter som er mer sensitive til høyere spenning. Spredt utover de to radene er det til sammen 8 jordingsporter som er koblet tilbake til spenningsforsyningen, slik at det kan dannes en lukket krets mellom datamaskinen og periferienhetene. I tillegg til disse er det også montert to såkalte ID EEPROM porter, men ettersom de ikke blir benyttet i dette prosjektet kan de overses. Til slutt er det totalt 28 GPIO pinner som individuelt kan programmeres etter behov [55].



Figur 2.16 Diagram av tilkoblingspinnene på Raspberry Pi Model 4B [52]

Skjerm – 4.3inch DSI LCD for Raspberry Pi

Skjermen som skal vise menyen til køsystemet er en 4.3 tommer LCD med kapasitiv touch. Denne skjermen er spesielt laget for bruk med diverse Raspberry Pi-enheter og kobles enkelt til datamaskinen med en flatkabel. Flatkabelen er direkte koblet til spenningsforsyningen til Raspberry Pi-enheten og trenger derfor ikke en ekstern spenningskilde. Skjermen er utstyrt med monteringshull i hvert av de fire hjørnene samt fire ekstra hull med ekvivalent posisjonering som ved Raspberry Pi-enheten. Slik kan de enkelt festes til hverandre for å forhindre løse komponenter. Skjermens kontroller er montert til et kretskort som er 68 mm i høyden og 106 mm i bredden.



Figur 2.18 Skjermens bakside med en Raspberry Pi 4B montert til skruehullene [69]

Figur 2.17 Skjermens framside [69]

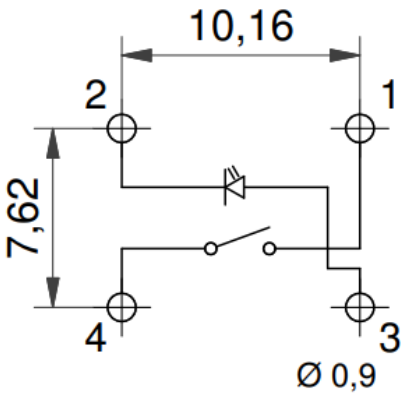
RFID-leser – MFRC522 Modul

MFRC522 er en integrert krets laget for å lese passive RFID-brikker og skrive ut informasjonen til en mikrokontroller eller datamaskin. Den integrerte kretsen er montert på et kretskort sammen med en rekke andre komponenter, som en antenne, for å utgjøre en modul som enkelt kan kobles til andre enheter og kommunisere med disse. Modulen har åtte tilkoblingspunkter, en for spenningsforsyning, en for jording, og seks logiske innganger for kommunikasjon med kontrollenheten. Modulen støtter flere ulike grensesnitt og er derfor svært fleksibel i forhold til hvilke kontrollere og datamaskiner det kan kommunisere med. Modulen opererer på et spenningsnivå mellom 2.5V og 3.3V, mens de logiske inngangene opererer på 5V signaler. Antennen har en avlesningsrekkevidde på 5 cm og en frekvensrekkevidde på 13.56 MHz [46].

Figur 2.19 MFRC522 RFID modul

Navigasjonsknapper – Navimec 5G 1ZB/1ZCS

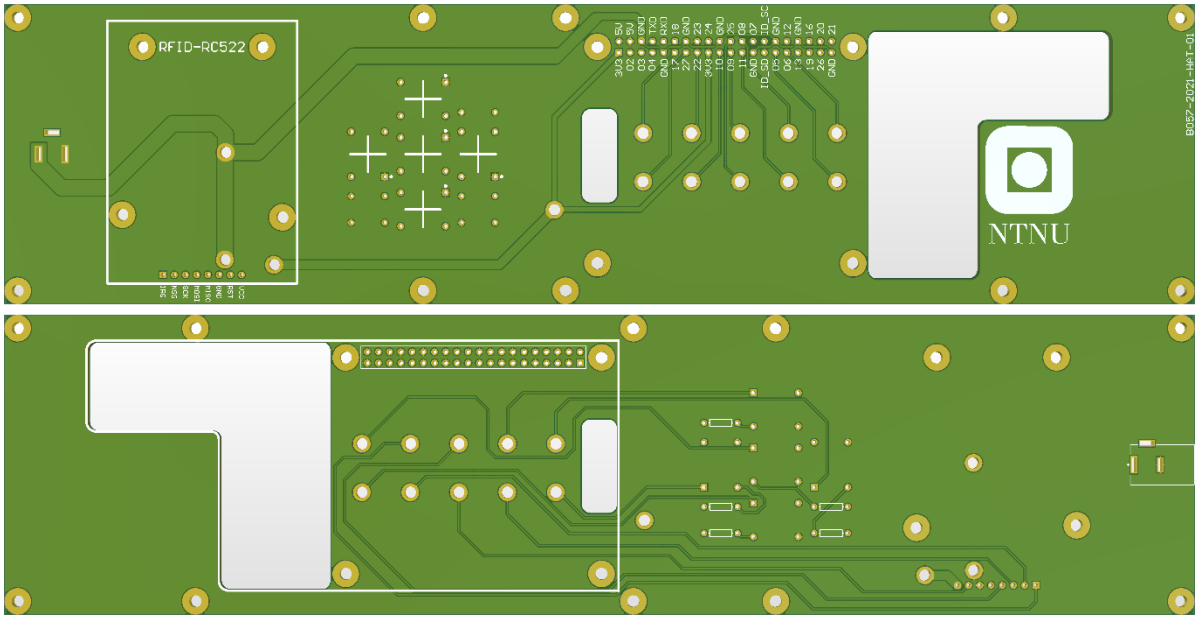
Navimec 5G er robuste trykk-knapper laget for langvarig bruk med minimal slitasje, ment for bruk til å navigere digitale menyer. Knappene kommer som både through-hole og SMD komponenter og utstyres med en av to forskjellige hatter etter ønsket bruk. Disse er 1ZB for pil-knapp og 1ZCS for OK knapp. Knappene har også innebygde LED-lamper og lyser derfor opp dersom de blir forsynt med en spenning. De monteres ved fire tilkoblingspunkter, to tilhører knapp-funksjonen og to tilhører LED-lampen. I dette prosjektet benyttes 5 slike knapper av typen through hole, 4 med 1ZB hatter og 1 med 1ZCS hatt [41].



Figur 2.20 Skjemategning av knapp med LED [42]

HAT – Tilpasset kretskort

En HAT, Hardware Attached on Top, er et kretskort som kobles til en SBC via tilkoblingspinnene for å utvide funksjonaliteten til datamaskinen. I dette prosjektet trengs det et kretskort som kan sørge for enkel tilkobling av de ulike periferienhetene til Raspberry Pi-enheten. Her er det blitt laget et enkelt tosidig kretskort. Den består av through-hole monteringspunkter slik at de ulike komponentene kan loddes fast via tilkoblingspinnene deres og baner som kobler disse pinnene sammen. Den inneholder også flere skruehull slik at komponentene kan trygt festes til kortet, og kortet kan monteres på lab plassen. Direkte over der Raspberry Pi-enheten monteres er det skjært ut to hull. Slik kan USB-portene og Ethernet-porten, samt flatkabelen til skjermen kan passere gjennom kortet. Raspberry Pi-enheten monteres på baksiden av kortet, mens de andre komponentene monteres på framsiden. Normalt når et kort produseres i en fabrikk kan kortene enkelt utstyres med viaer, men ettersom det er ønskelig å kunne produsere kortet på laben på campus, som ikke kan gjøre dette, må viaene tilpasses. Hullene er derfor langt større enn det de normalt ville vært slik at det kan loddes en kobling mellom de to sidene av kortet. Denne hindringen påvirker også monteringen av through-hole komponentene. Siden det ikke er noen kobling mellom de to sidene av hullene vil komponentene kun ha tilkobling til banene på den siden de er loddet fast [59].



Figur 2.21 Fremside og bakside av HAT uten komponenter

Ytterlige komponenter

For at oppkobling skal være fullstendig trengs noen få ekstra komponenter. Disse inkluderer fem motstander på 470 ohm hver (komponentnummer MBB02070C4700FCT00), en 2x20 female pin header (komponentnummer 61304021821), og en enkel female DC nettplugg (komponentnummer NEB 21 R).

Kostnader

Komponent	Pris [NOK]
Skjerm	344.00
RFID-Leser	39.00
Datamaskin	377.13
Kretskort	24.64
Ytterlige komponenter	~40.00
Sum	**824.77**

Tabell 2.1 Komponentkostnader (se vedlegg T)

2.2.2 Montering

Alle komponentene i systemet skal kobles sammen via det tilpassede kretskortet, som kan deles inn i tre seksjoner: til venstre der RFID-leseren monteres, i midten der navigeringsknappene monteres, og til høyre der skjerm og Raspberry Pi monteres. Raspberry Pi-enheten monteres på baksiden av kortet. Her er det loddet fast en 2x20 pin female header markert med navnet på de ulike tilkoblingene. Denne headeren monteres direkte på tilkoblingspinnene til Raspberry Pi-enheten slik at SBC-en får kontakt med kretskortet. Videre er det fire skruehull i kretskortet på lik linje med de fire skruehullene til SBC-en. Her monteres det søyleskruer med samme høyde som headeren slik at SBC-en er fullstendig festet til kortet.

På fremsiden av kortet er det ytterlige fire skruehull på lik linje med de fire skruehullene til LCD-en. Her òg monteres søyleskruer slik at skjermen er festet til kretskortet. Skjermen har ingen direkte tilkobling, men skal i stedet kobles til Raspberry Pi-enheten via en flatkabel. Denne flatkabelen kobles til SBC-en sin designerte port for skjermer, som videre tres gjennom det minste av det to utskårede hullene i kretskortet. Til slutt kobles kabelen til skjermens tilsvarende port slik at det er kontakt mellom de to enhetene.

Til venstre for skjerm og Raspberry Pi er det en gruppe hull egnet for navigeringsknappene. Disse hullene er på fremsiden markert med fem pluss tegn for å indikere sentrum av de fem forskjellige knappene. Hver knapp loddes fast til fire hull der hullet designert pin 1 er markert

med en hvit prikk. Det er viktig at knappens pin 1 samsvarer med hullet med den hvite prikken slik at koblingene blir riktige. Da de fem knappene er loddet fast til kortet gjenstår det fortsatt 10 hull. Til disse skal det loddes 5 separate motstander på 470 ohm som er koblet til knappene i en ende og til 5V spenningskilden i andre enden. De er koblet i serie med knappenes interne LED-lamper for å begrense strømmen som går gjennom disse. Internt i knappene er selve knapp funksjonen koblet til pin 1 og 4, og LED-lampen koblet til pin 2 og 3. Pinnene til knappene er koblet henholdsvis lik tabell 2.2.

Venstre	GPIO 17
Høyre	GPIO 27
Opp	GPIO 22
Ned	GPIO 5
OK	GPIO 6
Pin 1	Raspberry Pi GPIO
Pin 2	Jord
Pin 3	Motstander
Pin 4	Jord

Tabell 2.2 Oppkobling av knapper (se vedlegg A)

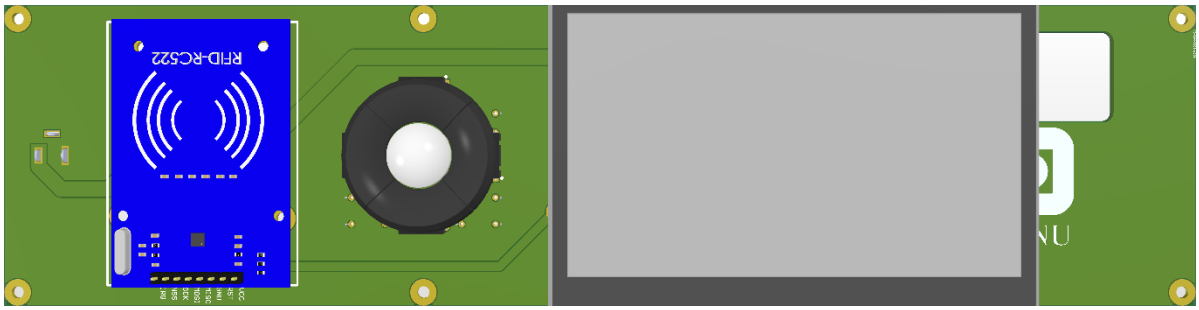
Videre til venstre monteres RFID-leseren. Her er de åtte tilkoblingspinnene loddet direkte til kretskortet, i motsetning til Raspberry Pi-enheten, som er koblet til en ekstern header. I tillegg til disse er det også fire skruehull i kortet på lik linje med de fire skruehullene på RFID-modulen. Oppkoblingen er vist i tabell 2.3.

VCC	3.3V på SBC
RST	GPIO 25
GND	Jord
MISO	GPIO 9
MOSI	GPIO 10
SCK	GPIO 11
NSS	GPIO 8
IRQ	GPIO 24

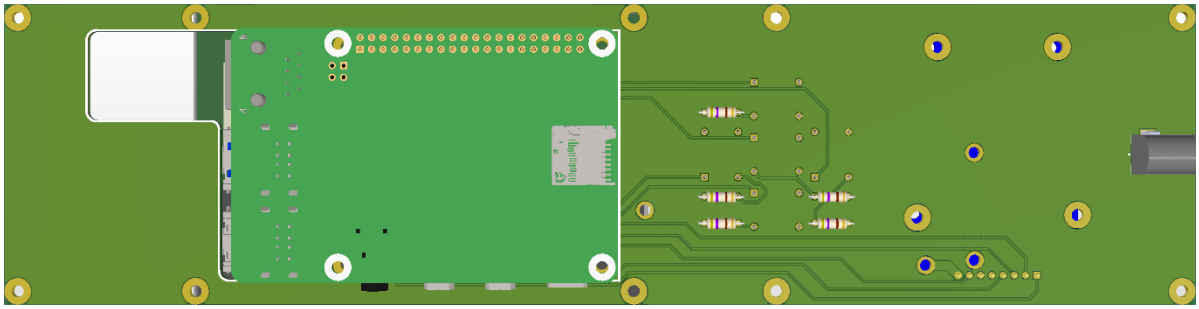
Tabell 2.3 Oppkobling av RFID (se vedlegg A)

På enden av venstre side av kortet monteres DC pluggen på baksiden markert med tre avlange hull. Ett hull er spenningskildens 5V og ett annet er spenningskildens jording. Det siste er der kun for montering av komponenten og er derfor ikke koblet elektrisk til resten av systemet. Denne pluggen kobles til en ekstern spenningskilde inne på laben.

Når alle komponentene er på plass monteres kretskortet med skruer til selve lab-plassen. Dette blir gjort inni kanalen som går langs skilleveggen mellom lab-plassene der alle andre ledninger til resten av utstyret allerede befinner seg. Skruer blir festet gjennom de seks resterende skruehullene på kortet til innsiden av denne kanalen. Se skjemategning, vedlegg A, for mer nøyaktig beskrivelse av tilkobling.



Figur 2.22 Fremside av ferdigmontert system



Figur 2.23 Bakside av ferdigmontert system

2.3 Utfordringer og åpenbaringer

Når det gjelder maskinvare er ofte den største utfordringen forsyningen av selve komponentene, og dette prosjektet er på ingen måte et unntak. Det er svært få norske leverandører av nødvendig elektronikk. De få som finnes har enten svært dårlig utvalg eller altfor høye priser. Utenlandske leverandører er derfor det eneste alternativet, men disse kommer også med sine egne utfordringer.

Den mest åpenbare utfordringen er leveringstid. Avhengig av hvor i utlandet varen kommer fra kan leveringstiden være alt fra et par dager til flere uker. Dette gjør at nøye planlegging er

viktig for at arbeidet skal gå effektivt. Gruppen gikk nøye gjennom vurdering av hver eneste komponent før bestilling for å være sikker på at det var den rette varen til prosjektet. Dersom gruppen i senere tid innså at komponenten ikke var like godt egnet som tidligere antatt var det stor risiko for å utvide ventetiden. Dette ville satt en stopper for hele prosjektet. Her var den største sinkeren det tilpassede kretskortet. Produksjon av PCB-er foregår hovedsakelig i Kina dersom en ønsker en fornuftig pris per enhet. Leveringstiden på det første kortet var antatt å være 10 dager, men på grunn av komplikasjoner med bestillingen var ikke kortene ankommet før totalt 20 dager etter bestilling. Dette kortet var ment hovedsakelig som en test for å sørge for at målene på skruehullene var riktige, og det skulle derfor gjøres en ny bestilling dersom justeringer ble nødvendig. Gruppen opplevde at justeringer var absolutt nødvendig. Skruehullene til RFID-leseren var ikke riktig plassert og de to hullene til Raspberry Pi-enheten ble ikke skjært ut. Ettersom kortet ankom såpass sent, var det ingen tid igjen til å bestille det på nytt. Det var mulighet for å gjøre disse justeringene på campus, men det ble bestemt at gruppen ikke skulle bruke mer tid på kretskortet ettersom de mest nødvendige testene kunne gjøres uten. Det viktigste for prosjektet var at designet eksisterte digitalt og at den nødvendige dokumentasjonen var utført.

En annen utfordring som ofte kommer frem, er leverandørens lagerbeholdning. I mange tilfeller kan det oppstå mangel på en vare, og et midlertidig alternativ må derfor finnes. Dette ble et problem med LCD og RFID-leser. Gruppen undersøkte hos flere forskjellige leverandører, men for både skjerm og RFID-leser var det fullstendig mangel på varene. Gruppen visste allerede at de valgte komponentene var de som skulle brukes i systemet, men for å kunne teste det var det nødvendig å ha disse komponentene fysisk til stede. I skjermens tilfelle var ikke løsningen verre enn å finne en annen skjerm med tilsvarende funksjonalitet. Gruppen kom frem til å benytte seg av en større variant av den ønskede skjermen. RFID-leseren viste seg å være et langt større problem. Hverken den ønskede modulen, eller noen tilsvarende moduler var tilgjengelige på aktuelle nettsider. Det var kun mulig å få tak i disse på sider med altfor høy enhetspris eller leveringstid. Heldigvis fantes det én RFID-leser på campus som gruppen fikk utstedt til å utføre prosjektet. På grunn av dette ble det aldri mulighet for å teste systemet med flere fullstendige enheter. Den nye RFID-leseren var en VMA405-modul.

Ved design av det tilpassede kretskortet var mål for montering av komponentene svært viktig, og plassering av skruehullene skapte de største problemene. For å vite avstand mellom de forskjellige hullene var det nødvendig med detaljert dokumentasjon for hver komponent.

Dette var spesielt et problem for RFID-leseren, hvor den eneste dokumentasjonen av målene som fantes var en enkel tegning med dårlig bildekvalitet. På grunn av dette endte målene med å være feil, og designet måtte derfor justeres.

2.4 Resultat

Alle de essensielle komponentene for testing passer sammen og gir et resultat med ønsket funksjonalitet. Periferienhetene kommuniserer godt med datamaskinen, og skjermen viser den nødvendige informasjonen, i tillegg til å være lett å bruke. Det tilpassede kretskortet endte med å ikke ble ferdigstilt på grunn av mangel på tid, men på papiret skal designet fungere etter ønskede krav. Den totale størrelsen på systemet ble aldri undersøkt og det er fortsatt usikkert om det er mulig å montere det i kanalen på laben, men dersom de gitte spesifikasjonene for kanalen er riktig burde passform ikke være et problem. Den største usikkerheten er foreløpig banene på kortet. Disse er ikke blitt testet og det er derfor ikke klart om disse må tegnes om for at kortet skal fungere som ønsket. Budsjettet ble tatt hensyn til og den totale enhetsprisen endte med å ligge under 1000 NOK.

2.5 Alternative løsninger

2.5.1 Alternativ datamaskin

Valget av datamaskin var viktig å få til riktig, ikke bare fordi den er kjernen som driver hele systemet, men også fordi den utgjør den største andelen av enhetsprisen per lab-plass. Det ble derfor vurdert et par ulike alternativer for å få redusert kostnadene uten å påvirke systemets funksjonalitet for mye.

Mikrokontrollere

En mulig løsning som drastisk ville redusert kostnadene var å ta i bruk enkle mikrokontrollere i stedet for fullstendige datamaskiner. Mikrokontrollere kan anses som svært forenklede datamaskiner og har derfor ikke muligheten til å drive en vanlig skjerm slik som en datamaskin kan (se 2.5.2 Alternativ skjerm). En mikrokontroller kan heller ikke kobles til en server, noe som systemet trenger for å kunne benytte seg av NTNU-kortene for adgangskontroll. Løsningen ble derfor designet slik at hver lab-plass var koblet til en enkel mikrokontroller. 8-

12 av disse mikrokontrollerne ble da koblet til en datamaskin, mest sannsynlig en Raspberry Pi enhet, som var montert i enden av hver rad på laben. Gruppen kom fort frem til konklusjonen at denne løsningen ikke var levedyktig på grunn av den komplekse koblingen av systemet, og at de reduserte kostnadene ikke var vært det nødvendige ekstra arbeidet for å få et fungerende system.

Raspberry Pi Zero

Dersom den nødvendige prosessorkraften ikke var spesielt høy, var en Raspberry Pi Zero et godt alternativ til den valgte datamaskinen. Den har redusert kraft og er langt billigere enn en Model 4B. Hovedproblemet oppstår ved antallet GPIO pinner. Siden Zero er ment som en forenklet versjon av standard versjonene av Raspberry Pi-enhetene har den også redusert antall tilkoblingspinner. Systemet har svært høy bruk av GPIO pinner og dersom systemet skal ha mulighet for utvidelser vil det være nødvendig å ha flere av disse til overs. Vurdert kun med hensyn til prosessorkraft og pris er Raspberry Pi Zero godt egnet til prosjektet, men når fremtidig utvikling også skal tas hensyn er den ikke nok og kan derfor ikke brukes.

Raspberry Pi Model 3B+

Model 3B+ er forgjengeren til Model 4B og har derfor ikke den samme prosessorkraften som de nyeste modellene. Likevel er den tilgjengelige kraften fortsatt nok til den tenkte bruken, og siden antallet GPIO pinner er den samme ble Model 3B+ regnet som det beste alternativet for systemet. Etter videre undersøkelse, derimot, fant gruppen ut at prisforskjellen mellom 3B+ og 4B var lik null, og forslaget ble derfor forkastet ettersom 4B fremtidssikrer systemet mer enn det 3B+ gjør, helt uten ekstra kostnader.

2.5.2 Alternativ skjerm

Skjermen er essensiell for at systemet skal være mest mulig intuitivt for studenten. Det må være enkelt og brukervennlig, men også kunne vise nødvendig informasjon for at studenten faktisk skal ha nytte av systemet. For å oppnå disse kravene finnes det alternativer til den valgte skjermen.

Enkel LCD skjerm

En svært enkel og minimalistisk LCD som kun viser ren tekst. Skjermen består av fire rader som hver rommer opptil 40 symboler. En slik skjerm trenger ekstremt lite prosessorkraft for å drives og er svært billig å produsere, slik at prisen per enhet faller drastisk. En slik skjerm krever såpass lite kraft at den også kan styres av en vanlig mikrokontroller. Det kan argumenteres at siden informasjonen er såpass forenklet er det et mer brukervennlig alternativ, men på grunn av restriksjonene i antallet symboler som kan vises på en gang kan det fort oppleves som klønete. En enkel LCD vil være lett å vedlikeholde, men er også lite fleksibel dersom systemet skal utvides med nye funksjoner i fremtiden. Modellen for dette alternativet ville vært en NHD-0440AZ-RN-FBW LCD modul.

Figur 2.24 NHD-0440AZ-RN- FBW [44]

LCD med resistiv touch

Etter det ble bestemt at den enkle LCD-en ikke var fleksibel nok for fremtidig utvikling, og ble regnet som for liten i størrelse, kom gruppen frem til at en mer detaljert skjerm var nødvendig. Vanlige skjermer i den rette størrelsen var vanskelige å få tak i og var ofte mye dyrere enn det budsjettet tillot. Det ble derfor diskutert bruk av berøringsskjermer, som var langt mer tilgjengelige. Utenom den valgte berøringsskjermen ble det også vurdert en skjerm med resistiv touch. En slik skjerm ville kreve bruk av en form for berøringspenn dersom touch funksjonene skal tas i bruk. En slik penn kan fort oppleves som klønete og ble derfor ansett som lite brukervennlig. Gruppen kom derfor frem til løsningen å bruke en resistiv berøringsskjerm uten å ta i bruk touch funksjonene. I dette tilfellet vil menyen eksklusivt bli navigert av de monterte trykk-knappene.

Denne løsningen har god brukervennlighet ettersom informasjonen vist på skjermen er langt bedre enn på den enkle LCD-en, men sliter fortsatt med den ønskede fleksibiliteten for fremtidige utvidelser. Eksklusiv bruk av knapper skaper restriksjoner for programvare som kan skrives. På en annen side, er også dette alternativet enkelt å vedlikeholde, ettersom skjermen aldri blir utsatt for fysisk bruk. Direkte kontakt er den største kilden for slitasje ved berøringsskjermer. Ved å unngå dette åpnes det for at skjermen kan tas i bruk over lengre tid. Sammenlignet med den kapasitive berøringsskjermen var et system med resistiv skjerm antatt å være rundt 110 NOK billigere per enhet, uten å ta hensyn til vedlikeholdskostnadene. Likevel ble det bestemt at prisforskjellen ikke var verdt funksjonaliteten som ble ofret til gjengjeld.

3 Oppsett av plattform

I dette kapittelet skal vi se på hvordan vi kom frem til en fungerende prototype av plattformen som vi brukte til å kjøre køsystemet vårt på. Her vil vi begynne med å gå igjennom en del teori som vi mener er relevant for å forstå hvordan systemet virker. Deretter vil vi gjennomgå arbeidsprosessen for utvikling av prototypen. Vi vil deretter presentere resultatet av denne arbeidsprosessen, hvor vi vil se på den endelige prototypen. Til slutt vil vi drøfte valgene som vi har gjort underveis i prosjektet. Her vil vi også se på alternativer som kunne blitt utført, og vil begrunne disse.

3.1 Teori

3.1.1 BASH-skript

BASH står for "Bourne Again Shell", og er et dataprogram som lar oss gjøre ulike ting på datamaskiner ved hjelp av tekstkommandoer [21]. Måten dette blir gjort på, er at man skriver inn kommandoene i BASH. Disse blir tolket av programmet og videre utført. Med BASH-skripting menes det at vi lager en liste over flere kommandoer som kjøres etter hverandre. På denne måten kan vi samle flere kommandoer og kjøre dem mer effektivt. Skriptet er egentlig bare en tekstfil som inneholder de samme kommandoene en ville ha skrevet inn i BASH.

3.1.2 Daemon

En *daemon* er et dataprogram som kjører i bakgrunnen på en datamaskin, og som ikke er beregnet for å brukes direkte av brukeren [29]. Disse daemon-ene kan ha ulike oppgaver på datamaskinen, som for eksempel å håndtere nettverkskommunikasjon, eller å overføre informasjon til printere. Daemon-er blir ofte startet i oppstartsfasen til datamaskinen. Dette blir ofte gjort via *SystemD* i flere operativsystemer basert på GNU/Linux.

3.1.3 Python

Python er et imperativt, objektorientert høynivå programmeringsspråk. Språket er veldig populært grunnet enkel lærekurve, og er godt egnet til små og store prosjekter. Takket være støtten for utvidelser gjennom rammeverk og biblioteker, er Python brukbart innenfor flere arbeidsområder. Python er en essensiell del av flere operativsystem, som Linux distribusjoner. Ved utvikling på Raspberry Pi benyttes hovedsakelig Python, grunnet dets allsidighet, kraft og brukervennlighet [70].

3.1.4 DHCP-server

DHCP står for «Dynamic Host Configuration Protocol", og er en protokoll som brukes til å konfigurere nettverksparametere for enheter som kobler seg på et nettverk. Dersom vi har en DHCP-server på et nettverk, kan man få denne serveren til å sende ut nettverksinformasjon. Denne informasjonen spesifiserer hvilken DNS-server og hvilken IP-adresse som kan brukes av klienten. Dette gjør at man ikke trenger å konfigurere hver ting som kobles på nettverket manuelt. En populær DHCP-server som er tilgjengelig heter «ISC DHCP», og denne kan installeres relativt enkelt på en server som kjører operativsystemet *Ubuntu Server*. Den kan konfigureres via tekstfiler som ligger lagret på servermaskinen [11].

3.1.5 DNS-server

DNS står for «Domain Name Service», og har blant annet som jobb å finne ut hvilken IP-adresse som stemmer over ens med en gitt nettsideadresse. Dette gjør at dersom man vil gå til en bestemt nettside, slipper man å huske den numeriske IP-adressen til nettsiden. En trenger derfor kun å huske navnet på nettsiden, så ordner en DNS-server opp i resten [10].

En annen positiv egenskap ved å bruke en DNS-server, er at man kan skille mellom flere nettsider som blir levert fra samme IP-adresse. Et eksempel på dette er at man kan ha flere nettsider, for eksempel «student.pi» og «studass.pi», som begge blir sendt fra samme webserver og er tilgjengelig på samme IP-adresse. Dette er fordi webserveren kan sjekke hvilket domenenavn som ble brukt for å hente IP-adressen. En vanlig DNS-server tilgjengelig for datamaskiner som bruker operativsystemet «Ubuntu Server», heter BIND9. BIND9 står for «Berkeley Internet Name Domain 9». Denne DNS-serveren kan konfigureres ved hjelp av tekstfiler som ligger lagret på datamaskinen. Man har mulighet for å sette opp serveren på

flere forskjellige måter. Ved å sette BIND9 opp som en såkalt *primær server*, kan en sette opp eget domene, og bruke dette til å gi navn til nettsider [10].

3.1.6 Klient-server

I datakommunikasjonssammenheng er klient-server-modellen mye brukt i datanettverk. Modellen går ut på at en datamaskin lytter etter forespørsler fra andre maskiner på nettverket. Dersom datamaskinen får en forespørsel, sender den et svar tilbake til maskinen som sendte forespørselen. Datamaskinen som lytter på nettverket kalles en *server*. Datamaskiner som kommer med forespørsler kalles *klienter* [57].

3.1.7 Raspberry Pi OS Lite

Raspberry Pi OS Lite er et operativsystem som er utgitt av Raspberry Pi Foundation. Operativsystemet er basert på distribusjonen «Debian Buster» av GNU/Linux. Det er laget for å ta lite lagringsplass, og være lite krevende for maskinen. Et resultat av dette er at det ikke følger med et grafisk brukergrensesnitt, og man må kommunisere med Raspberry Pi-en ved hjelp av tekstkommandoer [14].

3.1.8 SystemD

SystemD er en samling av dataprogrammer for mange operativsystemer som er basert på GNU/Linux. Disse programmene har ulike oppgaver, men en av funksjonene til SystemD er å lage et felles grensesnitt mot daemon-er og servicer som kjører på datamaskinen. SystemD blir kallet når datamaskinen starter opp, og har som oppgave å starte alle andre prosesser som skal kjøre i bakgrunnen [31].

Måten SystemD vet hvilke prosesser den skal kjøre ved oppstart, er ved såkalte *service-filer* som ligger lagret på bestemte plasser på datamaskinen. For hver bestemt prosess som skal startes, er det en tilhørende service-fil som inneholder informasjon om hvordan SystemD skal behandle den.

3.1.9 Webserver

En *webserver* er et dataprogram som har som hovedoppgave å levere nettsider til klienter som forespør dem [12]. Webserveren lytter etter forespørsler på nettverket. Når den mottar en slik forespørsel, sender den en nettside i form av HTML tilbake til klienten som sendte forespørselen. I tillegg sendes et skript som kan kjøres i klientens nettleser, og informasjon om hvordan nettsiden skal vises.

Apache2 er en webserver som blir mye brukt på servere med operativsystemer basert på GNU/Linux. Denne webserveren er modulær, og man kan bruke forskjellige moduler til å få webserveren til å virke etter ønske. Et eksempel på dette, er at en kan laste ned en modul for å bruke programmeringsspråket PHP til å lage nettsider. En kan også laste ned moduler for databaser slik som det populære databasesystemet MySQL. Slik vil man ha tilgang til databasen fra nettsiden.

3.1.10 X-Server og Openbox

X-Server er en prosess som behandler input og output for grafiske brukergrensesnitt [64]. Prosessen gjør det mulig å ha et grafisk brukergrensesnitt på en datamaskin. Openbox er en såkalt *window manager*, som styrer utseende og plasseringen av de grafiske elementene på skjermen.

3.1.11 IP-adresse

En IP-adresse er en adresse for enheter tilkoblet et felles nettverk. Adressene brukes for å kommunisere med hverandre [66]. Denne adressen består av to deler. Den ene delen er en nettverksadresse, og den andre delen er en vertsadresse. For at enheter skal kommunisere med hverandre, bør de være på samme nettverk. Det vil si at to enheter som skal kommunisere med hverandre på et nettverk, må ha samme nettverksadresse. Vertsadressen derimot er unik for de forskjellige enhetene på et gitt nettverk. To måter å spesifisere en IP-adresse for en nettverksenhet på, er enten å sette den manuelt, eller å få en DHCP-server til å sette den.

3.1.12 SSH

SSH står for *Secure Shell*, og er en protokoll som ofte brukes for å styre datamaskiner over et nettverk [62]. Ved å bruke SSH, kan man få tilgang til en annen datamaskin sin terminal, og gi kommandoer til disse. SSH er populært ved bruk av servere, og kan brukes uten et grafisk brukergrensesnitt.

3.2 Metode

3.2.1 Utstyr

Raspberry Pi 4 model B

En Raspberry Pi 4 model B (se kapittel 2.2.1) er en datamaskin på størrelse med et kredittkort. Alle komponenter som datamaskinen trenger for å kjøre, er montert på dette kretskortet. Man kan også koble datamus, tastatur og skjerm til datamaskinen dersom man ønsker det. I prototypen til prosjektet vårt, bruker vi fem Raspberry Pi 4 model B. Fire av disse virker som klienter, mens én fungerer som en server.

Strømforsyning for Raspberry Pi

For at Raspberry Pi 4 model B skal virke ordentlig er det viktig at vi har en strømforsyning som kan levere nødvendig effekt. På Raspberry Pi Foundation sine sider anbefales det en strømforsyning med spenning på 5,1V, og som kan levere 3,0A. Adapteren som vi brukte i prototypen vår følger denne anbefalingen, og er altså på 5,1V og 3,0A [55].

Touch-skjerm

Touch-skjermen som vi brukte i prototypen er en 7-tommers skjerm fra leverandøren *Element14*. Denne skjermen er 194mm bred, 110mm høy, og 20mm tykk. Oppløsningen på skjermen er 800x480 piksler. Det følger med en *DSI sløyfekabel* som vi brukte til å kople touch-skjermen til vår Raspberry Pi. Skjermen har også feste for montasje av SBC-en på baksiden, og vi brukte medfølgende standoffs og skruer til å sette dem sammen.

Nettverkskabler

For at prototypen skal kunne kommunisere via et nettverk brukte vi *TP-kabler* som var ferdig terminert med RJ45-konnektorer i hver ende. Vi endte opp med å bruke 8 slike kabler for å kople sammen prototypen.

Svitsj for nettverk

Vi brukte to svitsjer for å koble sammen de forskjellige enhetene som skulle være på nettverket. Svitsjene som vi brukte har 5 porter for tilkopling av RJ45-plugger, er fra produsenten Netgear, og har modellnummer GS105Ev2.

MicroSD-kort

Et MicroSD-kort virker som lagringsplass for klientplattformen. Dette minnekortet skal inneholde operativsystemet, og alle filene som plattformen skal bruke.

Operativsystem

Vi bruker operativsystemet *Raspberry Pi OS Lite* for klientdelen i systemet vårt. Dette operativsystemet er tilgjengelig fra Raspberry Pi Foundation sin nettside.

Trykknapper

Vi brukte enkle knapper som var kompatible med breadboard. Disse knappene la en GPIO-pinne til jord når den ble trykket inn.

Breadboard

For å kople sammen knapper, RFID-leser og Raspberry Pi var det gunstig å bruke et breadboard. På denne måten slapp vi å lodde ting sammen, og vi kunne enkelt gjøre endringer på systemet dersom det skulle være nødvendig.

Ledninger

Vi brukte såkalte *jumper wires* som er mye brukt i hobbyelektronikk for å sammenkoble komponenter. Disse kommer med termineringer i endene som passer i breadboard, og dette gjorde at oppsettet vårt var relativt lett å sette sammen og å gjøre endringer på.

3.2.2 Klientdel

For å få det nettbaserte systemet vårt til å virke måtte vi ha de grunnleggende tingene på plass. Vi hadde bestemt oss for å bruke en Raspberry Pi 4 model B som plattform, og måtte bestemme oss for hvilket operativsystem vi skulle bruke. Vi måtte også bestemme oss for hvilken nettleser vi ville at systemet skulle benytte seg av. Systemet vårt bruker en kombinasjon av trykknapper og en touch-skjerm for å innhente informasjon fra brukere. Vi måtte derfor også finne en metode for å overføre informasjon fra trykknappene til nettleser.

Det første vi gjorde var å sette opp en Raspberry Pi med touch-skjermen som vi hadde bestemt oss for å bruke i prototypen til klientplattformen. Dette gjorde vi ved å bruke de medfølgende skruene til å feste Raspberry Pi-en til baksiden av skjermen. Deretter brukte vi flatkabelen til å koble sammen de to komponentene. Vi måtte også bruke de medfølgende ledningene for å forsyne skjermen med en spenning fra Raspberry Pi-en. På grunn av skjermens størrelse trenger den mer strøm enn flatkabelen kan forsyne.

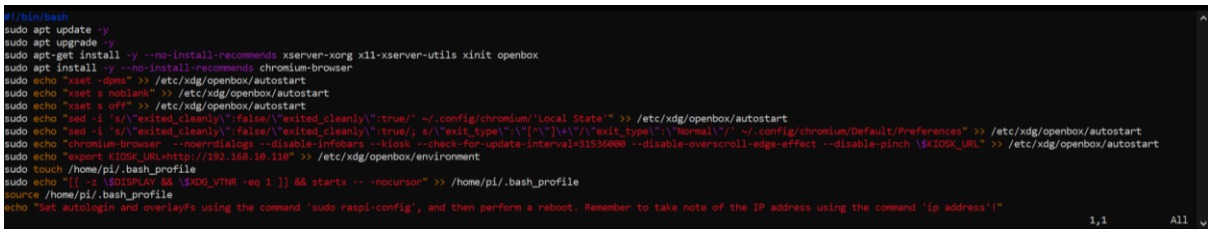
Vi lastet deretter over operativsystemet Raspberry Pi OS Lite på Raspberry Pi-en. Dette ble gjort ved å bruke *Raspberry Pi Imager*, et program som kan laste operativsystemer og IMG-filer over på et MicroSD-kort. Vi brukte en MicroSD-kortleser som kan kobles til en USB-inngang på en datamaskin. Deretter ble kortet satt inn i Raspberry Pi-en for å sjekke at alt virket som det skulle.

For at systemet skulle starte opp med en nettleser når det ble skrudd på måtte vi endre litt på hvordan operativsystemets brukergrensesnitt virket. Først og fremst måtte vi installere et grafisk brukergrensesnitt på systemet slik at vi har mulighet til å starte en grafisk nettleser. Dette ble gjort ved å installere *X Server*. I tillegg til dette, installerte vi en såkalt *window manager* som heter *Openbox*. Dette ga oss muligheten til å starte opp nettleseren og vise nettsider på plattformen vår. Vi måtte også installere selve nettleseren, Chromium Browser.

Vi ville at nettleseren skulle starte automatisk når Openbox starter. Dette fikk vi til ved å gå inn i diverse konfigurasjonsfiler og gjøre endringer i disse. Det første vi gjorde, var å gå inn i konfigurasjonsfilene til Openbox. Her skrudde vi av strømsparingsalternativer som kan føre til at skjermen slår seg av, eller at en skjermsparer starter etter en satt tid. I tillegg skrudde vi av pop-up meldinger som kan komme opp ved oppstart av nettleseren. Til slutt satte vi inn en kommando i denne filen som gjør at nettleseren starter opp med en bestemt nettside når Openbox startes.

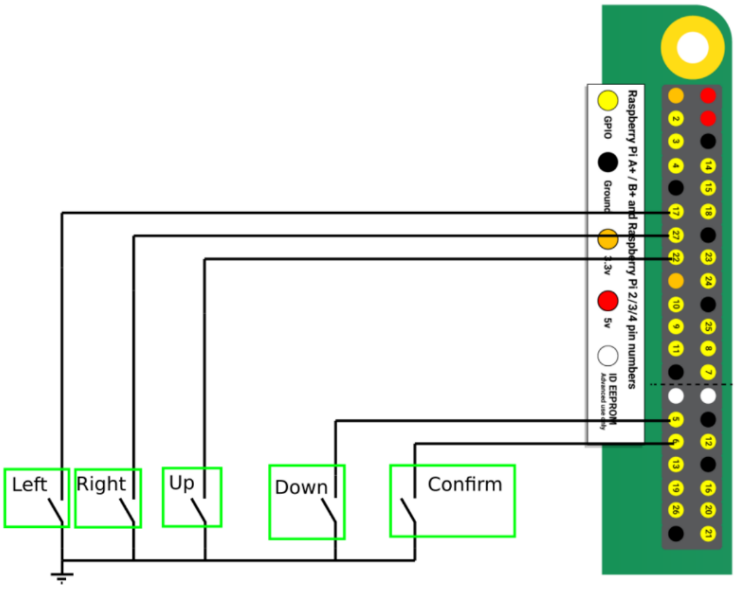
Når en bruker logger seg på operativsystemet kan vi få forskjellige ting til å skje automatisk ved å legge inn kommandoer i en fil som heter *bash_profile*. Her la vi inn en kommando som automatisk starter X-Server, og dermed Openbox og nettleseren. Vi endret deretter på en innstilling i operativsystemet som gjorde at brukeren logget seg inn automatisk ved oppstart. Dermed har vi en klientplattform som kunne starte en nettleser og åpne en gitt nettside ved oppstart av maskinen.

Å sette opp klientplattformen bestod hovedsakelig av å skrive inn kommandoer i BASH-terminalen. Vi bestemte oss for å lage et BASH-skript slik at denne prosessen kunne gjøres både raskere og sikrere. BASH-skriptet som ble brukt for å lage prototypen er vist i figur 3.1.



Figur 3.1 BASH-skript som skript for installasjon og oppsett av grafisk brukergrensesnitt.

Klientplattformen sitt brukergrensesnitt skulle bestå av en touch-skjerm og knapper. For å få knappene til å virke brukte vi de innebygde GPIO-portene som Raspberry Pi-enheten har. Disse har innebygde *pull-up-motstander*. Grunnet dette koblet vi GPIO-pinnene som vi trengte til elektrisk jord via trykknapper. Vi brukte programmeringsspråket Python til å sjekke om trykknappene var trykket inn eller ikke. Deretter brukte vi en *modul* som heter *keyboard* som kan lastes ned for Python (se vedlegg S). Denne modulen gir oss muligheten til å simulere tastetrykk fra tastaturet. Dette brukte vi i kombinasjon med Javascript som kjørte i nettleseren til å identifisere hvilke knapper som ble trykket på. Vi kunne deretter bruke denne informasjonen i Javascript til å navigere nettsidene våre med.



Figur 3.2 Oppkoblingsskjema for knapper

For at dette Python-skriptet skulle virke var vi avhengige av at den startet sammen med klientplattformen, og at den kjørte i bakgrunnen uten at brukeren behøver å tenke over det. For å få til dette brukte vi SystemD til å starte programmet. Vi laget en *service-fil* som inneholdt informasjon om hvordan skriptet skulle kjøres. Vi hadde nå en *daemon* som ble startet samtidig som systemet. Vi laget også et BASH-skript for å installere denne daemon-en på klientdelen på en rask og sikker måte.

Etter at operativsystemet og daemon-en var satt opp ordentlig tok vi en kopi av minnekortet. Kopien er i form av en IMG-fil, og kan brukes til å installere køsystemet direkte til et minnekort. Dette gjør at vi slipper å kjøre installasjonsskriptene for hver installasjon. På denne måten kunne vi raskt lage tre andre kopier av minnekortet, og dermed hadde vi tre klienter som var klare til bruk. Det er en liten detalj som må gjøres før systemet kan kjøres, men dette vil vi presentere i resultatdelen av kapittelet.

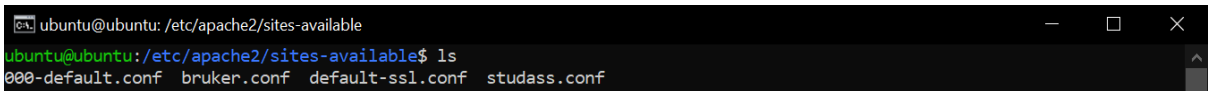
3.2.3 Serverdel

Serverplattformen som vi brukte i prototypen bestod av en Raspberry Pi 4 model B som kjørte en webserver, en DNS-server og en DHCP-server. Den var tilkoblet samme nettverket som klientplattformene. Serveren hadde som hovedoppgave å levere nettsidene som skulle vises på disse. I tillegg har den en del andre oppgaver. En annen oppgave er å fungere som en autorativ domenenavnserver. Dette betyr at den gir ut informasjon om IP-adresser og DNS-server til klientene som er koblet på nettverket.

Vi brukte webserveren Apache2 for å levere nettsidene våre til klientene. For å installere denne webserveren brukte vi tekstkommandoer i terminalen til operativsystemet. Man kan installere denne webserveren og nødvendige moduler ved å bruke følgende kommandoer i en terminal i en Debian-basert distribusjon av GNU/Linux:

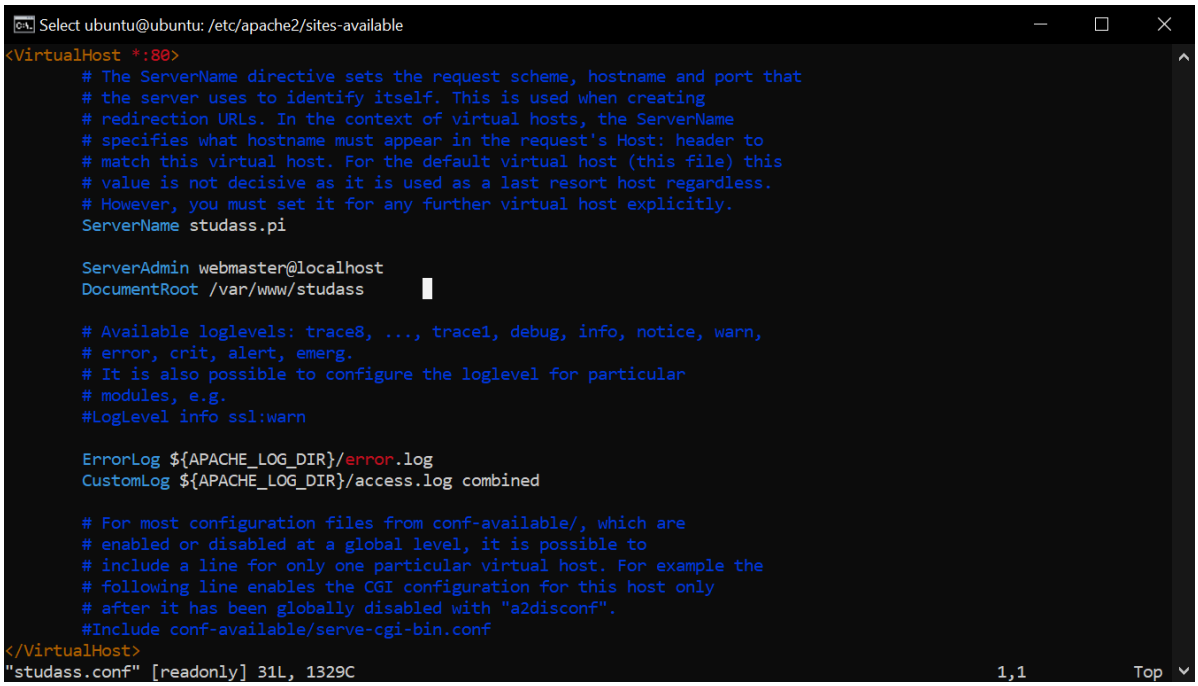
```
sudo apt install apache2 libapache2-mod-php php-mysql php -y
```

Etter at webserveren var installert måtte vi sette opp såkalte *virtual hosts*. Dette gjorde at vi kunne kjøre flere nettsider fra den samme serveren samtidig. For å få til dette, gikk vi inn i konfigurasjonsfilene til apache2, og la til to nye konfigurasjonsfiler for nettsidene som vi skulle bruke. Her spesifiserte vi hvilket domenenavn som tilhørte hver av nettsidene, og hvor HTML-koden til disse var på datamaskinen. Bildet under viser de to konfigurasjonsfilene som vi la til. Disse er «bruker.conf» og «studass.conf».



Figur 3.3 Konfigurasjonsfiler for Apache2.

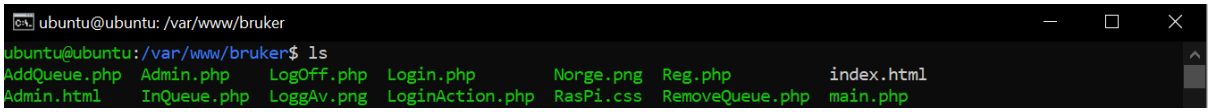
Oppsettet av hver av disse konfigurasjonsfilene vises i bildene under. Her er de to viktigste linjene å merke seg «ServerName bruker.pi» og «DocumentRoot /var/www/bruker». Den første linjen av de to, forteller oss at navnet på nettsiden, og det som skal skrives inn i URL-feltet, er «bruker.pi». Det andre feltet forteller oss at filene som webserveren skal servere ligger under «/var/www/bruker» på datamaskinen.



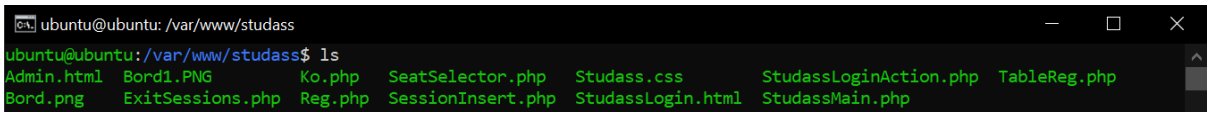
Figur 3.4 Innholdet av en av konfigurasjonsfilene for Apache2

Hver av de to sidene får sin egen lagringsplass for å oppbevare HTML-koden og det som har med utseende av selve sidene å gjøre. De tre bildene under, viser disse lagringsplassene, og innholdet i hver av dem. Dette er HTML-koden som blir overført til nettleseren når en bruker går inn på nettsidene våre.

Figur 3.5 De forskjellige nettsidene som blir levert av Apache2 på vår server



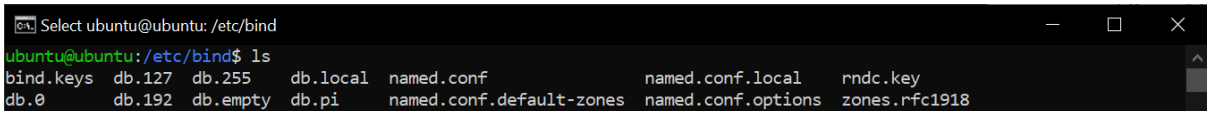
Figur 3.6 Innholdet som ligger under nettsiden for studenter.



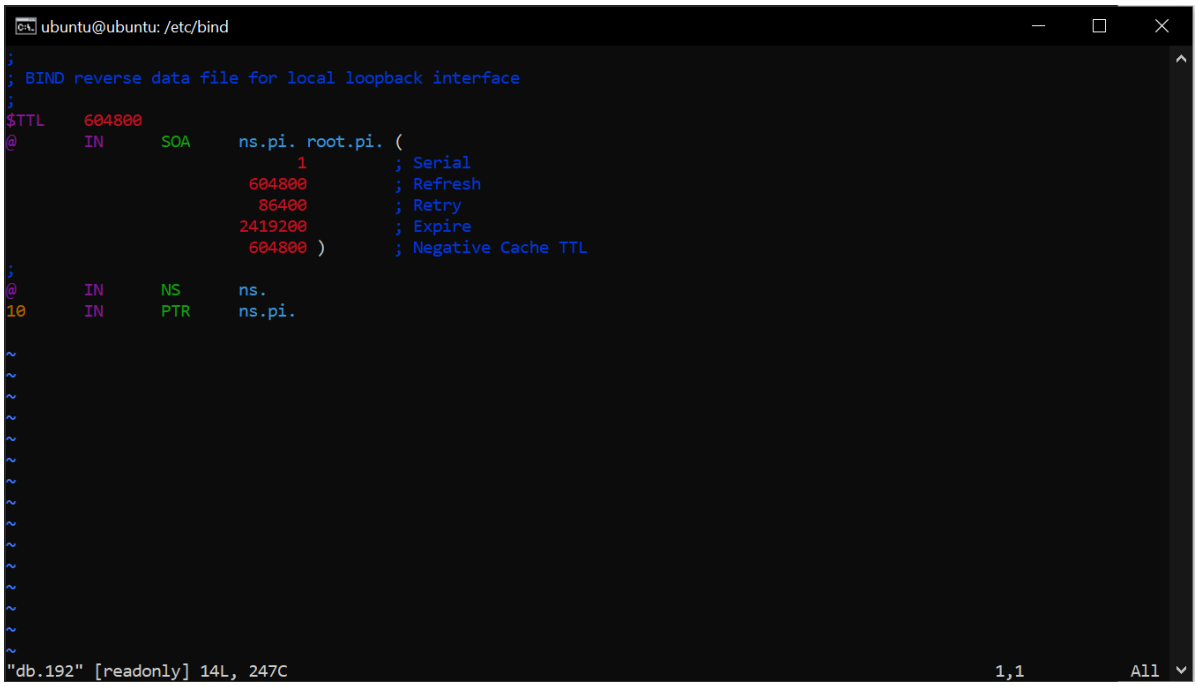
Figur 3.7 Innholdet som ligger under nettsiden for studentassistenter

Vi brukte en DNS-server som heter BIND9. Vi installerte denne ved å bruke terminalen i operativsystemet. For å konfigurere denne DNS-serveren som en autorativ DNS-server, måtte vi inn i konfigurasjonsfilene til BIND9. Vi ville lage domenet *pi*, og det gjorde vi i filen som ligger under «/etc/bind/db.pi». Vi måtte også sette opp et revers-domene. Dette ble gjort under

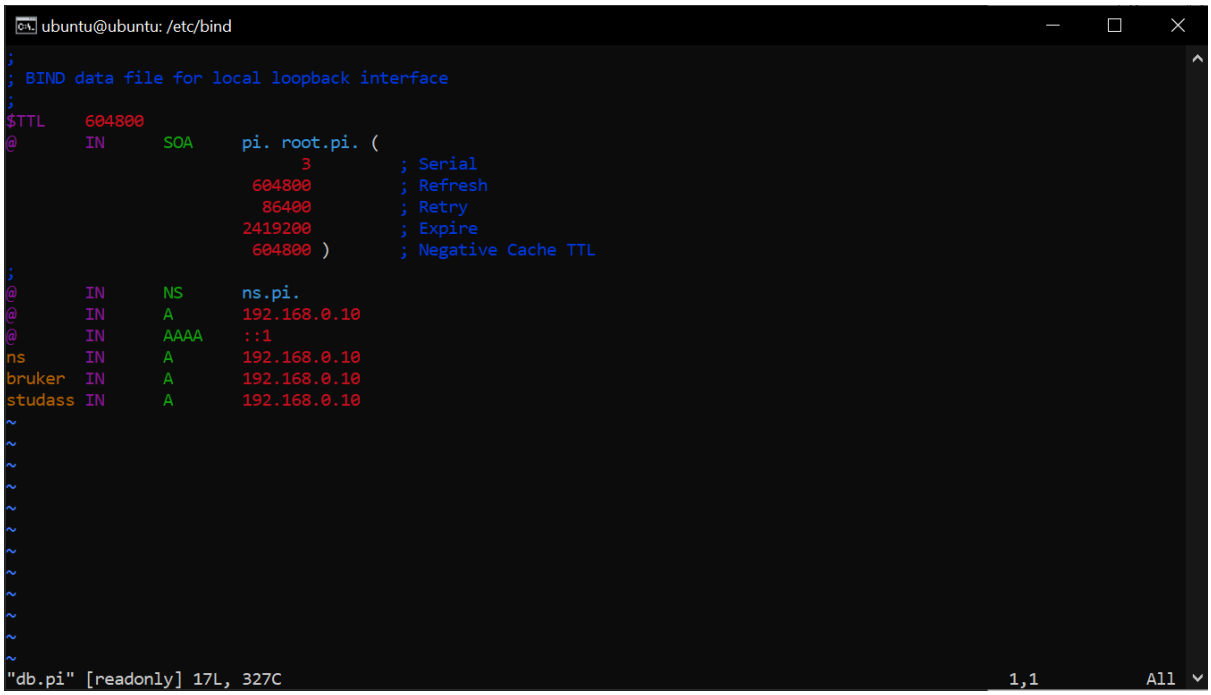
«/etc/bind/db.192». Konfigurasjonsfilene, samt innholdet til de to viktigste filene er vist i figur 3.8, 3.9, 3.10 og 3.11.



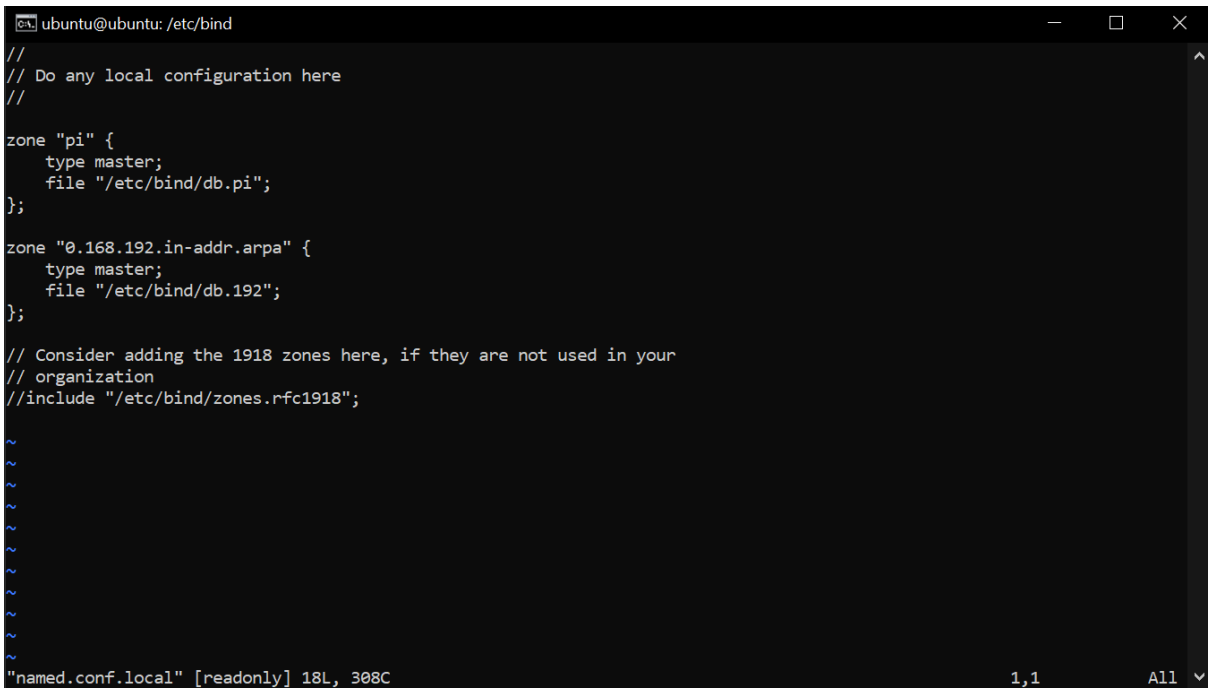
Figur 3.8 Konfigurasjonsfiler for DNS-serveren BIND9



Figur 3.9 Innholdet i konfigurasjonsfilen «db.192»



Figur 3.10 Innholdet av filen «db.pi»



Figur 3.11 Innholdet av filen «named.conf.local»

For at klientene skulle kommunisere over nettverket, måtte de ha IP-adresser som stemte overens. Dette ordnet vi ved å installere en DHCP-server på serverplattformen. Vi satte opp konfigurasjonsfilen «/etc/dhcp/dhclient.conf» som vist i figur 3.12. *Range* er start- og

sluttadresse for de adressene som DHCP-serveren deler ut. *Option domain-name-servers* angir adressen til DNS-serveren som vil brukes.

```
# This is a very basic subnet declaration.

subnet 192.168.0.0 netmask 255.255.255.0 {
  range 192.168.0.11 192.168.0.254;
  option routers 192.168.0.1;
  option domain-name-servers 192.168.0.10;
}
```

Figur 3.12 Innstillinger for DHCP-serveren ISC DHCP

3.3 Resultat

Prototypeplattformen består av fire Raspberry Pi-enheter med hver sin touch-skjerm, og en server. I tillegg har hver Raspberry Pi et eget sett med trykknapper som er ment som et alternativ for navigasjon av systemet dersom man foretrekker dette foran touch-skjermen. Hver av Raspberry Pi-enhetene er utstyrt med operativsystemet Raspberry Pi OS Lite, og starter automatisk opp med nettleseren i fullskjerm. De har også daemon-er for både trykknapper og RFID-leser ferdig installert, og disse starter også automatisk under oppstarten. To svitsjer kopler sammen kommunikasjonen mellom hver av Raspberry Pi-enhetene og serveren via ethernet. Serveren i prototypen kjører tre forskjellige tjenester for at systemet skal virke. Disse tjenestene er en webserver, en DNS-server og en DHCP-server. Webserveren sørger for at nettsider som blir forespurt av klientene blir levert. DNS-serveren sørger for at vi kan utnytte webserverens evne til å skille mellom flere nettsider ved hjelp av virtual hosts. I tillegg gjør DNS-serveren det mulig å bruke domenenavn i stedet for IP-adresser for å nå nettsidene våre. DHCP-serveren passer på at hver av de tilkoblede enhetene kan kommunisere sammen ved å administrere IP-adresser. Videre gir DHCP-serveren ut informasjon om hvilken IP-adresse som DNS-serveren er tilgjengelig på. Dette gjør at vi slipper å manuelt konfigurere nettverksinnstillingene til nye enheter som skal kobles til nettverket.

Dersom systemet skal tas i bruk slik det er i dag, må man installere operativsystemet med de tilhørende daemon-er. Dette kan gjøres relativt enkelt ved å kopiere en IMG-fil som inneholder disse nødvendige komponentene over på et MicroSD-kort. Et alternativ til dette, er å installere de nødvendige konfigurasjoner og daemon-er på en ny installasjon av Raspberry Pi OS Lite ved å kjøre tre separate skript. Det første skriptet er for å installere nettleseren, GUI og diverse nødvendigheter for å få systemet til å starte opp slik som vi ønsker. De to andre

skriptene er for å installere daemon-er for input av informasjon fra RFID og trykknapper. Detaljert informasjon om installasjon er gitt i medfølgende README-fil. Denne filen er ment som en bruksanvisning for hvordan systemet skal installeres.

Når operativsystemet og de to daemon-ene er satt opp ordentlig, skal systemet starte opp med å vise IP-adressen sin. På denne måten kan en systemadministrator enkelt koble seg til enheten ved hjelp av SSH, og gjøre endringer. Endringer som bør gjøres er også beskrevet i README-filen. De går ut på å sette nettsideadressen som man ønsker å vise på enheten. I tillegg bør man sette systemet i read-only-modus for å unngå unødvendig korrupsjon på minnekortet ved strømbrudd. Når systemet er satt i drift skal man kunne styre systemet både via touch-skjerm og knapper. Man skal kunne logge seg inn via en RFID-leser, og man skal kunne legge seg til i kø via nettsidene som er lagt til i webserveren.

3.4 Drøfting

3.4.1 Valg av operativsystem

Under utviklingsprosessen av klientdelen til plattformen måtte vi bestemme oss for hvilket operativsystem vi ville bruke. Operativsystemet skulle kjøres på en Raspberry Pi 4 model B. Operativsystemets oppgave var å holde styr på daemon-ene som vi skulle lage, kjøre en nettleser i fullskjerm, og håndtere nettverkskommunikasjonen for oss.

Vi laget en liste over operativsystemer som vi vurderte. Blant disse alternativene var Raspberry Pi OS, Raspberry Pi OS Lite, og Linutop. Hver av disse operativsystemene hadde sine fordeler og ulemper, og vi måtte velge det vi mente var best tilpasset vår situasjon.

Raspberry Pi OS er et operativsystem som er basert på Linux-distribusjonen Debian. Det er ment å kjøres på Raspberry Pi, og har mange nyttige applikasjoner ferdig installert. I tillegg kommer dette operativsystemet med et grafisk brukergrensesnitt. Dette betydde for oss at vi kunne slippe å installere og sette brukergrensesnittet opp manuelt. En av bakdelene med å bruke dette operativsystemet derimot, er at mange av de programmene som kommer ferdig installert er unødvendige. Ettersom at Raspberry Pi-enheten er en relativt kraftig datamaskin i forhold til rollen den har i vårt system, ville dette sannsynligvis ikke ha noe betydning for systemet. Likevel, dersom en systemadministrator bestemmer seg for å bruke andre, billigere alternativ til denne datamaskinen, kan det ha en innvirkning på ytelsen.

Raspberry Pi OS Lite er en versjon av Raspberry Pi OS hvor man har fjernet unødvendige programmer og GUI fra operativsystemet for å gjøre det lettere å kjøre for datamaskinen. Dette operativsystemet egner seg bedre til maskinvare som har mindre prosesseringskraft enn det Raspberry Pi OS gjør. Bakdelen med dette operativsystemet er at dersom vi skulle bruke en nettleser, måtte vi installere applikasjoner for å gi et grafisk brukergrensesnitt til dette formålet.

Et annet alternativ som vi vurderte, var et operativsystem som heter Linutop OS. Dette operativsystemet er beregnet for bruk i små datamaskiner som står i kiosker og slikt. Operativsystemet kommer med ferdig installert programvare som er beregnet for dette. Den største fordelen med dette operativsystemet er at man kan få support fra Linutop. Bakdelen er derimot at man må betale for fullversjonen av operativsystemet. I tillegg ville vi ikke ha bruk for all programvare som kommer ferdig installert.

Vi bestemte oss til slutt for at Raspberry Pi OS Lite var det beste alternativet for oss. Dette var fordi det er relativt lett å kjøre, og for at vi ikke trenger å avinstallere unødvendige programmer som følger med installasjonen av operativsystemet. Vi hadde også funnet ut at å installere et grafisk brukergrensesnitt på dette systemet ikke var så vanskelig som det vi hadde trodd. I tillegg har dette operativsystemet en innebygget funksjon som setter filsystemet i read-only-modus. Dette kan gjøre at korrupsjon av minnekortet ikke skjer like ofte.

3.4.2 Valg av kommunikasjonsmodell

Vi ville at systemet skulle være enkelt å gjøre endringer på for en systemadministrator. Derfor måtte vi tenke over hvor filene skulle være lagret. Et alternativ var å ha alle filene lokalt på hver klientmaskin, og ha selve køen i en database på en separat server. Dette ville ha redusert belastningen på nettverket, siden man ikke hadde trengt å overføre nettsider og skript fra en webserver. Derimot ville endringer i systemet blitt en svært tidkrevende prosess. Det er fordi at dersom man skal gjøre en endring på nettsiden må filene på hver enkelt klientmaskin endres manuelt. Dette kan være en grei måte å gjøre ting på dersom man har få antall klientmaskiner, men det er ikke skalerbart til et større antall klientmaskiner.

Vi bestemte oss for å gå for en serverbasert løsning. Dette ga en større belastning på nettverket og var en litt mer kompleks løsning, men det gjorde ting lettere for administratorer. Dersom en systemadministrator ønsket å gjøre en løsning, kunne de enkelt og greit logge seg inn på maskinen som webserveren kjørte på og endre filene én gang. Siden klientene hentet

nettsidene fra serveren, ville endringene som ble gjort her, påvirke hele systemet. Det å bruke en server gjorde systemet mye mer fleksibelt på bekostning av en liten økning i kompleksiteten, rett og slett fordi å sette opp en webserver er mer komplekst enn å bruke lokale filer.

3.4.3 Identifikasjon av klientplattform

Et problem som måtte løses, var hvordan systemet skulle kunne identifisere hver av klientmaskinene med hvilken arbeidsplass de var på. Dette ble gjort slik at studentassistenter kunne identifisere studenter i kø. Denne funksjonen var etter vår mening viktig for at systemet skulle fungere ordentlig. Vi hadde to alternativer for å få dette til. Begge metodene plasserer bordnummeret i nettsideadressen. Forskjellen ligger i metode for innhenting av bordnummer.

Den første metoden var å endre denne koden manuelt ved installasjon av køsystemet. I teorien skal man kun behøve å gjøre dette én gang per enhet. I praksis vil minnekortene til klientmaskinene bli korrupt over tid, og man vil ende opp med å måtte sette opp nye identifikatorkoder. Dette kan fort bli arbeidsintensivt for en systemadministrator.

Det vi kunne ha gjort dersom dette systemet skulle igangsettes i en større skala enn en prototype, er å fikse dette ved å implementere en form for hardware-identifikasjon. Vi kunne for eksempel ha brukt en mekanisk tallskive som gir forskjellige elektriske signaler basert på hvilke tallverdier den er innstilt på. Disse verdiene kunne leses av klientmaskinen og brukes til å bestemme bordnummeret som den sender til serveren. Ved denne metoden slipper systemadministratoren å manuelt endre på filene i systemet hver gang en klientmaskin skal settes opp. Grunnen til at vi ikke har gjort dette i vår prototype, er at dette problemet ikke ble innsett før i sluttfasen av prosjektet.

4 Adgangskontroll

Dette kapittelet omhandler delen av prosjektet knyttet til utfordringen for hver student å logge seg inn med adgangskortet sitt. Kapittelet tar for seg selve studentkortet, kommunikasjon samt relevant teori, utstyr og framgangsmåte. Kapittelet presenterer så et resultat, og drøfter hvorvidt resultatet er nådd i forhold til mål.

4.1 Teori

4.1.1 Studentkort

Adgangskort kommer i mange ulike varianter, og kan derfor være vanskelig å identifisere hvilke funksjoner og komponenter de består av. Tidligere student hos NTNU, Dag Erik Vikan utførte i 2013 flere tester for å få bedre innsikt i hvordan adgangskortene har fungert tidligere [68]. Denne informasjonen har gitt oss et godt grunnlag for å forstå hvordan studentkortet fungerer per i dag. Studentkortet er levert gjennom Nexus ID Solutions AS, og er av produkttypen «BK2050». Studentkortet er utstyrt med passive radiobrikker som opererer i forskjellige frekvensområder. Det er av typen EM4200, Mifare Classic 1K, samt en HiCo magnetstripe [45].

Figur 4.1 Studentkort pr. i dag

EM4200

EM4200 er en CMOS integrert krets, benyttet for bruk av *Read Only* RF transponder [19]. Denne har effektivt område mellom 100 til 150 kHz, som defineres som Low Frequency (LF). I tillegg finner vi en unik 128 bit kode lagret i laser-programmert ROM (Read-only memory). LF-radiobrikker kan operere gjennom tynne metallflater, samt i miljø med høy fuktighet. Den har en liten effektiv rekkevidde, som gjør den ideell for utveksling av sensitiv informasjon. Lagring er begrenset og kan medføre høyere produksjonskostnader.

Mifare Classic 1K

Mifare Classic 1K er en 1024 bytes lagringsenhet med evnen til å lese og lagre data basert på brukerdefinerte forhold. Radiobrikken opererer på frekvensen 13.56 MHz og defineres som High Frequency (HF) [47]. HF-radiobrikker fungerer på metallobjekt, har høyere effektiv rekkevidde og tilbyr anti-kollisjonsevner som gir muligheten for avlesning av flere brikker samtidig.

HiCo Magnetstripe

HiCo (High Coercivity) er typiske svartfargede magnetstriper kodet med et sterkt magnetisk felt [22]. Kortet vil dermed ikke bli utsatt for tilfeldige tap av data dersom den blir eksponert til andre magnetiske felt.

4.1.2 Kommunikasjonsprotokoll

Under oppsettet skal RFID-leseren kobles til vår Raspberry Pi. Ved informasjonsveksling mellom flere digitale enheter er det nødvending å fastlegge riktig sett av regler, også kalt kommunikasjonsprotokoller [28]. Slik kan en sikre at enheter kommuniserer riktig med hverandre. Her gjør vi rede for SPI, I^2C og USART som er serielle kommunikasjonsmetoder støttet av Raspberry Pi.

SPI

Serial Peripheral Interface er (SPI) er en synkron seriell kommunikasjonsmetode over fire eller flere ledere, gitt antall slaves vist i tabell 4.1. Konfigurasjonen opereres i full dupleks og tilbyr en dataoverføringshastighet på 8Mbits eller mer. Overførsel av informasjon foregår først ved at masteren velger et klokkesignal kompatibel med slaveenheten over SCLK. Slik synkroniseres enhetene. Deretter benyttes SS for å velge ønskede kommunikasjonsslave, og

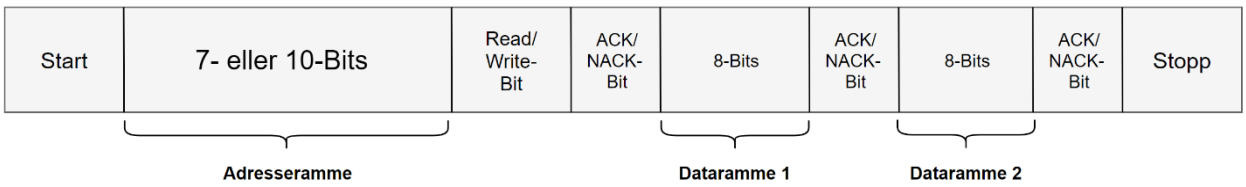
setter denne lav hos slaven. Data blir henholdsvis overført på MOSI til slaven, og MISO fra slave til masteren [71]. SPI tilbyr den raskeste dataoverføringsraten uten avbrudd, men er avhengig av flere porter, låst til en master og tilbyr ingen feilsjekking.

SLCK	Serial Clock
MOSI	Master Out Slave In
MISO	Master In Slave Out
SS	Slave Select

Tabell 4.1 Ledere i SPI

I^2C

Inter-Integrated Circuit (I^2C) er en synkron seriell kommunikasjonsprotokoll over to ledere, *SDA* og *SCL* gitt tabell 4.2. Protokollen er halv-dupleks, og støtter oppkobling av flere slaver til enkel master, samt flere master til en eller flere slaver. Informasjon sendes i form av meldinger, brutt ned til ulike datarammer. Rammene består av en 7- eller 10-bits adressering for slaven, 8-bits datarammer samt instruksjonsrammer, vist i figur 4.2. Kommunikasjon startes ved at master setter SDA og SCL til lav, ettersom de har normalverdi høy. Adresseringsrammen sendes til ønsket slave, samt instruksjonssett. Kommunikasjonen settes i gang etterfulgt av dette [8]. I^2C er en velkjent protokoll som kun utnytter to porter og tilbyr feilsjekking gjennom ACK/NACK (Acknowledge/No-Acknowledge). Datahastigheten er tregere enn SPI, og er begrenset til 8-bit.



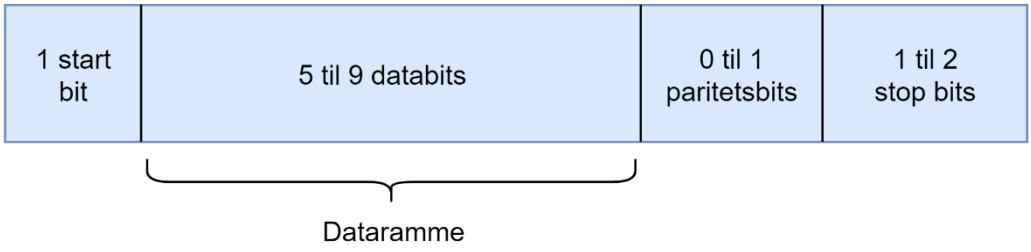
Figur 4.2 I2C Adresseblokk

SDA	Serial Data
SCL	Serial Clock

Tabell 4.2 Ledere i I^2C

UART

Universal Asynchronous Receiver/Transmitter (UART) er en asynkron seriell kommunikasjonsprotokoll. Dette er en fysisk krets i en mikrokontroller eller en ekstern modul som består av to tilkoblinger for å overføre og motta informasjon gitt tabell 4.3. UART kan opereres som simpleks, halv-dupleks og dupleks. Data er overført som pakker, med start- og stop-bits samt paritetsbits vist i figur 4.3. Slik vil mottakeren få instruksjoner når avlesningen skal skje og endringer har forekommet. Overføringshastigheten er basert på baud raten, som må være omtrentlig lik i sender og mottaker. Data overføres parallelt fra bussen til UART-sender. Pakken sendes bit for bit på Tx, og leses bit for bit av på Rx. UART-mottaker samler så datainformasjonen og fjerner informasjonsbits før den parallelt blir overført til mottakerbuss. UART benytter kun to porter, er simpel å bruke og har feilsjekking gjennom paritetsbits. Den er låst til kun en slave-master, begrenset til 9-bits datarammer og er avhengig av baudraten [9].



Figur 4.3 UART datapakke

Tx	Transmit
Rx	Receive

Tabell 4.3 Ledere i UART

4.2 Metode

I problemstillingen er det gitt at gruppen skal implementere en adgangskontroll hvorav studentene logger på arbeidsplass med utstedt kort. Følgende kort gir muligheten til flere metoder for innlogging, enten gjennom magnetstripen eller radiobrikkene. For enkelhets skyld vil vi avgrense oss til behandlingen av radiobrikkene, ettersom dette er mer intuitivt og vil ikke påføre særlig slitasje på kortet etter gjentatt bruk. Funksjon, brukervennlighet og produksjonskostnader er tatt til hensyn. Basert på dette, og ønske fra oppdragsgiver velger vi å utnytte oss av Mifare Classic 1K. Målet blir å få avlesning av dette kortet, samt utskrift av den unike identifikatoren.

4.2.1 Utstyr

VMA405

En av de mest utbredte RFID-enhetene tilgjengelig er RFID RC522. De er basert på MFRC522 fra NXP referert i 2.2.1. Enhetene er rimelige og effektive samt har et bredt programvarebibiliotek vi kan utnytte oss av. I tillegg har de lese- og skriveevner. VMA405 fra Velleman er RFID-leseren vi nytter oss av. Denne er tilnærmet lik MFRC522 [46][67]. Vist i tabell 4.4, har denne en driftsspenning på 3.3 VDC, effektiv rekkevidde på 5 cm samt støtte for SPI.

Driftsspenning	3.3 VDC
Arbeidsstrøm	13-26 mA
Frekvensområde	13.56 MHz
Grensesnitt	SPI
Effektiv rekkevidde	5 cm
Overføringshastighet	Maks 10 Mbit/s

Tabell 4.4 VMA405 Spesifikasjoner [67]

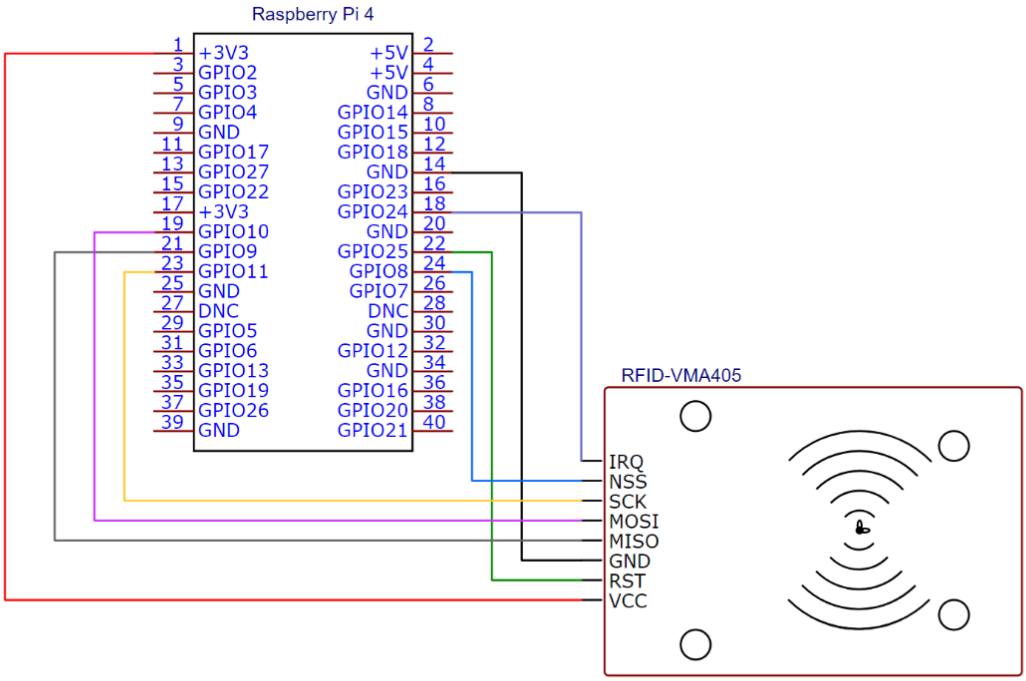
I tabell 4.5 ser vi at denne er utstyrt med NSS (No Slave Select), som kun har støtte for SPI-protokollen. Da vil en ikke få utnyttet UART og I^2C, som normalt kan justeres gjennom valg av grensesnitt på mikrokontrolleren. Vi har nok av ledige porter, og nytter oss derfor av protokollen som gir oss den raskeste hastigheten samt god flyt i systemet. Dette passer godt til modulens høye krav for dataoverføring. Til testing benyttet vi oss av enkle kort og brikke utstyrt med Mifare Classic 1k.

VCC	Strømforsyner
RST	Reset/Av strøm
GND	Jordingspunkt
MISO	Master In Slave Out
MOSI	Master Out Slave In
SCK	Serial Clock
NSS	Active-low slave select
IRQ	Interrupt pin

Tabell 4.5 VMA405 Pinout [67]

4.2.2 Konfigurasjon

Oppkobling av RFID-leseren til prototypen er gjort i henhold til tabell 2.3, vist i figur 4.4. Her er modulen direkte koblet opp til Raspberry Pi med enkle kabler. Oppsettet tar hensyn til riktige kobling av GPIO-pinner gitt grensesnitt, samt oppkobling av knapper.



Figur 4.4 Kretsskjema av RFID og Raspberry Pi

For at modulen skal kunne kommunisere med vår datamaskin er det nødvendig å aktiver SPI-protokollen. Dette gjøres gjennom terminalen på datamaskinen.

```
sudo raspi-config
```

Denne kommandoen vil åpne konfigurasjonsmenyen for enheten, hvorav en kan aktivere SPI-protokollen. For å sikre at modulen fungerer må en oppdatere maskinen.

```
sudo apt-get update
sudo apt-get upgrade
```

Vi ender med å installere nødvendige bibliotek samt verktøy for informasjonsvekslingen. Bibliotekene *spidev* og *mfrc522* forenkler interaksjonen med SPI- og RFID- grensesnittet gjennom en python-modul [13][72]. Slik kan en enkelt begynne å kjøre skript for interaksjoner med leser.

```
sudo apt-get install python3-dev python3-pip
sudo pip3 install spidev
sudo pip3 install mfrc522
```

Denne prosessen er automatisert gjennom bruken av daemon på datamaskinene, referert i kapittel 3.2.2.

4.2.3 Programmering

I henhold til satte mål ble koden *read.py* vist i figur 4.5 opprettet. Koden er en modifisert variant basert på testkoden presentert i mfrc522-modulen [72]. Fra 1-2 har vi importert bibliotekene *RPI.GPIO* samt *keyboard.* Førstnevnte modul tilbyr bedre kontroll av GPIO-pinnene på vår Raspberry Pi, mens *keyboard* vil bidra med simulering av tastetrykk. Slik kan vi enklere få utskrift av identifikatoren. I tillegg importeres fra 4-5 *SimpleMFRC522* for håndtering av RFID og *sleep* fra *time*. *Sleep* vil bidra til å unngå dobbelutskrift av ID. Linje 8 benyttes for å deaktivere feilmeldinger som skrives ut til konsollen. Modulen *SimpleMFRC522* blir satt i variabelen *reader*. Fra 11 er det gitt en boolsk variabel *rfid_Y* for regulering av *while*-løkken. Denne ble ikke benyttet i den endelige koden, men er beholdt for eventuelle fremtidige implementeringer. Så lenge den boolske variabelen er satt til True, vil løkken utføres. Her blir *id* samt *text* avlest fra modulen. Med *keyboard* simulerer vi utskriften av innholdet og spesifiserer *ctrl* og *enter* til dette. Vi trenger kun identifikatoren *id*, som skrives ut som en 12-sifret integrer. Vi omformer denne til en string for bruk i PHP og benytter

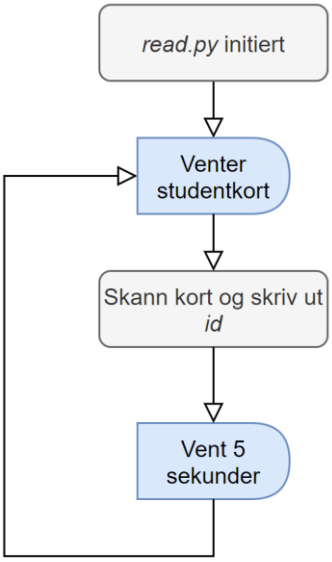
så *sleep* med 5 sekunders intervall. Fra 24 håndterer vi error-blokken. *GPIO.cleanup()* sikrer at portene resettes tilbake til input-modus. Dette avslutter koden trygt, og hindrer eventuelle kortslutninger som kan oppstå.

```
1.  import RPi.GPIO as GPIO
2.  import keyboard
3.
4.  from mfrc522 import SimpleMFRC522
5.  from time import sleep
6.
7.
8.  GPIO.setwarnings(False)
9.  reader = SimpleMFRC522()
10.
11. rfid_Y = True;
12.
13. while rfid_Y:
14.       try:
15.
16.
17.           id, text = reader.read()
18.           keyboard.send('ctrl')
19.           keyboard.write(str(id))
20.           keyboard.send('enter')
21.
22.           sleep(5)
23.
24.       except:
25.             GPIO.cleanup()
```

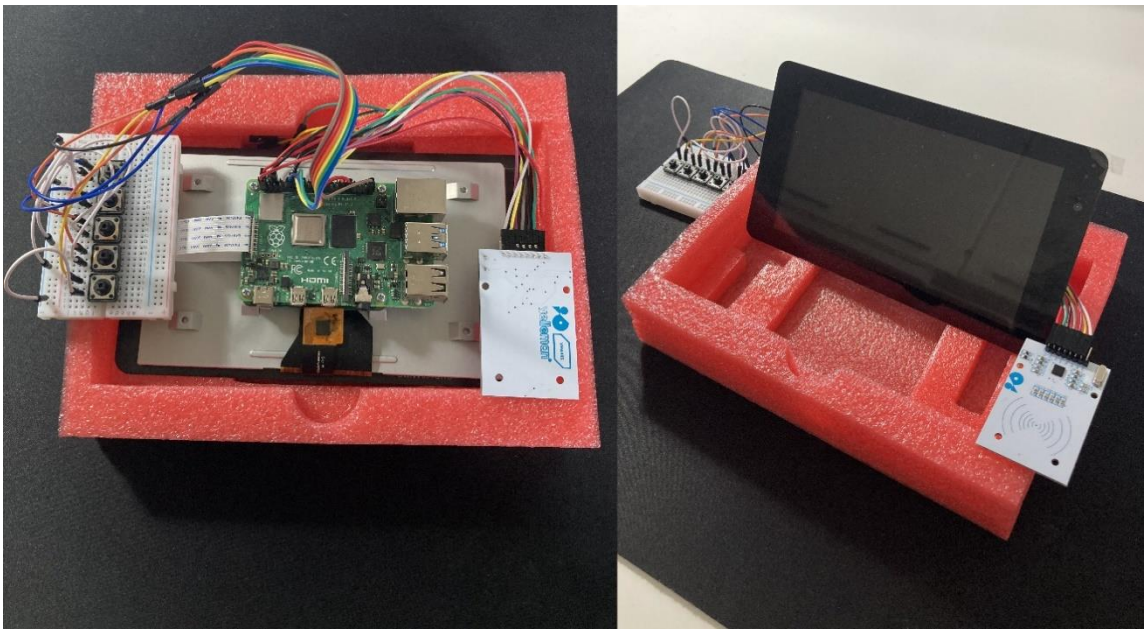
Figur 4.5 Avlesning av RFID-leser «read.py»

4.3 Resultat

RFID-modulen er oppkoblet til systemet sammen med knappene referert i 3.2.2. Oppkoblingen er vist i figur 4.7 med et forenklet handlingsdiagram av koden på 4.6. Gjennom oppsettet av RFID-modulen resulterer vi med en løsning hvor studentkortet blir avlest når den er innenfor RFID-leserens rekkevidde. Her får vi en utskrift av et unikt 12-sifret kode i string-format som skal i utgangspunktet benyttes videre for innlogging og registrering av brukere i systemet. Grunnet valg av bibliotek, benyttes ikke IRQ, og er derfor ikke koblet inn i prototypen. I design for ferdigstilt PCB skal den kobles til for framtidig bruk. Koden kjøres kontinuerlig til programmet er avsluttet.



Figur 4.6 Handlingsdiagram read.py



Figur 4.7 Oppsett av moduler

4.4 Drøfting

Innkjøp av RFID-leseren har vært noe utfordrende grunnet manglende leverandører. Derfor har gruppen kun benyttet seg av en modul utstedt av NTNU. Dette har virket litt begrensende under testing av flere system, men ikke påvirket utviklingsfasen. I henhold til satte mål gir modulen resultatene tiltenkt, men noe erfaring ble gjort underveis. Det ble tidlig i prosessen bestemt å bruke ferdigstilte bibliotek for kommunikasjon med modulen, istedenfor å produsere vårt eget. Argumentet ville være en mer helhetlig forståelse av kodestrukturen dersom en utvikler den selv. Gruppen er derimot meget sikker på valget gjort, ettersom utbyttet ikke ville være gunstig i forhold til gitt tidsplan. Vi forholdt oss derfor til biblioteket *mfrc522*. For å kunne oppdage studentkortet krever denne konstante kommander, som låser en til while-løkker. Slik vil en skape et CPU-forbruk mellom 80-100%, som er ikke levedyktig for systemet. En løsning ville vært å implementere en *time.sleep()* på while-løkken i *SimpleMFRC522* [72]. Ved å plassere et tidsavbrudd på 100-300ms mellom avlesninger har andre opplevd nedsatt forbruk på 10-40%. Dette kunne en eventuelt innført dersom en utviklet eget bibliotek. Det mest ideelle for systemet ville være å få utnyttet IRQ på leseren. Dette vil åpne for støtte av event listener, som vil begrense forbruket ytterligere. Henholdsvis til denne utfordringen har gruppen funnet biblioteket *pi-rc522* [48]. Den utnytter IRQ, og tilbyr funksjonen *wait_for_tag()* som avventer lesing av kort til den er registrert på modulen. Denne vil være enkel å implementere i systemet ved fremtidige iterasjoner.

Ettersom vi benytter oss av Mifare Classic 1k tilbyr dette også lagring av data på kortene. Dette ligger utenfor gruppens mål, men åpner flere muligheter for systemet i fremtiden. Det er ikke innført noen sikkerhetsmessige funksjoner ved registrering og innlogging av bruker med kortet. Systemet er derfor avhengig av troverdigheten til studenter og studass.

5 Nettside og kommunikasjon

Dette kapittelet omhandler prosess og resultat for sammensetting av det server-baserte køsystemet med database og nettside. Kapittelet vil ta for seg alle aspekter fra databasestruktur til designvalg av grafisk framstilling.

5.1 Teori

5.1.1 MySQL

MySQL er et Open-Source Database Management System, støttet av Oracle-gruppen. MySQL er bygd slik at det er mulig å bruke samme databaseoppsett på tilnærmet alle programvare- og maskinvarekonfigurasjoner for en server. MySQL-systemet er skrevet i C++, og bygger på en databasestruktur med bruk av tabeller, rader og kolonner for å skape et oversiktlig rammeverk. Disse tabellene kan ha tilhørighet til hverandre samt gir deg mulighet til å bygge «pekere» som kan sortere data og endre dynamisk på informasjon. SQL står for Structured Query Language, og er en av de mest brukte språkene for å kommunisere med databaser. SQL lar deg bygge systemer hvor kommunikasjon med databasen kan bygges inn blant annen kode og bli kjørt av din valgte serverprogramvare. Endring i verdier på et MySQL-databasesystem gjøres via enkle kommandoer som «Insert», «Select», «Update» og «Delete». Her kan man legge til ekstra informasjon som peker kommandoen til riktige områder av databasenes tabeller [43].

5.1.2 PHP

PHP er et Open-Source server-side scripting-Language. Det er spesielt godt egnet for bruk i hosting av nettsider og er et språk brukt veldig mye i startfasen av internett (1995-2000). Programmeringsspråket fases nå ut i moderne design på grunn av fleksibilitetmangel i koding [50]. PHP er bygd med mulighet for enkle, kraftige kommandoer med dype verktøymuligheter for avansert skripting. Språket er velegnet til bruk med MySQL, hvor du kan koble til og kontrollere databasen ved bruk av de standardiserte kommandoene. PHP kan brukes på alle store operativsystemer og støtter store deler av server-hosting programvare. Alt som må til for at PHP skal kunne bli brukt er at en PHP-tolkningsmodul er installert på serveren. På grunn av denne fleksibiliteten har maskinvarevalg og serverprogramvare veldig lite å si i henhold til design av web-løsninger med PHP som kjerne [49].

5.1.3 JavaScript

Javascript er et kraftig skripting-language som kjøres lokalt i en valgt instans, som regel en nettleser. JavaScript er et såkalt «Just-In-Time» språk. Dette betyr at koden kan bli utført til enhver tid i motsetning til PHP, som kun kjøres før nettsiden vises. Javascript lar deg endre hvordan nettleseren viser fram elementer, og har interne variabler som gjør at man kan skrive programmer som kjører i bakgrunnen. JavaScript har syntaks veldig lik C og Java, men har egentlig ingenting ellers til felles med dem. JavaScript har støtte for komplekse objektorienterte programmeringsmetoder samt funksjonsbasert programmering. Dette gjør at JavaScript er ekstremt kraftig som programmeringsspråk for bruk i nettlesere. JavaScript har ingen Input eller Output, og språket gjenkjenner kun det som er lagt til i tjenesten som kjører språket. Med tanke på nettsider betyr dette at alt som ikke er lagt til i nettleseren kan ikke JavaScript endre eller lese [38].

5.1.4 HTML / CSS

HTML står for «HyperText Markup Language» og er grunnstrukturen for framvisning av informasjon på internett. HTML er ikke et standard programmeringsspråk men heller et tekstbasert rammeverk for strukturering av tekst og informasjon på nettsider. Dette gjøres ved hjelp av tags som definerer hvordan tekst og bilder skal framvises. HTML har innebygde definisjoner for hvordan forskjellige tags skal se ut, men man kan også definere egne regler ved hjelp av CSS. CSS står for «Cascading Style Sheets» og er en måte å endre på framvisningen av HTML. Ved hjelp av CSS og HTML kan man forandre på alle aspekter ved framvisning av informasjon på en nettside [24].

5.2 Design og Implementasjon

5.2.1 Utstyr og programvare

Oracle VirtualBox

Operativsystemet som er installert på serveren ved elektrolaboratoriet er basert på Linux. Dermed var det viktig å utvikle database og nettsidehosting slik at det fungerer på operativsystemet. Den mest praktiske metoden for å skape en Linuxbasert testplattform var ved hjelp av en Virtual Machine (VM). Oracle Virtualbox ble brukt for å lage en VM med Ubuntu som operativsystem. Ubuntu ble valgt fordi det er en av de mest populære distribusjonene av Linux. Da kunne vi forsikre oss om at alle funksjoner vi ønsket ville fungere i en fullstendig prototype. Oracle VirtualBox lar deg virtuelt simulere programvare ved å skape et avlukket filsystem på datamaskinen.

Apache 2.0

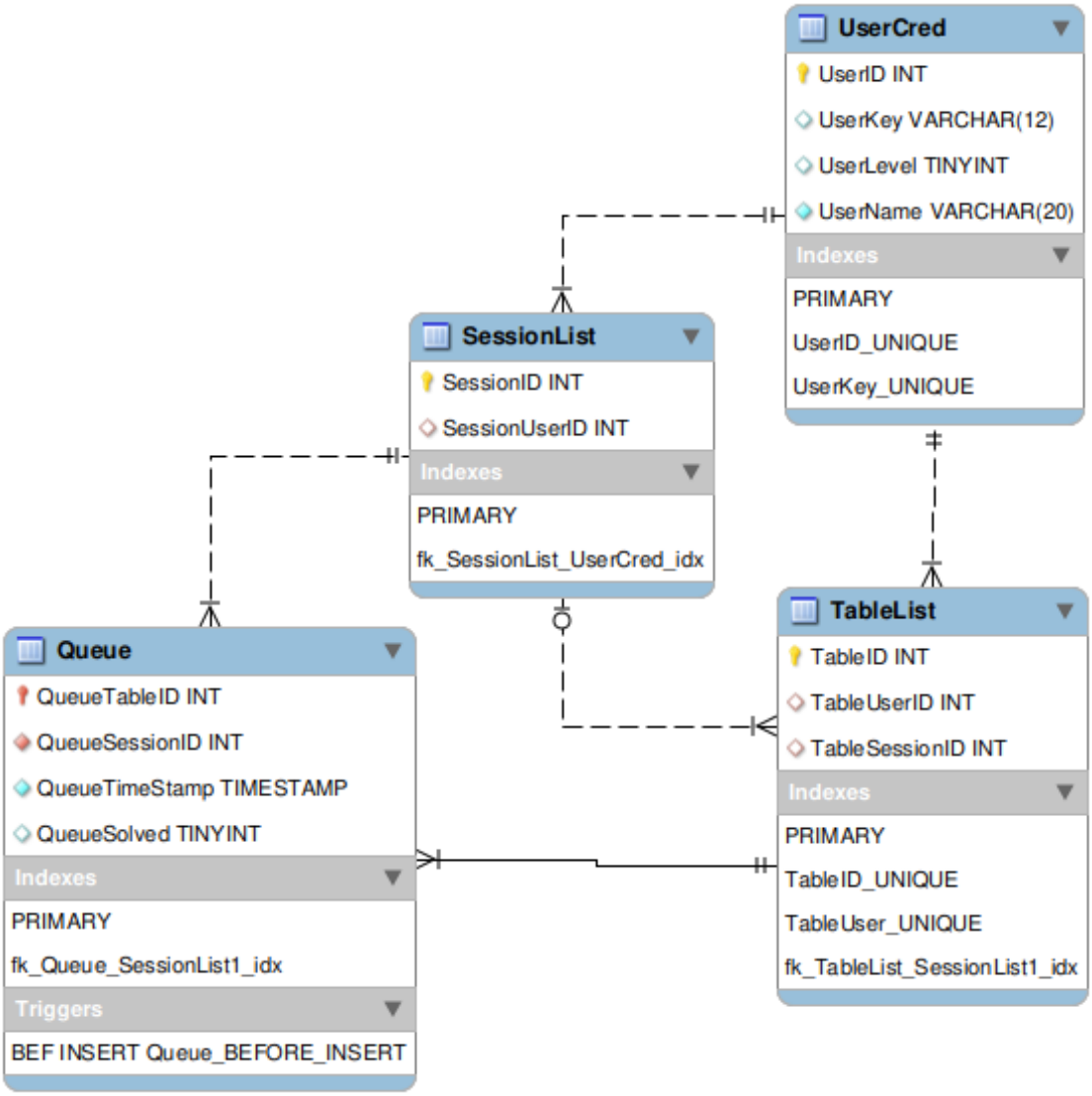
For å kunne designe nettsider med PHP er det nødvendig å sette opp en server for nettsidene. For dette brukte vi Apache 2.0 med PHP- og MySQL-modul installert. Apache ble valgt på grunn av at det er mye brukt som lokal server-hosting programvarene for Linuxbaserte maskiner. Ved hjelp av Apache og moduler kan man sette opp et nettsidehierarki som kan kommunisere med en valgt database via PHP.

MySQL

For håndtering av data inn- og ut av nettsiden valgte vi å bruke MySQL. Dette på grunn av tidligere erfaring. MySQL er gunstig for dette systemet på grunn av lav kompleksitet. Databasen ble satt opp med 4 forskjellige tabeller. Tabellene skal håndtere all informasjon som er nødvendig for administrering av innlogging på bordene, informasjon om brukerne og informasjon om den aktive køen.

5.2.2 Oppsett av Database

For å sikre at all data som brukes i systemet er sortert på en oversiktlig måte er databasestrukturen bygd med flere tabeller som har tilkobling til hverandre. Tabellene inneholder dermed informasjon som kan refereres til ved hjelp av et identifikasjonsnummer. Dette nummeret vil være unikt for hver tabell og lar oss enkelt referere mellom tabellene uten unødvendig kopiering av store mengder data. Figur 5.1 viser et ER-diagram som illustrerer databasestrukturen.



Figur 5.1 ER-diagram

Den viktigste tabellen for systemet er UserCred. Her er all informasjon om brukere som er registrert i systemet lagret. I denne tabellen ligger det navn for brukeren, hvilke tilganger brukeren har samt passordet som kreves for innlogging.

Tilgangene er sortert etter tall fra 0-3. Her tilsvarer 0 ingen tilganger, 1 student, 2 studass og 3 administrator. Denne inndelingen gjør at man kan bygge forskjellige funksjoner inn i systemet med felles innloggingspunkt. Man kan også verifisere tilganger for endringer.

Tablelist inneholder listen over alle bordene i systemet. Denne har som hovedansvar å holde orden på hvem som er logget inn i systemet og hvilken studass som har ansvar for bordet. Når man logger inn i systemet vil brukerens identifiserende nummer bli lagt til i denne tabellen.

Sessionlist er en liste over aktive studass-sessions og hvilke brukere som har ansvar for de forskjellige sessionene. Ved å fordele bordene i sessions kan man også skille mellom fagområder.

Hvis brukere som er logget inn på et aktivt bord velger å legge seg til i køen, vil denne informasjonen bli lagt til i Queue-tabellen. Her har man informasjon om hvilket bord som trenger hjelp, samt hvem som har ansvar for å hjelpe. I tillegg ligger det informasjon om når brukeren spurte om hjelp. Dette er brukt for å sortere køen slik at de som har ventet lengst får hjelp først.

For automatisering av Queue-tabellen brukes følgende MySQL Trigger vist i figur 5.2.

```
1.  CREATE DEFINER=`Andy`@`localhost` TRIGGER `Queue_BEFORE_INSERT` BEFORE INSERT ON `Queue`
    FOR EACH ROW BEGIN
2.  DECLARE SessionID INT;
3.  SELECT
4.     TableSessionID
5.  INTO SessionID FROM
6.     TableList
7.  WHERE
8.     TableID = NEW.QueueTableID;
9.  SET NEW.QueueSessionID = SessionID;
10. END
```

Figur 5.2 MySQL Trigger

Triggeren aktiveres før «Insert» gjøres på tabellen. Den finner da SessionID som tilhører det bordet som ønskes satt i kø. Deretter oppdaterer den SessionID-verdien til tilhørende verdi. Dette gjør at man kan minske informasjon som må sendes via PHP-skript.

5.2.3 Student

Dette delkapittelet vil gi en summering av metodene som er brukt for design av nettsidene for studenter og hvordan de kommuniserer med databasen. I tillegg vil kapittelet gå over hvordan JavaScript er brukt for å skape interaktivitet og samle inn informasjon fra fysiske knapper og RFID.

CSS

For design av siden var det viktig å designe rundt de fysiske begrensingene skjermene setter for nettleseren. Skjermen systemet er designet for har en oppløsning på 800x480 piksler med en størrelse på 4.3 tommer i diameter. Det må også være designet for bruk av touch som input.

Etter disse begrensingene var det viktig å skape et design som var enkelt å bruke og var lettforståelig. Dermed brukte vi CSS Grid for å skape egendefinerte soner for interaktivitet på siden. CSS Grid er en relativt ny metode for sortering av elementer i design av nettsider. Det ble først støttet av alle store nettlesere i 2017 og er på vei inn som standard for nye nettsider [39]. Ved hjelp av Grid kan du ved veldig enkel syntaks lage et underliggende tabellsystem hvor man kan legge elementer i bokser med definerte posisjoner på siden.

```
1.  body{
2.      display: grid;
3.      height: 460px;
4.      width: 780px;
5.      grid-template-rows: 1fr 4fr;
6.      grid-template-areas: "Header" "Interact";
7.
8.  }
```

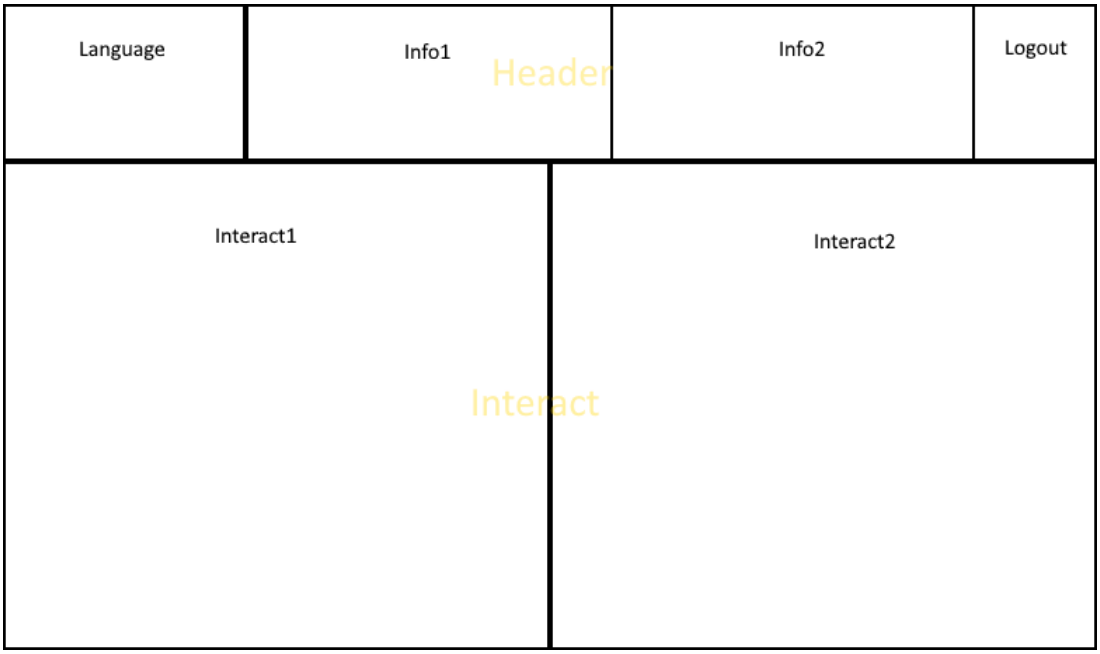
Figur 5.3 CSS Grid syntaks

Denne koden endrer på hvordan <body> taggen i HTML er satt opp. Den begrenser høyde og bredde slik at alle interaktive elementer vil holde seg innenfor dimensjonene til skjermen. Høyden og bredden av de aktive elementene er mindre enn skjermens oppløsning grunnet padding som automatisk legges til i nettlesere. Ved testing førte en oppløsning på 800x480 piksler til at nettleseren ikke viste alle elementer samtidig. Dermed er det satt av ca. 20 piksler i hver retning som en buffer. Det legges også til grid-elementet i linje 2. Senere endres rad-elementene med en fordeling på $\frac{1}{5}$ og $\frac{4}{5}$ (1fr 4fr) i linje 5. Disse radene får navn «Header» og «Interact», og blir utdypet i figur 5.4.

```
1.  .Header{
2.      grid-area: "Header";
3.      display: grid;
4.      grid-template-columns: 2fr 3fr 3fr 1fr;
5.      grid-template-areas: "Language info1 info2 Logout";
6.  }
7.
8.  .Interact{
9.      grid-area: "Interact";
10.     display: grid;
11.     grid-template-columns: 1fr 1fr;
12.     grid-template-areas: "Interact1 Interact2"
13. }
```

Figur 5.4 CSS Grid utdyping

«.Header» er bygd opp med 4 soner for elementer. Disse sonene er navngitt og plassfordelt på linje 4 og 5. «.Interact» er bygd opp med 2 soner for elementer. Disse sonene er navngitt og plassfordelt på linje 11-12. Ved hjelp av denne CSS-koden vil man få en inndeling av nettsiden hvor man senere kan legge til informasjon innenfor de satte områdene. Figur 5.5 viser en illustrasjon om hvordan dette systemet ser ut. Linjene vil ikke være synlige for en bruker, men nettleseren vil bruke disse for å sortere innholdet på siden.



Figur 5.5 CSS Grid visualisering

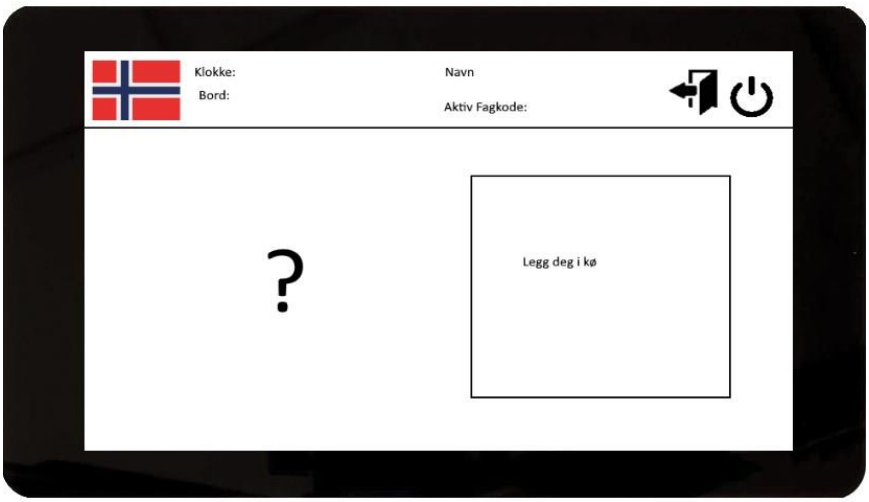
Nettleseren vi bruker har noen uheldige innebygde funksjoner for å bla fram og tilbake mellom sider ved hjelp av touch på skjerm. Dette er noe som ikke var ønskelig for produktet fordi det ville skape forstyrrelser for innloggingsscript og databaseinformasjon. Dette problemet har derimot en veldig enkel løsning.

```
html,body {overscroll-behavior: none;}
```

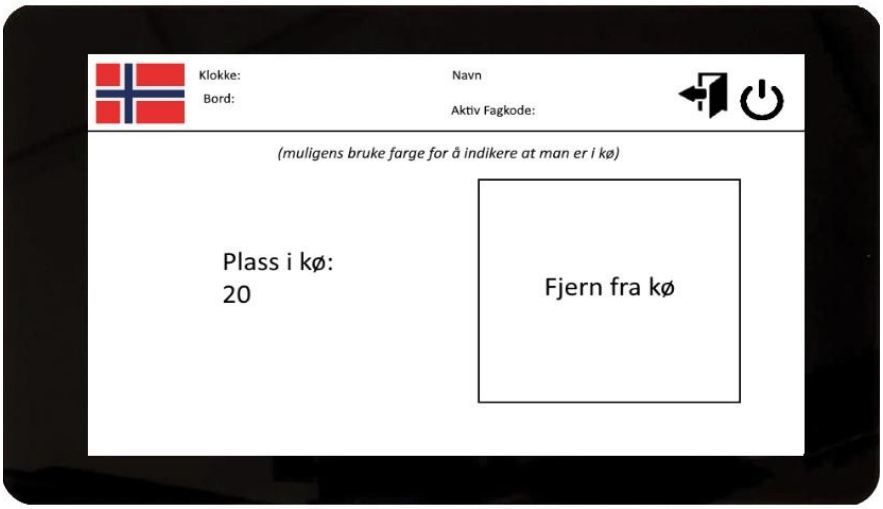
Ved bruk av 3 fingre aktiverer man vanligvis «overscroll». Normalt ville dette ha endret på siden man var på, som tilbakeknappen på en nettleser. Dette er uønsket og funksjonaliteten fjernes ved hjelp av CSS-koden ovenfor.

HTML

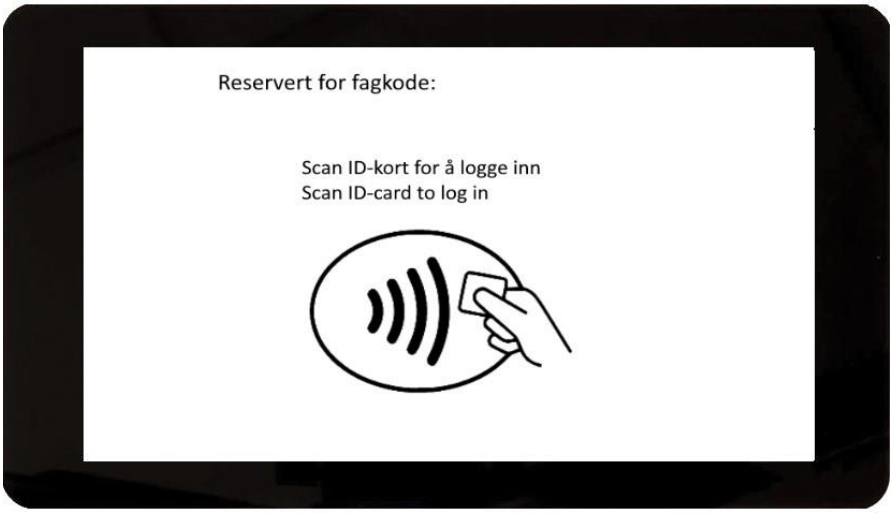
HTML-koden setter inn elementene i posisjonene som er definert av CSS-koden. Det er 3 konseptbilder i prototypen for bruker, vist henholdsvis i figur 5.6, 5.7 og 5.8.



Figur 5.6 Konseptbilde main



Figur 5.7 Konseptbilde inQueue



Figur 5.8 Konseptbilde Login

Kommentering av HTML-kode vil følge CSS Grid-illustrasjonen i figur 5.5. Alle HTML-objekter som blir satt inn er pakket i <element>-tags med tilkobling til en CSS Grid posisjon. På denne måten vil alltid det som er lagt innenfor <element>-tags være posisjonert på riktig sted på siden. Innenfor «Header» CSS Grid-elementet vil HTML-koden være lik for alle sider med unntak av Login-skjermen.

«Header» CSS Grid-elementet vil da inneholde kode vist i Figur 5.9.

```
1.  <element class ="Header">
2.    <element class="Language">
3.      <img src="../Norge.png" height="120px">
4.    </element>
5.      <element class="info1">
6.                  Time: <div width = "40px"id="clock"></div> <br>
7.                  BordNummer: <?php echo $Table; ?>
8.      </element>
9.      <element class="info2">
10.                 Navn: <?php echo $Name; ?>
11.     </element>
12.     <element class="Logout">
13.     <input type="image" src="../LoggAv.png" id="1" onclick="LogOut()" height="120px"/>
14.   </element>
15. </element>
```

Figur 5.9 «Header» HTML-kode

På linje 3 er det en lenke til et lokalt lagret bilde av et norsk flagg. Dette elementet var planlagt å være en knapp som skulle gi mulighet for endring av visningsspråk på sidene. På grunnlag av at NTNU er et internasjonalt universitet er det ikke gitt at alle kan norsk. Denne funksjonaliteten ble derimot nedprioritert og senere sløyfet på grunn av tidsmangel. Bildet er fortsatt med i designet grunnet mulighet for fremtidig implementasjon av et språkendringssystem.

Innen «Info1»-elementet er det en -tag som blir styrt av JavaScript-kode og viser en oppdatert klokke. I tillegg er det satt av plass for å vise bordnummeret bruker er på via PHP-utskrift. På «Info2»-elementet er det satt av plass for navnet til den innloggede brukeren. «Logout»-elementet har en <input>-tag som inneholder et bilde av en avknapp. Dette bildet er trykkbart og viser til JavaScript-kode som håndterer avlogging.

For main-siden vil HTML-koden være følgende vist i figur 5.10.

```
1.  <element class="Interact">
2.      <element class="Interact1"></element>
3.      <form method="GET" action="/AddQueue.php">
4.                  <input type="hidden" name="ID" value="<?php echo $Table ?>">
5.        <input class="Interact2" id="0" type="submit" value="Legg til i kø" />
6.      </form>
7.  </element>
```

Figur 5.10 «Interact» main HTML-kode

«Interact1» er tomt på grunnlag av mulig utvidelse av systemets funksjonalitet ved tilkobling til PLS-system. Innenfor «Interact2» er det et HTML-form-element som videresender deg til InQueue-siden og legger deg til i den aktive køen. For InQueue-siden vil HTML-koden være følgende vist i figur 5.11.

```
1.  <element class="Interact">
2.      <element class="Interact1">
3.                  <?php
4.                  $count = 0;
5.                  while($rad = mysqli_fetch_array($queueData)){
6.                          $QueueTableID = $rad["QueueTableID"];
7.                          if ($QueueTableID != $Table) { //skriven plassen din i kø. bruker index i tables så 0 --> 1
    i køen og dermed øke med 1
8.                                  $count++;
9.                          }else{
10.                                 echo "Din plass i køen er: ". ++$count .".";
11.                         };
12.                 }
13.                 ?>
14.     </element>
15.     <form method="GET"  action="/RemoveQueue.php">
16.                 <input type="hidden" name="ID" value="<?php echo $Table ?>">
17.       <input class="Interact2" id="0" type="submit" value="Trekk deg fra kø" />
18.     </form>
19. </element>
```

Figur 5.11 «Interact» InQueue HTML kode

Innen «Interact1»-elementet er det PHP kode som skriver ut din posisjon i køen. Dette blir identifisert ut ifra bordnummeret som ble sent med i nettsideadressen.

«Interact2»-elementet inneholder et HTML-form-element som sender deg tilbake til main-siden og trekker deg fra køen.

PHP

Samme tilkoblingsmetode er benyttet ved all PHP-kommunikasjon med databasen. Følgende kode er dermed med som første kodebit i alle relevante filer.

```
1.  <?php
2.      $tilkobling = mysqli_connect("localhost","Andy","Halla123","Backend");
3.
4.      if ($tilkobling -> connect_errno) {
5.                  echo "Failed to connect:" . $tilkobling -> connect_error;
6.      };
7.
8.      $Table = $_GET['ID'];Figur x.x – (tittel ikke oppgitt)
```

Figur 5.12 PHP-tilkoblingskode

På linje 2 lages en tilkobling til databasen «Backend» på server «Localhost» med brukernavn «Andy» og passord «Halla123». Deretter på linje 4 sjekkes tilkoblingen. Etterfulgt på linje 8 hentes bordnummeret ut av nettsideadressen. Hver enhet har en fast posisjon og dermed et fast bordnummer. Dette bordnummeret blir lagt til i nettsideadressen ved oppstart.

For å håndtere kommunikasjon med databasen er det bygd PHP skript-filer mellom alle HTML-sidene som sender og verifiserer data. PHP kode bakt inn i HTML-sidene er kun for avlesning av databaseverdier. Avlesning blir gjort av MySQL-kommandoen «SELECT». For alle valg av verdier er følgende PHP-struktur brukt i alle skript. Koden på linje 1 og 2 brukes for å hente dataen ut av databasen. Koden på linje 3 og 4 er brukt for å ekstrahere kun den informasjonen vi trenger fra datapakken.

```
1.  $user_query = "SELECT TableUserID FROM TableList WHERE TableID = '$Table'";
2.  $user_data = $tilkobling -> query($user_query);
3.  $user_data_array = mysqli_fetch_array($user_data);
4.  $UserID = $user_data_array["TableUserID"];
```

For endring av databaseverdier er PHP-kode brukt for verifisering av tilganger ved hjelp av IF-klausuler, og endring av verdier via MySQL Queries «INSERT» og «UPDATE».

```
$insert_query = "INSERT INTO Queue(QueueTableID) VALUES ('$Table')";
$reset_query = "UPDATE Queue SET QueueSolved = '0' WHERE QueueTableID = '$Table'";
```

For videreføring til neste side bruker man PHP-kommandoen header().

```
header("Location: /InQueue.php/?ID=". $Table ."");
```

JavaScript (Capture Keys/RFID/Klokke)

Den viktigste bruken av JavaScript-koden i systemet er for oppsamling og tolking av input fra RFID-leser samt eksterne knapper. JavaScript brukes også i klokkemodulen for å skape klikkbare knapper, og for å oppdatere nettsiden. Funksjonene er ikke nødvendige å forklare siden de omhandler enkeltlinjer av kode. Klokkefunksjonen er heller ikke viktig for systemets funksjon.

Knapper

Knappene på det fysiske systemet er koblet til GPIO og tolket av Python, referert i kapittel 3.2.2. Det som er relevant for forståelse av JavaScript-koden er å vite at ved trykk på en retningsknapp vil tilsvarende retning i piltastene på tastatur registreres. Knappen i midten vil tilsvare «enter»-knappen. Siden de simulerte trykkene er på et tastatur, kan dette samles opp av JavaScript-koden vist i figur 5.13.

```
1.  var loc = 0; // 0 = kø knapp 1 = logg av knapp
2.  document.getElementById('0').focus();
3.  window.addEventListener("keydown", function (event) {
4.    if (event.defaultPrevented) {
5.      return;
6.    }
7.      switch (event.key) {
8.        case "ArrowDown":
9.          document.getElementById('0').focus();
10.         loc = 0;
11.         break;
12.       case "ArrowUp":
13.         document.getElementById('1').focus();
14.         loc = 1;
15.         break;
16.       default:
17.         return;
18.     }
19.   event.preventDefault();
20. }, true);
```

Figur 5.13 Knappetrykk Input

Alle HTML-knapper som skal være mulig å trykke på er navngitt med ID korresponderende til kommentaren på linje 1. På linje 9 brukes *document.getElementById('0')* for å finne knappen som tilsvarer «Sett i kø», og velger å sette «focus» på den. Focus vil vise et sort omriss om knappen og lar deg se valgt knapp. Dette er gjort på grunn av at det første en bruker vil gjøre, er sannsynligvis å legge seg til i køen.

Fra linje 2-22 bygges en JavaScript event listener. Event listeneren aktiveres ved «Keydown»-event, og koden mellom linje 2 og 22 kjøres. På linje 4-6 har vi en exit for å passe på at ikke systemet reagerer på flere trykk av samme input. Dette er for å filtrere ut feil. Eksempelvis, hvis A er koblet til en funksjon vil AAAA bare utføre funksjonen en gang. Dette er på grunn av at på linje 21 setter vi *defaultPrevented = True*. På linje 7 har vi en *switch* for *event.key*. Denne vil aktiveres og sende igjennom verdien *Key*, dette kan brukes for å skape funksjoner. På linje 8-15 har vi to funksjoner som lar deg navigere på siden. På grunnlag at vi kun har to knapper er koden veldig enkel. Koden sjekker om du trykker opp eller ned og setter fokus på den tilhørende knappen. Ved bruk av denne koden kan man navigere på nettsiden uten bruk av touch-skjerm.

RFID

Input av RFID-informasjon på nettsiden var en viktig problemstilling for systemet. RFID-skanning ligger som bakgrunn for systemets funksjonalitet, og dermed var det en prioritering å få denne informasjonen inn på nettsiden. Vi løste dette med samme kodegrunnlag som input for knapper.

```
1.  <script type=”text/javascript”>
2.  window.addEventListener("keydown", function (event) {
3.    if (event.defaultPrevented) {
4.      return;
5.    }
6.      switch (event.key) {
7.        case "Control":
8.          document.getElementById('input').focus();
9.          break;
10.       default:
11.         return;
12.     }
13.
14.   event.preventDefault();
15. }, true);
```

Figur 5.14 RFID Input

Denne koden har samme funksjon som koden for tolking av knappetrykk, men her venter vi heller på at *Control*-knappen er trykket. Når dette er registrert skal man endre fokus til tekstboksen som heter «input». Etterfulgt kan man via simulerte tastetrykk skrive inn RFID-informasjonen direkte og sende dataen videre til et innloggingsskript ved å trykke «Enter»-knappen.

5.2.4 Studass

Dette kapittelet summerer metodene som er brukt for å designe et system for framvisning og administrering av køsystemet. Hovedsakelig er dette systemet bygd på PHP med interaktivitet gjennom JavaScript kode.

CSS

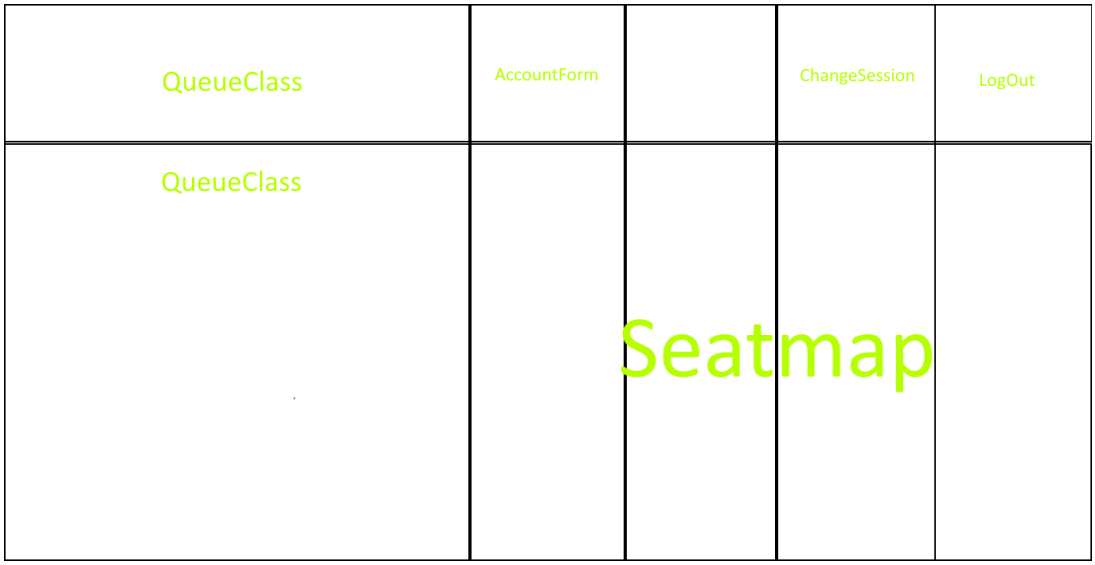
For design av nettsidene som studass skal bruke, er begrensningene på størrelse en mindre faktor. Dette grunnet forventninger om at nettsidene vil bli fremvist på en stasjonær datamaskin i elektrolaboratoriet. Dermed er nettsiden designet for å passe innenfor den mest gjennomsnittlige skjermstørrelsen, 1366x768 piksler.

Her er CSS Grid layout brukt. Kodebit i figur 5.15 setter opp definisjoner for designet av studass-nettsidene.

```
.mainBody{
display: grid;
                min-width: 1366px;
                min-height: 700px;
                grid-template-columns: 3fr 1fr 1fr 1fr 1fr;
                grid-template-rows: 1fr 3fr;
                border: black 1px solid;
}
```

Figur 5.15 Studass CSS Grid

Det er brukt *min-height* og *min-width* ettersom dette skal vises på skjermer som ikke nødvendigvis har fast størrelse. CSS grid vil endre på seg i forhold til størrelsen på nettleseren ved automatisk skalering.



Figur 5.16 Studass CSS Grid Visualisering

HTML

Konseptbildene for Studass-systemet er vist i figurene under.

Figur 5.17 StudassMain konseptbilde

Figur 5.18 TableReg konseptbilde

StudassMain

CSS Grid-området «QueueClass» vil bli fylt av en oppdatert liste over personer i køen som venter på hjelp. Strukturen for tabellen er vist i figur 5.19.

```
<table class='QueueClass'>
   <tr>
   <th>Kø SessionID</th>
   <th>Kø Nummer</th>
   <th>Bord Nummer</th>
   <th>Fjern</th>
  </tr>
```

Figur 5.19 «QueueClass» HTML-kode

Denne tabellen fylles via PHP som skriver ut informasjonen med HTML «tr»-tags (se figur 5.24).

I CSS Grid områdene ChangeSession og LogOut er det knapper. ChangeSession åpner opp et HTML-form som lar deg logge deg inn og lage/endre aktive sessions. Knappenes funksjon er styrt av enkel JavaScript kode. Figur 5.20 viser når knappen i ChangeSession registrerer trykk.

```
<div class="accountForm" id="accountVer">
<form method="post" action="/StudassLoginAction.php">
   <label><strong>Logg inn for å skape eller endre sessions</strong></label><br>
   <label>
   Brukernavn
   <input type="text" name="UserName">
  </label>
  <label>
   Passord
   <input type="password" name="UserPwd">
  </label>
  <label>
  <input type="submit" value="Logg Inn">
  </label>
</form>
<button onclick="goBack();">Lukk Login</button>
</div>
```

Figur 5.20 «accountForm» HTML-kode

Dette HTML-form-elementet sender login-informasjon videre og lar deg logge deg inn på «TableReg»-siden. Dette er nettsiden hvor du kan administrere bordenes tilganger til køen.

I *SeatMap* CSS Grid elementet er det et bilde av en oversikt over elektrolaboratoriet. Ettersom brukere registrert i køen har bord og bordnummer, gir dette studass bedre oversikt og et kart å navigere seg etter.

TableReg

HTML-koden for nettsiden TableReg er sammensatt av en enkel HTML form og et sett med HTML-knapper som er linket til et JavaScript-skript (se figur 5.29)

```
1.  <form method="POST" action="/SessionInsert.php">
2.    <div>fagkode?</div>
3.    <div>start/slutt?</div>
4.    <input type="hidden" name="sessionID" <?php echo "value = '". $sessionID ."'"; ?>>
5.    <input type="text" id="PlassValgBoks" name="PlassValg"><br>
6.    <input type="submit" value="Velg Seter" > <br>
7.  </form>
8.  <div class="Selector">
9.  <input type="button" class="Sel" id="37" value="37">
10. <input type="button" class="Sel" id="38" value="38">
11. _____
12. <input type="button" class="Sel" id="39" value="39">
13. <input type="button" class="Sel" id="40" value="40"><br><br>
14.
15. <input type="button" class="Sel" id="36" value="36">
16. <input type="button" class="Sel" id="35" value="35">
17. <input type="button" class="Sel" id="34" value="34">
18. <input type="button" class="Sel" id="33" value="33">
```

Figur 5.21 «TableReg» HTML-kode

Det er en knapp for hver plass i systemet. Dermed er det 88 knapper hardkodet i HTML. Figur 5.21 viser til 8 av dem. HTML form elementet inneholder «sessionID» som identifikator for personen som er logget inn samt valget av bord. Når man fullfører dette HTML-Formet vil man bli videresendt til StudassMain. Bordene er da registrert på økten din.

PHP

Slik som for studentsidene starter alle skript med PHP-kode med samme snitt som kobler til og gir beskjed ved tilkoblingsfeil.

```
1.  <?php
2.      $tilkobling = mysqli_connect("localhost","Andy","Halla123","Backend");
3.
4.      if ($tilkobling -> connect_errno) {
5.                  echo "Failed to connect:" . $tilkobling -> connect_error;
6.      };
```

Figur 5.22 PHP MySQL tilkobling

På hovedsiden er det PHP kode som sorterer og viser fram alle som trenger hjelp. Dette gjøres ved hjelp av kodesnitt vist i figur 5.23.

```
1.      $FjernID = $_GET['Fjern'];
2.      $FjernSessionID = $_GET['FjernSession'];
3.
4.      if(isset($FjernID)){ //Henter og setter slik at Hjelp på bordet er gitt.
5.          $fjernSql = "UPDATE Queue SET QueueSolved = '1' WHERE QueueTableID = '$FjernID' AND QueueSessionID
    = '$FjernSessionID'";
6.          $tilkobling->query($fjernSql);
7.          echo(mysqli_error($tilkobling));
8.      };
9.      $queueSql = "SELECT QueueSessionID, QueueTableID FROM Queue WHERE QueueSolved = '0' ORDER BY
    QueueSessionID ASC, QueueTimeStamp ASC"; //Henter alle bord som trenger hjelp og sorterer de med hensyn til
    Session og Tid
10.     $queueData = $tilkobling -> query($queueSql);
```

Figur 5.23 Innhenting av Kø data

På linje 1 og 2 henter vi variabler som fins i nettsideadressen hvis noen skal fjernes fra køen. På linje 4-7 sjekker vi om *$FjernID* er satt. Hvis den eksisterer skal vi endre på statusen til det bordet i køen som har relatert bordnummeret. Fra 9-10 henter vi alle i køen som trenger hjelp. Dette er identifisert ved «QueueSolved = ‘0’». Deretter sorterer vi resultatet etter QueueSessionID og QueueTimeStamp i ASC. På denne måten vil dataen som ligger i $QueueData-variabelen på linje 10 inneholde alle som er i køen sortert etter hvem som har ansvar gitt sessionID samt nummer i køen.

Videre bruker vi *echo* og en *while*-løkke for å vise køen gitt figur 5.24.

```
1.  <?php
2.      $QueueNumber = 1;
3.      $LastSessionID = "";
4.      while($rad = mysqli_fetch_array($queueData)){ //Skriver ut dataen som hentes ettehvert som den blir sendt. slipper å
    bruke loops
5.      $QueueSessionID = $rad["QueueSessionID"];
6.      $QueueTableID = $rad["QueueTableID"];
7.          echo "<tr>";
8.          echo "<td>";
9.          echo $QueueSessionID;
10.         echo "</td>";
11.         echo "<td>";
12.         if ($LastSessionID == $QueueSessionID OR $LastSessionID == ""){
13.             echo $QueueNumber;
14.             $QueueNumber++;
15.         }else{
16.         $QueueNumber = 1;
17.         echo $QueueNumber;
18.         $QueueNumber++;
19.       };
20.       $LastSessionID = $QueueSessionID;
21.         echo "</td>";
22.         echo "<td>";
23.         echo $QueueTableID;
24.         echo "</td>";
25.         echo "<td>";
26.         echo "<a href=/StudassMain.php/?Fjern=" . $QueueTableID . "&FjernSession=". $QueueSessionID .">Fjern</a>";
27.         echo "</td>";
28.         echo "</tr>";
29.     };
30.  ?>
```

Figur 5.24 Framvisning av kø data

På linje 2-3 opprettes variabler som benyttes for å kontrollere utskriften. Mellom linje 4 og 29 kjøres en *while*-løkke. Denne vil iterere så lenge «mysqli_fetch_array($QueueData)» er aktiv. Mysqli_fetch_array($QueueData)-funksjonen henter all informasjon innenfor $QueueData radvis. Innenfor løkken henter vi ut informasjonen vi trenger på linje 5-6. Siden listen er sortert etter SessionID først fulgt av tid, vil vi kunne klumpe det sammen og iterere nummer i køen manuelt. På linje 8-10 skriver vi ut SessionID-nummeret i riktig posisjon av raden ved hjelp av *echo* og HTML-tags. Fra 11-21 skriver vi ut nummeret som det tilhørende bordnummeret har i køen. Dette gjøres ved å iterere på variabelen $QueueNumber. Hvis SessionID-nummeret er identisk eller ikke satt, skrives $QueueNumber ut og økes med en. Ellers blir variabelen tilbakestilt. På denne måten vil hver sessionID ha sin egen separate nummerering av kø i samme tabell. På 20 oppdateres $LastSessionID-variabelen for å sammenligne i neste iterasjon av løkken. Fra linje 22-24 skriver vi ut bordnummeret til den som trenger hjelp. 25-27 skriver ut en lenke som fungerer som en knapp. Denne fjerner raden fra serien, slik at studass aktivt kan administrere hvem som er i køen.

TableReg

På bordregistreringssiden er det PHP-kode som henter inn informasjon om bordene. Slik kan brukeren kun velge bord som er ledige. PHP brukes og for å hente informasjonen om hvilken session som skal endres på.

```
1.  $tilkobling = mysqli_connect("localhost","Andy","Halla123","Backend");
2.
3.      if ($tilkobling -> connect_errno) {
4.                  echo "Failed to connect:" . $tilkobling -> connect_error;
5.      };
6.   $sessionID = $_GET[“ID”];
```

Figur 5.25 PHP-tilkobling database

```
1.  $SeatSQL = "SELECT TableID, TableSessionID FROM TableList WHERE TableSessionID IS NOT NULL"; //Henter
    listen over bord som allerede er med i en session
2.     $data = $tilkobling -> query($SeatSQL);
3.     echo "<script type='text/javascript'>";
4.     echo "var Vals = [];";
5.     while($rad = mysqli_fetch_array($data)){
6.     $TableID = $rad["TableID"];
7.     $tableSessionID = $rad["TableSessionID"];
8.     if ($tableSessionID != $sessionID) {
9.      echo "document.getElementById('".$TableID."').disabled = true;"; //Skrur av knappenen for bord som allerede er
    med i session
10.    }else{
11.     echo "Vals.push('". $TableID ."');";
12.    };
13.   };
14.  echo "</script>";
```

Figur 5.26 PHP innhenting av data

På linje 1-2 henter vi informasjon om alle bordene som er med i en eksisterende session. Dette filtreres ved bruk av MySQL Queryen «TableSessionID IS NOT NULL».

Linje 3-4 skriver ut JavaScript kode med *echo*. Vi lager en JavaScript-variabel som heter Vals.

Mellom linje 5-13 kjører vi en *while*-løkke liknende den som er brukt i StudassMain. Her filtrerer vi bordene i henhold til hvilken session de er med i. På denne måten kan man passe på at kun de bordene som er registrert på deg er mulig å endre. Dette utføres på 9, hvor man deaktiverer knapper som ikke tilhører den aktive brukeren. På 11 lagrer man alle bord som tilhører brukeren.

Script for innlogging

```
1.  $UserName = $_POST["UserName"];
2.      $Password = $_POST["UserPwd"];
3.
4.      $user_query = "SELECT UserLevel, UserID FROM UserCred WHERE UserName = '$UserName' AND UserKey =
    '$Password'";
5.      $user_data = $tilkobling -> query($user_query);
6.      $user_data_array = mysqli_fetch_array($user_data);
7.      $userID = $user_data_array["UserID"];
8.
9.      $session_query = "SELECT SessionID FROM SessionList WHERE SessionUserID = '$userID'";
10.     $session_data = $tilkobling -> query($session_query);
11.     $session_data_array = mysqli_fetch_array($session_data);
12.     $sessionID = $session_data_array["SessionID"];
13.
14.
15.     if (isset($sessionID)) {
16.                     header("Location: /TableReg.php/?ID=". $sessionID .""); //hvis man har session id send de til
    bordregistrering
17.     }else{
18.                     if ($user_data_array["UserLevel"] > 1) { //kan evt byttes ut med ENUM
19.                                     $data = "INSERT INTO SessionList(SessionUserID) VALUES ($userID)";
20.                                     $tilkobling -> query($data);
21.                                     $session_query = "SELECT SessionID FROM SessionList WHERE SessionUserID =
    '$userID'"; //ellers lag session og send de videre
22.                                     $session_data = $tilkobling -> query($session_query);
23.                                     $session_data_array = mysqli_fetch_array($session_data);
24.                                     $sessionID = $session_data_array["SessionID"];
25.                                     header("Location: /TableReg.php/?ID=". $sessionID ."");
26.                     }else{
27.                                     echo "No user with that key exists";
28.                                     echo "<button onclick='Back()'>Tilbake</button>"; //brukeren har ikke tilgang.
29.                     };
30.     };
31.
```

Figur 5.27 PHP inloggingsscript

Etter tilkobling til databasen med fastsatt kode henter vi brukernavn- og passorddata som ble sendt via POST metoden på 1-2. På 4-7 sjekker og henter vi ut UserLevel og UserID fra brukeren som har det satte brukernavn- og passord vi hentet i 1-2. Fra 9-12 henter vi den

tilhørende sessionID-verdien til den valgte brukeren som vi hentet ID nummeret for i 7. Hvis brukeren allerede har en sessionID, videresender vi brukeren til TableReg med tilhørende sessionID. Dette gjøres på linje 15-16. Om brukeren ikke har noen session fra før, må vi skape en. Dette gjøres mellom 18-26. Her sjekker vi først om brukeren har tilganger over studentnivå, henholdsvis admin eller studass. Om brukeren ikke har tilgangene får du feilmelding med knapp som sender deg tilbake. Deretter legges userID-nummeret inn i sessionlisten på linje 19-20. Etterfulgt av dette vil det automatisk lages en sessionID tilhørende den brukeren. Vi må dermed hente denne ut på linje 23-24. På linje 25 videresender man brukeren med tilhørende sessionID.

Registrere setevalg

Etter man har registrert alle setene kjøres skriptet vist i figur 5.28. Dette for å sende ut informasjonen til databasen.

```
1.  $Plasser = $_POST["PlassValg"];
2.      $sessionID = $_POST["sessionID"];
3.      $select_query = "SELECT TableID FROM TableList WHERE TableSessionID = '$sessionID'";
4.      $select_data = $tilkobling -> query($select_query);
5.      while($rad = mysqli_fetch_array($select_data)){
6.                  $Plass = $rad["TableID"];
7.                  $remove_query = "UPDATE TableList SET TableSessionID = NULL WHERE TableID = '$Plass'";
    //fjerner alle tidligere bord med den session
8.                  $tilkobling -> query($remove_query);
9.      };
10.     $update_query = "UPDATE TableList SET TableSessionID = '$sessionID' WHERE TableID in ($Plasser)"; //legger
    til de nye bordene
11.     $tilkobling -> query($update_query);
```

Figur 5.28 PHP Bordvalg

Først henter man ut plassvalgene på 1. Eksempelvis, hvis man valgte sete 1 og 5 vil $Plasser-variabelen inneholde «1,5», med komma mellom hvert bordnummer. Dette gjør det enkelt å legge det inn i databasen siden man kan legge variabelen rett inn i MySQL Query. Ved bruk av «Where Tableid IN ($Plasser)» kan man enkelt legge valg av plasser direkte inn, ettersom MySQL separerer verdier med komma.

På linje 3-4 velger man ut alle bord som tidligere var registrert på den aktive SessionID-verdien. Videre på 5-9 oppdaterer man alle verdiene til de bordene som tidligere var aktive og nullstiller de. Dette er på grunnlag av at man har nye verdier som man skal ha inn, og da må man fjerne de gamle valgene. Det er derfor like enkelt å bare fjerne alle gamle og legge til på nytt.

På linje 10-11 bruker man UPDATE MySQL Query for å legge inn de nye valgene.

JavaScript (SeatSelector)

På sidene for studentassistenter brukes JavaScript for å bygge funksjonalitet for knapper, samt for å oppdatere siden hvert 30. sekund. Den viktigste bruken av JavaScript er ved valg av bord. Dette systemet er bygd med følgende JavaScript kode vist i figur 5.29.

```
1.  <script type="text/javascript">
2.    // Systemet er lagd slik at man kan bare legge til flere knapper hvor verdiene er i samsvar med setenummereringen i
    databasen og at de har class = "Sel" så vil det fungere.
3.    var Plass = document.getElementsByClassName('Sel'); // Finner alle knappene som skal brukes for valg av seter
4.    var PlassSel = []; // Lager en tom rad for å midlertidelig lagre dataen
5.    for (let i = 0, l = Plass.length; i <= l; i++) { // Lager plass i raden og styringsfunksjon ValgSete() for hver knapp
6.      PlassSel[i] = 0;
7.        Plass[i].addEventListener('click', function VelgSete() {
8.      Sete = Plass[i].id;
9.      if (PlassSel[Sete] == 0 ){
10.       PlassSel[Sete] = 1;
11.       Plass[i].style.backgroundColor = "#7FFF00"; // ved trykk endres fargen og korrisponderende index i PlassSel blir
    endret til 1
12.
13.     } else if (PlassSel[Sete] == 1) {
14.       PlassSel[Sete] = 0;
15.       Plass[i].style.backgroundColor = "#FFFFFF";
16.     };
17.      });
18.     };
19.     let LocString = "";
20.       var LocPlassSel = PlassSel.slice(); // lager lokale verdier for å ikke ødelegge brukerens valg
21.       var CurrentIndex = PlassSel.indexOf(1);
22.       while(CurrentIndex != -1){
23.         LocString = LocString + CurrentIndex;
24.         LocPlassSel[CurrentIndex] = 0; // bygger string som skal inn i php form og legger inn dynamisk
25.         CurrentIndex = LocPlassSel.indexOf(1);
26.         if (CurrentIndex != -1) { //avslutter når det ikke lenger er noen flere 1 i PlassSel
27.           LocString = LocString + ",";
28.         };
29.       };
30.       document.getElementById('PlassValgBoks').value = LocString;
31.
32.     function loadVals(){
33.       for (var i = Vals.length - 1; i >= 0; i--) { // skal preloade alle bord som du allerede har registert før på deg.
34.         document.getElementById(Vals[i]).click();
35.       };
36.     };
37. </script>
38.
```

Figur 5.29 SeatSelector JavaScript kode

Som vist i Figur 5.21 har alle knappene som er bygd for dette systemet sine egne identifikatorer som tilsvarer bordnummeret de har i virkeligheten.

```
<input type="button" class="Sel" id="1" value="1">
```

Her brukes *class* som felles indikator for alle knappene og *id* for å kunne velge ut individuelle knapper med JavaScript. *Value* benyttes for å vise til brukeren hvilket bordnummer knappen skal representere i systemet.

Gitt figur 5.29 lager vi på linje 3-4 to variabler. Variabelen «Plass» inneholder en liste over alle knappene i systemet. «PlassSel» er en variabel som inneholder en tom vektor. Denne holder oversikten over alle knappenes tilstand.

Mellom linje 5-18 lager vi en *for*-løkke. Denne kjører alt innenfor hver knapp i «Plass»-variabelen. Man legger en tom posisjon i vektoren PlassSel for hver knapp. Systemet har opprettet en event listener for enhver knapp, fra 7-16. Denne venter til en knapp blir trykket og kjører tilhørende kode VelgSete().

VelgSete()-funksjonen endrer på tilhørende verdi i PlassSel-variabelen og endrer på fargen på knappen for å vise at den er trykket. Videre på 19-30 henter vi ut alle posisjoner i PlassSel-variabelen hvor verdien har blitt satt til 1, og skriver indeksverdien til denne inn i en kommaseparert variabel LocString. Når alle indeksverdier er skrevet ut endres verdien til HTML-elementet PlassValgBoks til LocString.

LoadVal funksjonen blir kallet når systemet starter opp. Den er bygget for å trykke på alle knapper som brukeren allerede hadde registrert tidligere. Slik vil de bli endret til gul farge, samt automatisk laste inn verdier.

5.3 Admin

Dette kapittelet vil ta for seg designprosessen for verktøyene administrator vil ha for å registrere nye brukere i systemet.

5.3.1 Kortregistrering

Vi forventer at studentene benytter seg av adgangskortet for innlogging, og må derfor designe en enkel måte å legge denne dataen inn i databasen. Vi designet et system som lar deg legge til brukere i systemet som studenter. Denne administrasjonssiden vil en få tilgang til dersom en skanner et kort med administrative rettigheter på en vilkårlig studentplass. Følgende HTML-kode for administrasjonssiden er vist i figur 5.30.

```
1.  <!DOCTYPE html>
2.  <html>
3.  <?php
4.    $Table = $_GET["ID"]; // Hvert bord vil ha et bordnummer som legges i URL via startopp program
5.  ?>
6.  <head>
7.  </head>
8.  <body>
9.  <strong>Skriv inn navn først. Deretter scan kort</strong><br>
10. <form method="post" action="/Reg.php">
11.   <input type="hidden" name="ID" value="<?php echo $Table ?>">
12.   <label>
13.     Navn <br>
14.     <input type="text" name="UserName">
15.   </label> <br>
16.   <label>
17.     Key <br>
18.     <input type="text" name="UserKey" id="input">
19.   </label> <br>
20.   <input type="submit" name="Submit"> <br>
21. </form>
22. <br><button onclick='Back()'>Tilbake</button>
23. </body>
24. </html>
```

Figur 5.30 Admin HTML-kode

Her er det bygd et HTML Form som inneholder tekstfelt for navn og RFID-informasjon. Ved bruk av kode for input av RFID-informasjon som er forklart i kapittel 5.2.3 kan man skanne RFID kort. Informasjonen blir sendt videre til PHP-skriptet vist i figur 5.31.

```
1.    $Key = $_POST["UserKey"];
2.    $Name = $_POST["UserName"];
3.    $Table = $_POST["ID"];
4.
5.    $ver = "SELECT UserKey FROM UserCred WHERE UserKey = '$Key'";
6.    $ver_data = $tilkobling -> query($ver);
7.    $data_array = mysqli_fetch_array($ver_data);
8.    $ver_info = $data_array["UserKey"];
9.    if(isset($ver_info)){
10.       echo "Dette kortet er allerede registrert";
11.   }elseif ($Name != "" AND $Key != "") {
12.     $data = "INSERT INTO UserCred (UserKey,UserLevel,UserName) VALUES ('$Key','1','$Name')"; //legg inn bruker
      som student nivå.
13.     $tilkobling -> query($data);
14.     echo "Følgende navn lagt inn:" . $Name ."Som student";
15.   }else{
16.     echo "Ingenting ble lagt inn. prøv på nytt";
17.   };
18.   echo "<br><button onclick='Back()'>Tilbake</button>";
19.
```

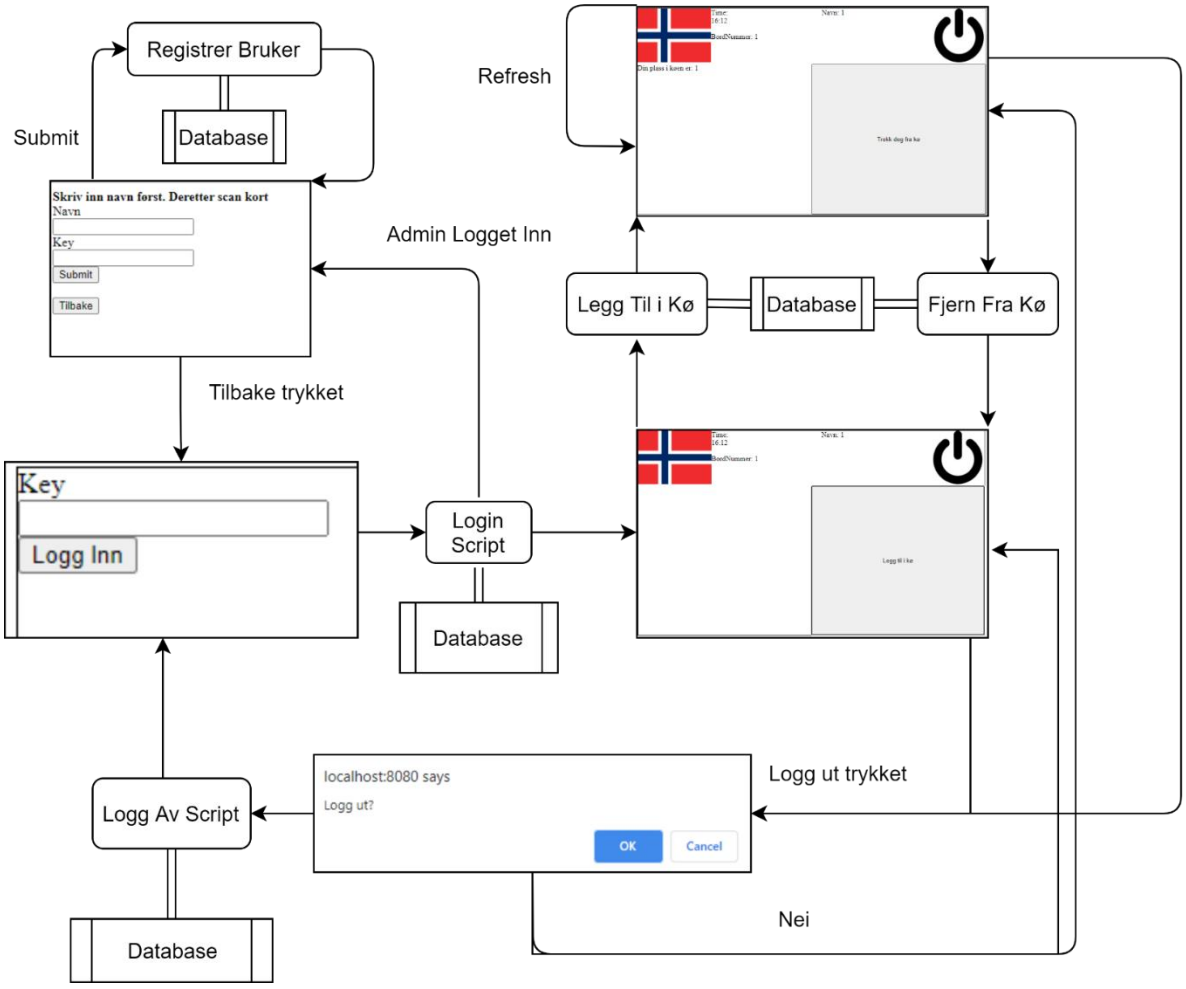
Figur 5.31 PHP admin script

På linje 1-2 henter man ut informasjonen om brukeren som skal registreres. Etterfulgt av dette sjekker man om personen allerede finnes i systemet fra 5-10. Feilmelding vil bli oppgitt dersom brukeren allerede eksisterer i systemet. Om ikke, sjekker en kontroll om input-verdiene er valide. Så lenge det eksisterer informasjon for navn og RFID vil man kjøre en «Insert» MySQL Query som legger til brukere i UserCred med tilgangsnivå som student. 14 vil gi tilbakemelding til brukeren dersom studenten er lagt til i systemet. På linje 16 har man en feilmelding som aktiveres hvis noe er galt, eksempelvis om en av input-verdiene er tomme. På 18 er det en knapp som lar deg gå tilbake til administrasjonssiden, ettersom det er forventet at denne prosessen foretar registreringen av flere brukere i praksis.

5.4 Resultat

5.4.1 Student

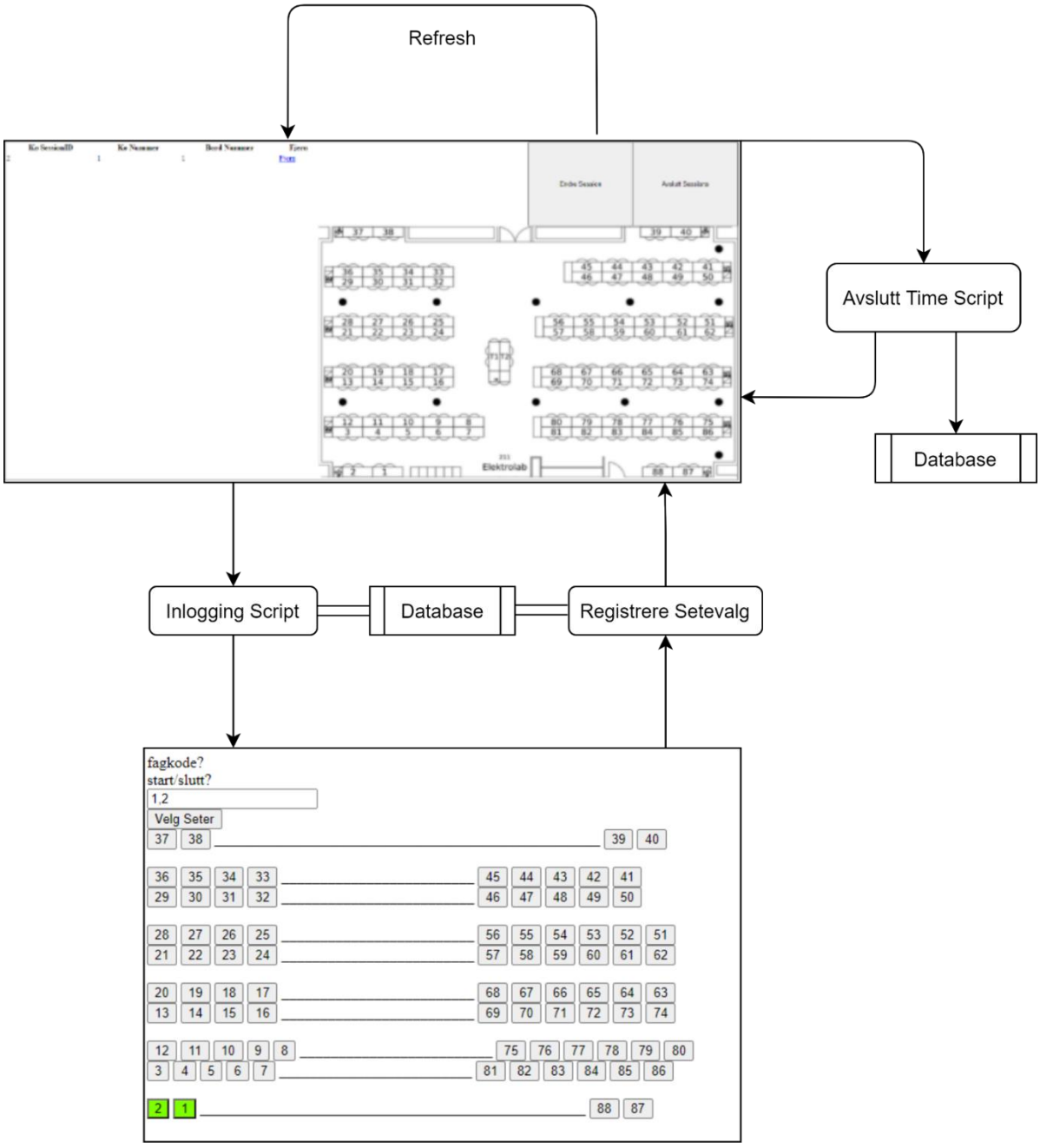
Figur 5.32 viser til hvordan systemet for studenter vil fungere. Bildene viser hvordan systemet vil reagere til input og hvordan dette vil vise seg for bruker. Det viser og oversikten over hvordan data blir håndtert med kommunikasjon mot databasen. Resultatet av systemvalgene og kodingen er et system som tillater innlogging ved RFID-input og navigasjon ved knappetrykk og touch-skjerm. Brukergrensesnittet utnytter skjermplassen sparsomt, og gir derfor muligheten for utvidelser til nye moduler i fremtiden.



Figur 5.32 Student Flowchart

5.4.2 Studass

For studentassistenter er nettsidene designet med muligheter for å skille kø og bord mellom flere studasser. Slik kan en delegere ansvaret for klasserommet opp systematisk. Det er også muligheter for å avslutte og logge av brukere fra studass sine nettsider. Figur 5.33 viser hvordan kommunikasjonen vil vise seg for brukeren og hvordan denne er håndtert til databasen.



Figur 5.33 Oppkobling av prototypemodul

5.5 Drøfting

5.5.1 Valg av designspråk

Valg av relativt enkle og usikre metoder for systemet var gjort på grunnlag av tidligere erfaring innenfor HTML, PHP og CSS. Ved å bruke programmeringsspråkene gruppen hadde kjennskap til fra før, kunne en delegere mer av tidsrommet til å finne gode løsninger til systemet.

5.5.2 Kompleksitet

Systemet er bygd med relativt enkelt kodegrunnlag. Dette ble gjort på grunn av at systemet skal være skalerbart, og ved kompleksitet medfølger stabilitetssenking. Ved bruk av enkel kode vil vi forsikre oss at systemet er skalerbart uten store økninger i krav for prosesseringskraft. Server vil kun måtte forholde seg til HTML-filer og PHP-kode som utfører enkle MySQL-kommandoer.

5.5.3 Bedre administrasjonsverktøy

Det nåværende systemet har veldig få administrative verktøy. Optimalt ville en kunne ha administrert brukere registrert i systemet samt ha en bedre kontroll over innlogging på bordene individuelt. Dette ble ikke gjort på grunnlag av at administrative verktøy ikke er nødvendig for en fungerende prototype. Dette ble dermed nedprioritert over ferdigstilling av system.

Optimalt ville en hatt et system for administrering av bord via en nettside, hvor en kan logge av individuelle brukere eller sessions. Det er fornuftig å ha en måte å få oversikt over alle som er logget inn når de ikke er lagt i kø. I tillegg burde man utvide og forbedre metoder for å registrere nye brukere samt endre på allerede registrerte brukere. Eksempelvis, om en skal registrere noen som studass må dette legges inn manuelt i databasen via en MySQL GUI eller som en kommando direkte på serveren.

5.5.4 Innlogging og input

Systemene som er bygd for innlogging av brukere i systemet, er bygd på prinsippet at man ikke endrer nettsideadressen. Innlogging er egentlig bare en registrering av en bruker på et bord. Det er ingen funksjon for automatisk avlogging hvis personen går ifra systemet uten å logge seg av. Det er heller ingen sikkerhet for hvordan brukernavn- og passordvariablene blir håndtert. Systemet er ikke bygd for filtrering av MySQL Injections heller. Alle passord er lagret i tekstform på databasen uten kryptering. Alle variabler sendes mellom databasen og systemene direkte uten annen sikkerhet enn det som ligger internt i PHP.

Optimalt hadde man endret dette slik at systemet er selvkorrigerende i forhold til innlogging over lang tid. Det er naturlig å logge brukere av etter f.eks en time uten aktivitet. Samtidig burde man legge til krypteringsalgoritmer ved prosessering av RFID- og passorddata. Slik lagres ikke brukerinformasjonen direkte på databasen. I tillegg filtrere ut muligheter for MySQL Injections. Den nåværende implementasjonen er dårlig på grunnlag av at sikkerhet ikke er tatt hensyn til forutsett systemets lukkede natur.

5.5.5 Bedre håndtering av knappetrykk

Måten navigasjon med knapper på nettsiden er satt opp for øyeblikket er ikke et system som er lett skalerbart for videre funksjonalitet, som for eksempel kontroll av lab-enheter via PLS-systemet. Optimalt burde Javascript-systemet bygges på objektbasert navigering mellom HTML-elementene. Dette er noe som er godt støttet innen Javascript og vil gjøre at man kan lettere skalere systemet med ny funksjonalitet. I senere tid slipper man å omgjøre hele navigasjonssystemet hver gang.

5.5.6 Datatyper i database

Flere av datatypene som er brukt for tabellene i databasen er ikke valgt for å være 100% plasseffektive. Mange av de har mer kapasitet enn nødvendig for systemets funksjonalitet. Metoden som er brukt for identifisering av brukertilganger på systemet burde eksempelvis endres til ENUM, i forhold til et nummerert system som er i bruk nå.

5.5.7 AJAX

Metodikk for oppdatering av endringer på databaseverdier samt framvisning av disse er nå bygd på at systemene forespør og laster om nettsiden ved satte tidsintervaller. Dette er ikke gunstig for båndbredde eller brukervennlighet. Ved omlasting av nettsiden vil det slette all input som man holder på med, og systemet har større sjanse for å feile.

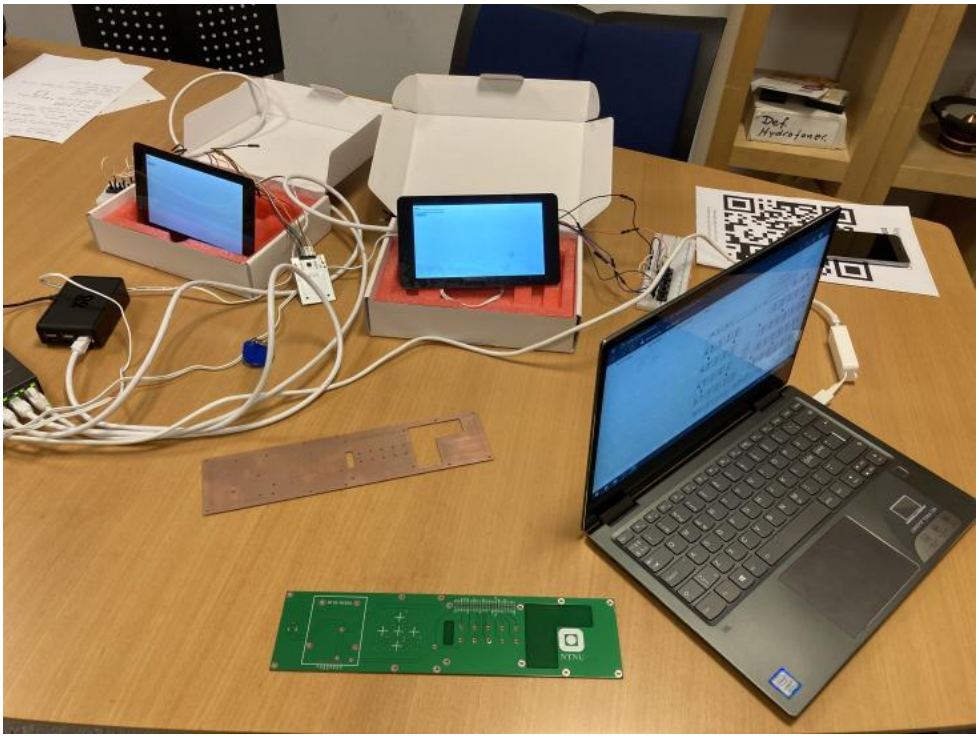
Optimalt hadde man sluppet at enhetene forespør endringer fra databasen. Istedenfor kan databasen sende ut nye oppdateringer til de rette enhetene når den oppdateres. En måte å løse dette på som gruppen undersøkte, men ikke gjennomførte grunnet tidsrestriskjoner, er AJAX (Asynchronous JavaScript and XML). AJAX lar deg opprette kommunikasjon med databasen som kan oppdatere databaseutskrift uten omlasting av nettsiden. Ved bruk av AJAX kan man bygge løsninger med JavaScript som vil oppfylle dette kravet om lavere båndbreddebruk og et mer moderne design.

6 Resultat

Dette kapittelet vil omhandle den helhetlige løsningen for prosjektet. Løsningen er bygd ved å sy sammen de forskjellige arbeidsområdene som ble utført til en samlet prototype. Denne prototypen består av to eller flere Raspberry Pi-systemer som er koblet til en felles server via en nettverks-svitsj. En ekstern datamaskin fungerer som tilkoblingspunkt for studass. Protypen lar deg danne sessions og logge inn/ut av alle tilkoblede enheter og teste kø-funksjonaliteten til systemet.

6.1 Konstruksjon av system

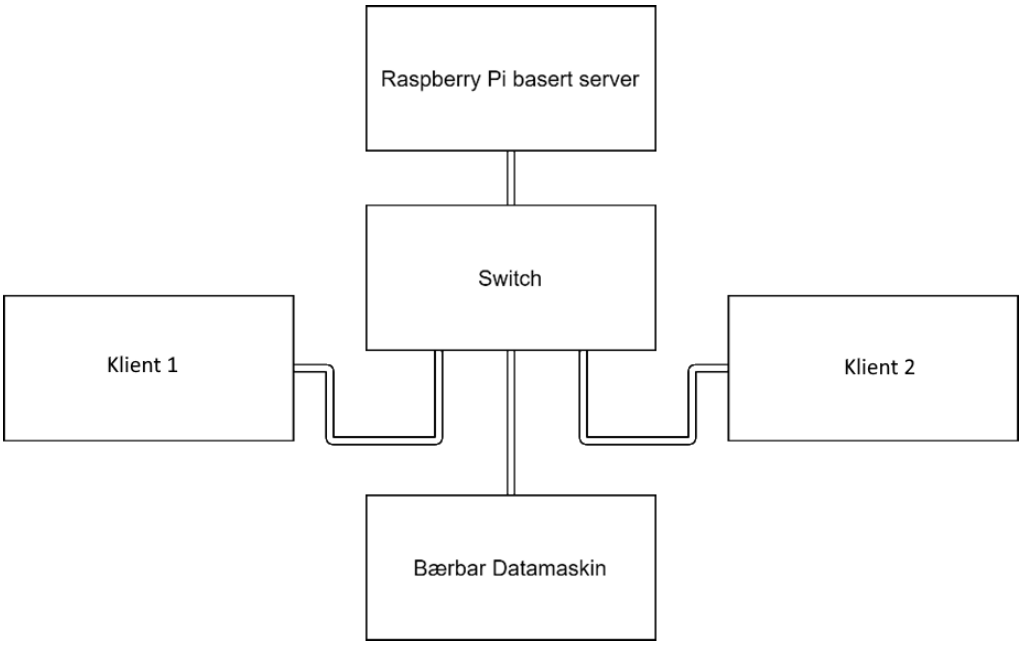
De forskjellige enhetene for det oppkoblede systemet ble satt sammen vist i figur 6.1. Her har vi benyttet oss av to Raspberry Pi-systemer. RFID-leser ble koblet til ved enkel kabling direkte til GPIO-pinner på Raspberry Pi-modulen. Knappene ble montert på et breadboard og koblet til GPIO-pinner på samme måte som RFID-leseren. Skjermen ble montert sammen med Raspberry Pi-kortet ved hjelp av standoffs med skruehull som kommer med skjermmodulen. Data til skjermen er sendt via systemets «DSI connector». All programvare som er designet for systemet er lagt til på SD-kort som er satt inn i Raspberry Pi systemene. For prototyping er ikke PCB kortet brukt for oppkobling.



Figur 6.1 Oppkoblet prototype

6.2 Oppkobling

Oppkobling av systemet for prototyping følger det satte diagrammet vist i figur 6.2. Her er en Raspberry Pi-basert server med Ubuntu brukt som en analog for det eksisterende serversystemet som er på elektrolaboratoriet nå. Serveren hostet alle filene og den nødvendige databasen for bruk av denne nettbaserte løsningen. Den fysiske oppkoblingen av systemet ble gjort via en ekstern 5-port svitsj. Dette er nødvendig for oppkobling av to eller flere enheter samt testing av systemets validitet. Ved bruk av DNS på server kunne man koble til en datamaskin for å administrere databaser og alle enheter via SSH. Denne viser og studass-sidene samt administrerer køsystemet enkelt i prototyping.



Figur 6.2 Koblingsskjema av køsystem via ethernet

7 Drøfting

7.1 Teknisk drøfting

Resultatet av prosjektet er en prototype av køsystemet som bruker mange av de konseptene som vi har utarbeidet under arbeidsprosessen. Prototypen viser at måten som vi har prøvd å løse oppgaven på virker i praksis, men har noen mangler. I henhold til utarbeidede konsepter, har gruppen gjort noen erfaringer som kan bidra til forbedring og videre utvikling. I dette kapittelet vil vi oppsummere noen observasjoner som vi ser for oss bør forbedres. Vi vil i grove trekk komme med forslag til hvordan dette kan gjennomføres.

7.1.1 Installasjon og vedlikehold av køsystemet

For en systemadministrator er det viktig at systemet er enkelt å installere og vedlikeholde. I prototypen vår har vi prøvd å ta hensyn til dette ved at alle innstillinger og programmer som skal installeres er inkludert i en fil som kan lastes direkte inn på minnekortet til enheten. Selv om installasjonen er relativt enkel, må fortsatt en systemadministrator logge seg inn på hver eneste enhet for å sette hvilken nettside som skal vises når enheten starter opp. Vedkommende må også sette identifikasjonen til den enkelte enheten i denne nettsideadressen. Dette er med hensikt gjort slik i prototypen, fordi det gav oss fleksibilitet under utviklingsprosessen. For en systemadministrator vil dette ta veldig lang tid å gjøre dersom man skal sette i drift flere enheter. For å løse dette problemet, bør en ferdig versjon av køsystemet inkludere nettsideadressen som skal brukes ved installasjon. Dette vil gjøre livet veldig mye lettere for en systemadministrator.

For å få systemet til å inkludere en standard nettsideadresse ved installasjon, slik som beskrevet i avsnittet over, må man også finne en løsning hvor identifikasjonen til enheten blir satt automatisk. I prototypen må identifikasjonsnummeret settes manuelt inn i en del av nettsideadressen. En løsning på dette, som blir presentert i kapittel 3.4.3, kan være å bruke en såkalt hardware-ID hvor man får enheten til å lese av en elektronisk ID fra kretskortet som den er montert på. Denne ID-en kan for eksempel komme fra en rotasjonsbryter. Rotasjonsbryteren må også manuelt settes, men har den fordelen at dette kun trenger å gjøres én gang. Dersom enhetens minnekort går korrupt, vil fortsatt ID-en være intakt, og ved reinstallasjon slipper systemadministrator å logge seg inn på enheten for å gjøre endringer. Denne ID-en må leses av enheten, for eksempel via GPIO og bruk av en spesialisert daemon.

ID-en må også legges inn i nettsideadressen. Dette kan gjøres ved hjelp av en BASH-skript som redigerer tekstdokumentet hvor nettsideadressen ligger lagret. Ved å bruke denne løsningen vil installasjon og vedlikehold av systemet ikke kreve innlogging på enhetene, og vil være mye lettere for systemadministratoren.

7.1.2 Brukergrensesnitt

Brukergrensesnittet på prototypen er ment å demonstrere funksjonaliteten til køsystemet. Det er muligheter for å navigere både med knapper og med touch-skjerm, og man har mulighet til å logge seg inn og å logge seg ut. Utover dette, har vi ikke hatt et veldig stort fokus på å optimalisere brukervennligheten på designet av nettsidene. I et system som skal settes i drift, ser vi for oss at nettsidene for både studenter og for studentassistenter har fått en overhaling. Dette kan blitt gjort ved å legge til egne figurer og symboler for å representere knapper på skjermen, og å endre farger og utlegg på nettsidene for å pynte på dem. Dette vil sannsynligvis gi brukerne av systemet en mer positiv opplevelse, og vil også kanskje gjøre systemet mer intuitivt.

7.1.3 Optimalisering av nettverkskommunikasjon

Bruk av båndbredde av den nåværende prototypen er ikke gunstig. For at endringer på nettsiden skal fremstilles for brukeren så lastes nå sidene inn på nytt ved bruk av JavaScript. En bedre løsning for dette er framstilt i kapittel 5.5.7, hvor vi foreslår bruk av en teknikk kalt AJAX.

7.1.4 Forbedre RFID-kode

Slik oppsett av RFID-leser er i dag, utnytter den for mye prosessorkraft. Dette er ikke på langtidssikt gunstig for systemet, og bør endres dersom en vil fortsette å bruke denne typen modul. Som referert i kapittel 4.4, har gruppen funnet et bibliotek som løser dette problemet. Implementasjon av dette vil være enkelt, og vil påføre systemet mindre stress. Gruppen har ikke gjort seg erfart med bedre metoder, og mener denne er tilstrekkelig for systemet.

7.1.5 Ferdigstilling av HAT og vurdering av alternativ maskinvare

Det tilpassede kretskortet er designet, men ikke testet og finjustert. Det er blitt foretatt snarveier som er tilstrekkelige for en prototype, men som må rettes på for et fullstendig produkt. Kortet må passe inn i kanalen og kunne romme alle de nødvendige komponentene. Til slutt må det være plass til mulige utvidelser dersom et PLS-system skal integreres inn i dette. Maskinvare som datamaskin og skjerm kan fortsatt revurderes for å optimalisere både kostnad og funksjonalitet. Som nevnt i kapittel 2.5 finnes det flere realistiske alternativer som kan oppfylle disse ønskene.

7.2 Drøfting av arbeidsprosess

Arbeidsprosessen er viktig å få en ordentlig oversikt over med tanke på den unike situasjonen gruppen har funnet seg i. Koronapandemien har sterkt påvirket arbeidsprosessen og til en viss grad resultatene i prosjektet i helhet. Personlig helse har alltid blitt høyest prioritert, og på grunn av smittevernreglene har gruppen måtte finne alternative metoder for samarbeid.

Først og fremst har de fleste former for møter foregått digitalt, med kun få unntak. Gruppen har benyttet seg godt av programvare laget for digitale møter og samtaler, hovedsakelig Microsoft Teams og Discord. Microsoft Teams har blitt brukt for å organisere møter mellom aktive parter, slik at alle en kan ta del i diskusjon av prosjektet uten bekymring for smitte. Det har gitt gruppen et nyttig medium for kommunikasjon med veileder og oppdragsgiver, ikke bare gjennom digitale møter, men også gjennom chatfunksjoner. Teams har også vært et svært nyttig verktøy for organisering av aktuelle filer for prosjektet. Diverse filer lagt til i Teams kan redigeres av hele gruppen samtidig slik at samkjøring av rapportskriving og loggføring av arbeid kan skje uten konflikter. Discord har blitt brukt i stor grad til møter mellom gruppemedlemmene der veileders tilstedeværelse ikke var nødvendig. Discord har den fordelen at det ikke er behov for å kalle inn til et møte for å starte en samtale og gjør det veldig enkelt for gruppen å starte en kort diskusjon av arbeidet.

Det tok gruppen en del tilvenning før vi effektivt klarte å benytte de ulike verktøyene for kommunikasjon. Som helhet har de ikke bare fungert som en god erstatning for fysiske møter, men også tidsbesparende. Fysiske møter krever nøye logistisk planlegging, og kan ofte være tidkrevende. Teams og Discord har tillatt gruppen å møtes hjemmefra slik at møtene blir langt mer fleksible. Likevel har også mangelen på fysisk kontakt negativt påvirket arbeidet. Enkelte gruppemedlemmer har lettere for å konsentrere seg i et mer arbeidsorientert miljø, og sliter

med å holde en god arbeidsmoral fra hjemmet sitt. Dette har ført til svært varierende produktivitet utover prosjektets tidsrom. Ikke før testing av prototypen som en helhet begynte, møttes gruppen fysisk i samme rom. Etter det har det vært flere samlinger for de som arbeider bedre på denne måten.

Til tross for den besparte tiden fra de digitale møtene har koronasituasjonen også tapt gruppen en god del tid i form av venting på levering av komponenter. All postvirksomhet i hele verden har saknet, noe som har hatt en merkbar påvirkning på prosjektets arbeidsprosess. Dette har stilt gruppen et større krav på å fremstille resultater tidligere. Det har også gått bort en del tid på å finne alternativer til det ønskede utstyret for å sikre at det blir sendt og mottatt i tide. Som en helhet har prosjektarbeidet blitt strukket ut over en lengre tid enn det kanskje ville blitt dersom pandemien ikke var en hindring.

I begynnelsen av prosjektet, før utviklingen av systemet begynte, satte gruppen fokus på opplæring av nytt stoff. Oppgaven krevde mye varierende kunnskap og erfaring i programvareutvikling, noe gruppen hadde svært lite av. Utenom programmering av enkle mikrokontrollere, som er en del av pensum for elektroingeniørstudenter på NTNU, var det kun to gruppemedlemmer som hadde tidligere erfaring i grunnleggende web-design. Dette stod sentralt i utviklingen av systemet. På grunn av dette ble flere uker benyttet til kursing for å bygge grunnleggende forståelse i de relevante programmeringsspråkene. Denne opplæringsperioden var dårlig strukturert, og de fleste gruppemedlemmene hadde bestemt seg for å få en god oversikt over alt som var nødvendig. I ettertid ble det diskutert om dette var den rette måten å ta for seg opplæringen. På den ene siden var det ikke nødvendig for at alle gruppemedlemmene skulle kunne alt. Prosjektet ble delt inn i fire hoveddeler som hvert medlem fikk ansvaret for. Slik fikk alle benyttet den forkunnskapen og erfaringen de hadde på mest effektiv måte. På en annen side har denne opplæringen gjort det mulig for gruppemedlemmene å bli kjent med nye teknikker som kan være nyttige i et fremtidig arbeidsliv. Selv om det kunne blitt spart tid på å spesialisere opplæringen, har hvert gruppemedlem fått god nytte av den bredere opplæringen som ble gjennomgått.

Gjennom prosjektløpet har gruppen hatt flere møter med oppdragsgiver og veileder med jevne mellomrom for å sikre at arbeidet som blir gjort er etter oppdragsgivers ønsker. På grunn av dette har det oppstått nye åpenbaringer om alternative løsninger til diverse problemer underveis. Sluttmålet for prosjektet har derfor vært i endring for å sikre at gruppen fullfører med den beste løsningen. Den største overgangen gruppen foretok fra forprosjektet, er endringen til en web-basert løsning fremfor en selvkodet en. Dette er grunnet enklere oppsett

og bedre støtte for utvidelser. Takket være avgjørelsen har gruppen laget en prototype som viser godt hvordan et fullstendig system kan se ut, og som er svært fleksibelt for videre utvikling.

8 Konklusjon

Oppgaven for prosjektet var å designe et køsystem for å løse problemstillingen knyttet til studentenes bruk av elektrolaboratoriet til IES. Gruppen tok som mål å utvikle en prototype som ville danne et fleksibelt grunnlag for fremtidige implementasjoner av systemet. Prosjektet har innfridd målene gitt gruppens avgrensinger definert i kapittel 1.3, og presenterer et levedyktig konsept som løsning på utfordringen knyttet til oppgaven. Løsningen er bygd opp av et SBC av typen Raspberry Pi 4 som håndterer instruksjoner og kommunikasjon mellom bruker og server. Denne er oppkoblet til en LCD med kapasitiv touch via DSI-grensesnitt, Navimec 5G navigeringsknapper samt en VMA405 RFID-leser. Systemet nytter en web-basert løsning og tilbyr registrering og innlogging av bruker til køsystemet. Modulen vil avlese studentkortet, samt gi brukeren muligheten til navigering med touchskjerm og knapper. Et sentralisert tilkoblingspunkt for studentassistenter er implementert, og gir mulighet for framvisning samt administrering av systemets tilstand. På grunnlag av funksjonalitet nytter systemet seg av daemons og tilbyr høy fleksibilitet. Et forslag for PCB til installasjon i INKA101 kanal er også presentert.

En stor del av prosjektet har krevet ferdigheter utenfor gruppens eksisterende felt. Dette har krevet mye tid for innhenting av nødvendig kompetanse på flere områder innenfor Linux, programmering og IT-kunnskaper. Gruppen har gjennom prosjektet tilegnet seg ny kunnskap som har bidratt til å levere en prototype som inkorporerer IT og teknologi. Arbeidet er godt dokumentert, og tilrettelegger for fremtidig videreutvikling. Erfaringen stiller gruppen positiv til, og mener den er relevant for fremtidig arbeid om ingeniører.

Bibliografi

[1] ALLPCB. (2017). *1 Layer PCB* [Bilde]. ALLPCB. https://file.allpcb.com/web/image/20180731/636686423230994649165959 9.jpg

[2] ALLPCB. (2017). *Single Layer PCB vs Double Layer PCB* [Bilde]. ALLPCB. https://file.allpcb.com/web/image/20180731/636686423717683259629556 9.png

[3] ALLPCB. (2017, 11. april). *1 Layer PCB – Detailed Introduction from 6 Aspects*. ALLPCB. https://www.allpcb.com/1_layer_pcb.html

[4] Amos, E. (2016). *Raspberry Pi 2 Model B* [Bilde]. Wikipedia. https://en.wikipedia.org/wiki/Single-board_computer#/media/File:Raspberry-Pi-2-Bare-BR.jpg

[5] atlasRFIDstore. (2019, 10. desember). *Active RFID vs. Passive RFID: What's the Difference?*. atlasRFIDstore. https://www.atlasrfidstore.com/rfid-insider/active-rfid-vs-passive-rfid

[6] Brown, E. (2020). *Raspberry Pi Zero WH* [Bilde]. LinuxGizmos. http://linuxgizmos.com/files/rpi_zerowh.jpg

[7] Campbell, S. (2016, 13. februar). *BASICS OF THE I2C COMMUNICATION PROTOCOL*. Circuit Basics. https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/

[8] Campbell, S. (2017, 11. april). *Basics of the I2C Communication Protocol*. Hentet mai 18, 2021 fra https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/

[9] Campbell, S. (2017b, 11. april). *Basics of UART Communication*. Hentet mai 18, 2021 fra https://www.circuitbasics.com/basics-uart-communication/

[10] Canonical. (u.d.). *Domain Name Service (DNS)*. Hentet mai 18, 2021, fra https://ubuntu.com/server/docs/service-domain-name-service-dns

[11] Canonical. (u.d.). *Dynamic Host Configuration Protocol*. Hentet mai 18, 2021 fra https://ubuntu.com/server/docs/network-dhcp

[12] Canonical. (u.d.). *HTTPD - Apache2 Web Server*. Hentet mai 18, 2021 fra https://ubuntu.com/server/docs/web-servers-apache

[13] Caudle et al., (2018, 28. juni). *Python Spidev*. Hentet mai 18, 2021 fra https://github.com/doceme/py-spidev

[14] Cawley, C. (29, november 2017). *The 7 Best Lightweight Operating Systems for Raspberry Pi*. Hentet mai 18, 2021 fra makeuseof: https://www.makeuseof.com/tag/lightweight-operating-systems-raspberry-pi/

[15] Computer memory. (2021, 30. april). I *Wikipedia.* https://en.wikipedia.org/wiki/Computer_memory

[16] Distil Union. (Hentet 2021, 5. mai). *Debit Card with RFID* [Bilde]. Distil Union. https://distilunion.com/blogs/news/rfid-to-block-or-not-to-block

[17] Dunmur, D & Walton, H. G. (2015, 3. august). Liquid crystal display. I *Britannica.* https://www.britannica.com/technology/liquid-crystal-display

[18] Eastwin. (Hentet 2021, 5. mai). *Multi-layer PCB* [Bilde]. Eastwin. https://www.eastwinpcba.com/eastwin-perfect-experienced-multilayer-pcb-manufacturer-china/

[19] EM Microelectronic (2014, 07. oktober). *FACT SHEET EM4200*. Hentet mai 18, 2021 fra https://www.emmicroelectronic.com/sites/default/files/products/datasheets/4200-fs.pdf

[20] Farahat, A. (2016, 31. januar). *Switch mechanisms:Slide switches and Push-button switches*. Machinery Equipment Online. http://machineryequipmentonline.com/electric-equipment/switch-mechanismsslide-switches-and-push-button-switches/

[21] GNU. (2020, september 22). *GNU Bash*. Hentet mai 18, 2021 fra https://www.gnu.org/software/bash/

[22] Grafstein, L. (2016, 14. mars). *High Coervicity "HiCo" Low Coevicty "LoCo" Magnetic Stripe Cards*. Hentet mai 18, 2021 fra https://www.advantidge.com/whats-difference-hico-loco-cards/

[23] Harding, S. (2019). *What Is a CPU Core? A Basic Definition*. Tom's Hardware. https://www.tomshardware.com/news/cpu-core-definition,37658.html

[24] HTML. (2017). *What is HTML?*. HTML. https://html.com/#What_is_HTML

[25] indiamart. (2017). *Printed Circuit Board* [Bilde]. indiamart. https://www.indiamart.com/proddetail/double-sided-pcb-11402964555.html

[26] Intel. (2020). *CPU Frequencies Comparison* [Bilde]. Intel. https://www.intel.com/content/dam/www/public/us/en/images/photography-consumer/rwd/gaming/a1035872-cpu-frequencies-comparison-rwd.png.rendition.intel.web.1920.1080.png

[27] Intel. (2020, 29. november). *What Is Clock Speed?*. Intel. https://www.intel.com/content/www/us/en/gaming/resources/cpu-clock-speed.html

[28] Johnsen, R. (2020, 27. juli). *Kommunikasjonsprotokoll*. Hentet 18. mai 2021 fra https://snl.no/kommunikasjonsprotokoll

[29] Jones, B. (2021, mai 5). *Everything about Daemons in Linux*. Hentet mai 18, 2021 fra Foss linux: https://www.fosslinux.com/46765/daemon-linux.htm

[30] Jstrom99. (2019). *Raspberry Pi 4* [Bilde]. Wikipedia. https://en.wikipedia.org/wiki/Raspberry_Pi#/media/File:RaspberryPi_Model_4B.svg

[31] Lerch, R. (2015, October 21). *What is an init system?* Hentet mai 18, 2021 fra https://fedoramagazine.org/what-is-an-init-system/

[32] Li, B. (2018). *Resistive Touch vs Capacitive Touch – What's The Difference* [Bilde]. Digi-Key. https://forum.digikey.com/t/resistive-touch-vs-capacitive-touch-whats-the-difference/1063

[33] Li, B. (2018, 3. mai). *Resistive Touch vs Capacitive Touch – What's The Difference*. Digi-Key. https://forum.digikey.com/t/resistive-touch-vs-capacitive-touch-whats-the-difference/1063

[34] Machinery Equipment Online. (2016). *Operating principle of the momentary-contact push-for-on switch* [Bilde]. Machinery Equipment Online. http://machineryequipmentonline.com/electric-equipment/wp-content/uploads/2016/01/Switch-principles-0875.jpg

[35] Mdscott. (2020). *RFID tag* [Bilde]. IT Law Wiki. https://itlaw.wikia.org/wiki/RFID_tag?file=RFID_tag.jpg

[36] Microprocessor. (2021, 22. april). I *Wikipedia.* https://en.wikipedia.org/wiki/Microprocessor

[37] Moko Technology. (Hentet 5. mai 2021). *Through-hole and Surface-mount components* [Bilde]. Moko Technology. https://cn.mokotechnology.com/through-hole-pcb-surface-mount-pcb-which-one-do-we-recommend/

[38] Mozilla. (2013, 5. juni). *JavaScript*. Mozilla. https://developer.mozilla.org/en-US/docs/Web/JavaScript

[39] Mozilla. (2016, 20. mars). *CSS Grid Layout*. Mozilla. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout

[40] multimec. (Hentet 2021, 6. april). *Cap solution for navigating*. https://no.mouser.com/datasheet/2/26/Apem_08222017_Navimec-1158126.pdf

[41] multimec. (Hentet 2021, 6. april). *Navimec Module standard and illuminated*. https://www.mouser.com/datasheet/2/26/navimec_datasheet-221750.pdf

[42] multimec. (Hentet 2021, 6. mai). *1 LED* [Bilde]. multimec. https://no.mouser.com/datasheet/2/26/Apem_08222017_Navimec-1158126.pdf

[43] MySQL. (2016, 20. september). *What is MySQL?*. MySQL. https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

[44] Newhaven Display Intl. (Hentet 2021, 18. mai). *NHD-0440AZ-RNFBW* [Bilde]. Digi-Key. https://www.digikey.no/product-detail/no/newhaven-display-intl/NHD-0440AZ-RN-FBW/NHD-0440AZ-RN-FBW-ND/2165859#gallery-1

[45] Nexus ID Solutions AS (2017, 27. januar). *Produktkatalog 2017*. Hentet mai 18, 2021 fra https://issuu.com/nexusidsolutionsas/docs/170124_nexus_catalogue_no-v1

[46] NXP Semiconductors (2016, 27. april). *MFRC522*. Hentet mai 18, 2021 fra https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf

[47] NXP Semiconductors (2018, 23. mai). *MF1S50YYX_V1*. Hentet mai 18, 2021 fra https://www.nxp.com/docs/en/data-sheet/MF1S50YYX_V1.pdf

[48] Ondryáš et al.,(2019, 12. september). *Python RC522 library*. Hentet mai 18, 2021 fra https://github.com/ondryaso/pi-rc522

[49] PHP. (2001, 29. januar). *What can PHP do?*. PHP. https://www.php.net/manual/en/intro-whatcando.php

[50] PHP. (2009, 1. februar). *History of PHP*. PHP. https://www.php.net/manual/en/history.php.php

[51] Raspberry Pi Foundation. (2012, 20. februar). *About us*. Raspberry Pi. https://www.raspberrypi.org/about/

[52] Raspberry Pi Foundation. (2014). *Raspberry Pi 2/3/4 pin numbers* [Bilde]. Raspberry Pi Foundation. https://www.raspberrypi.org/documentation/usage/gpio/

[53] Raspberry Pi Foundation. (2014, 2. juni). *GPIO*. Raspberry Pi. https://www.raspberrypi.org/documentation/usage/gpio/

[54] Raspberry Pi Foundation. (2014, 7. september). *Power Supply*. Raspberry Pi. https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md

[55] Raspberry Pi Foundation. (2020, 16. oktober). *BCM2711*. Raspberry Pi. https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/README.md

[56] RFID tag. (Hentet 2021, 30. april). I *IT Law Wiki*. https://itlaw.wikia.org/wiki/RFID_tag

[57] Rossen, E. (2021, januar 17). *klient/tjener-teknologi*. Hentet mai 18, 2021 fra Store Norske Leksikon: https://snl.no/klient/tjener-teknologi

[58] Ryerson University. (2020). *Introduction to System-on-Chip* (COE838/EE8221). Ryerson University. https://www.ee.ryerson.ca/~courses/coe838/lectures/Intro-SoC.pdf

[59] Sedra, A. S. & Smith, K. (2015). *Microelectronic Circuits*. Oxford University Press Inc

[60] Shepard, J. (2020). *Brushed DC motor internal construction* [Bilde]. Power Electronic Tips. https://www.powerelectronictips.com/motor-fundamentals-dc-motors-faq/

[61] Single-board computer. (2021, 20. april). I *Wikipedia*. https://en.wikipedia.org/wiki/Single-board_computer

[62] SSH. (u.d.). *SSH Protocol - Secure Remote Login and File Transfer*. Hentet mai 18, 2021 fra SSH: https://www.ssh.com/academy/ssh/protocol

[63] Starting Electronics. (2012). *Liquid Crystal Display (LCD)* [Bilde]. Starting Electronics. https://startingelectronics.org/beginners/components/LCD/

[64] Techopedia. (u.d.). *X Server*. Hentet mai 18, 2021 fra https://www.techopedia.com/definition/15328/x-server

[65] The Chromium Projects. (u.d.). *Chromium*. Hentet mai 18, 2021 fra https://www.chromium.org/Home

[66] Trond, U., Halvor, B.-B., & Ola, N. (2021, januar 16). *IP-adresse*. Hentet mai 18, 2021 fra Store Norske Leksikon: https://snl.no/IP-adresse

[67] Velleman (2017, 02. februar). *VMA405*. Hentet mai 18, 2021 fra https://www.robotshop.com/media/files/pdf/arduino-compatible-rfid-read-write-module-datasheet1.pdf

[68] Vikan, D. E. (2013, 29. august). *Hvordan fungerer NTNU adgangskorte*ne. Hentet mai 18, 2021 fra https://dvikan.no/hvordan-fungerer-ntnu-adgangskortene

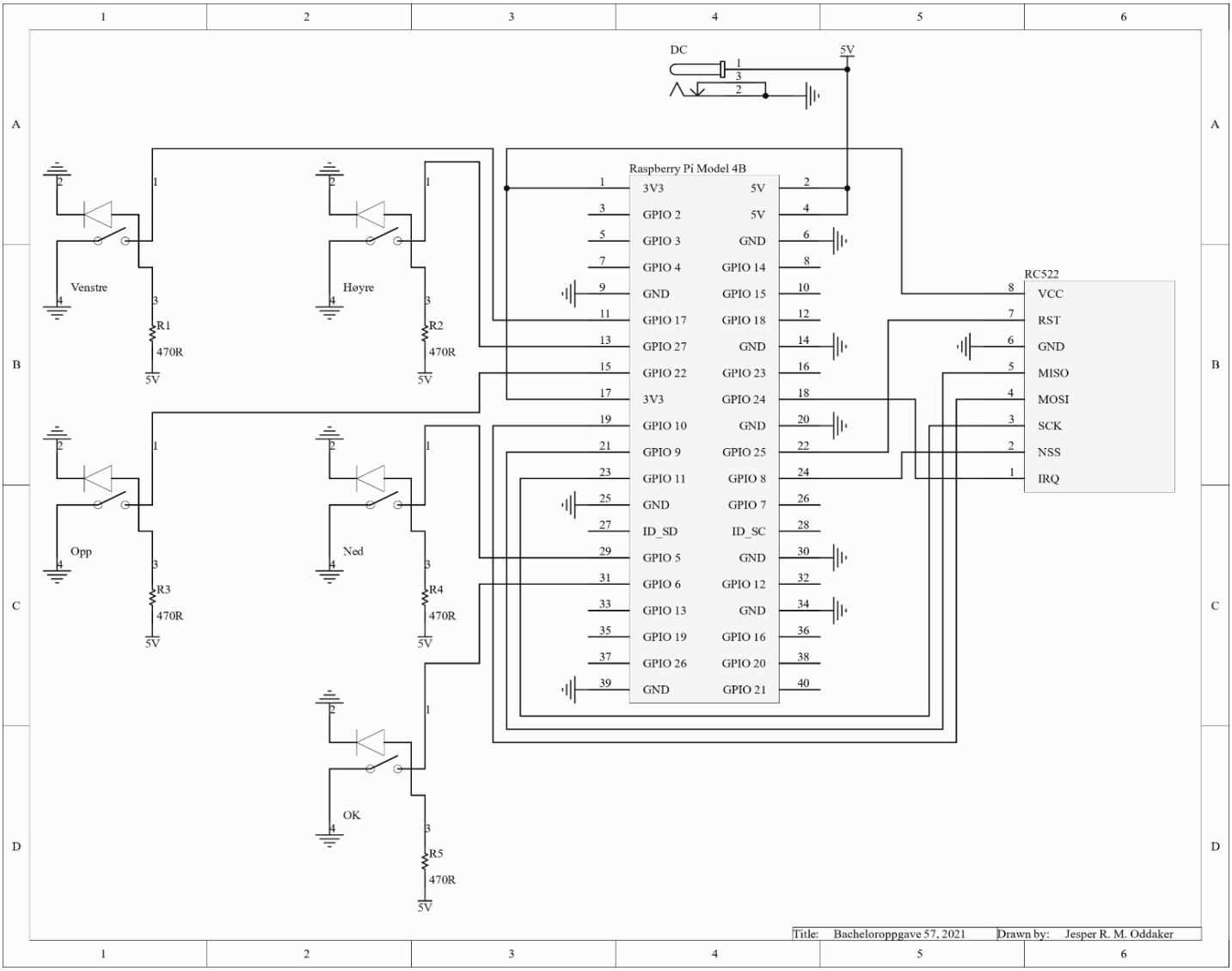
[69] Waveshare. (Hentet 2021, 4. mai). *Working with Raspberry Pi 4* [Bilde]. Waveshare. https://www.waveshare.com/4.3inch-dsi-lcd.htm

[70] Wikipedia. (2021, 18. Mai*). Python (programming language)*. Hentet mai 18, 2021 fra https://en.wikipedia.org/wiki/Python_(programming_language)

[71] Wootton C. (2016) *Serial Peripheral Interface (SPI). In: Samsung ARTIK Reference. Apress, Berkeley, CA*. Hentet mai 18, 2021 fra https://doi.org/10.1007/978-1-4842-2322-2_21

[72] Young, E & Macdonald-Wallace, M. (2019. 02. april) *mfrc522*. Hentet mai 18, 2021 fra https://github.com/pimylifeup/MFRC522-python

A. Skjemategning



B. AddQueue.php

```
1.  <?php
2.      $tilkobling = mysqli_connect("localhost","Andy","Halla123","Backend");
3.
4.      if ($tilkobling -> connect_errno) {
5.                      echo "Failed to connect:" . $tilkobling -> connect_error;
6.      };
7.
8.      $Table = $_GET['ID'];
9.
10.     $queue_query = "SELECT QueueSolved FROM Queue WHERE QueueTableID = '$Table'";
11.     $queue_data = $tilkobling -> query($queue_query);
12.     $queue_data_array = mysqli_fetch_array($queue_data);
13.     if (isset($queue_data_array["QueueSolved"])) {
14.                     $reset_query = "UPDATE Queue SET QueueSolved = '0' WHERE QueueTableID = '$Table'"; //hvis
    bordet eksisterer. sett det i kø
15.                     $tilkobling -> query($reset_query);
16.     }else{
17.                     $insert_query = "INSERT INTO Queue(QueueTableID) VALUES ('$Table')"; //ellers sett inn bordet.
    all annen info kommer automatisk
18.                     $tilkobling -> query($insert_query);
19.     };
20.     header("Location: /InQueue.php/?ID=". $Table ."");
21. ?>
22.
```

C. ExitSessions.php

```
1.  <?php
2.      $tilkobling = mysqli_connect("localhost","Andy","Halla123","Backend");
3.
4.      if ($tilkobling -> connect_errno) {
5.                      echo "Failed to connect:" . $tilkobling -> connect_error;
6.      };
7.
8.      $table_query = "SELECT TableID FROM TableList WHERE TableSessionID IS NOT NULL"; //velg alle bord som
    er i sessions
9.      $table_data = $tilkobling -> query($table_query);
10.     while($rad = mysqli_fetch_array($table_data)){
11.                     $Plass = $rad["TableID"];
12.                     $remove_query = "UPDATE TableList SET TableSessionID = NULL, TableUserID = NULL WHERE
    TableID = '$Plass'"; //fjern sessionID og logg av alle fra alle bord med sessionID
13.                     $tilkobling -> query($remove_query);
14.                     $delete_query = "DELETE FROM Queue WHERE QueueTableID = '$Plass'"; //fjern all data i køen
15.                     $tilkobling -> query($delete_query);
16.     };
17.     $session_query = "SELECT SessionID FROM SessionList";
18.     $session_data = $tilkobling -> query($session_query);
19.     while($rad = mysqli_fetch_array($session_data)){
20.                     $ID = $rad["SessionID"];
21.                     $end_query = "DELETE FROM SessionList WHERE SessionID = '$ID'"; //fjern alle sessions
22.                     $tilkobling -> query($end_query);
23.     };
24.
25.     header("Location: /StudassMain.php"); //redirect
26. ?>
27.
28.
```

D. Admin.php

```
1.  <!DOCTYPE html>
2.  <html>
3.  <?php
4.    $Table = $_GET["ID"]; // Hvert bord vil ha et bordnummer som legges i URL via startopp program
5.  ?>
6.  <head>
7.  </head>
8.  <body>
9.  <strong>Skriv inn navn først. Deretter scan kort</strong><br>
10. <form method="post" action="/Reg.php">
11.   <input type="hidden" name="ID" value="<?php echo $Table ?>">
12.   <label>
13.     Navn <br>
14.     <input type="text" name="UserName">
15.   </label> <br>
16.   <label>
17.     Key <br>
18.     <input type="text" name="UserKey" id="input">
19.   </label> <br>
20.   <input type="submit" name="Submit"> <br>
21. </form>
22. <br><button onclick='Back()'>Tilbake</button>
23. </body>
24. </html>
25. <script>
26.   window.addEventListener("keydown", function (event) {
27.   if (event.defaultPrevented) {
28.     return; // Do nothing if the event was already processed
29.   }
30.     switch (event.key) {
31.       case "Control":
32.         document.getElementById('input').focus();
33.         break;
34.       default:
35.         return; // Quit when this doesn't handle the key event.
36.     }
37.   // Cancel the default action to avoid it being handled twice
38.   event.preventDefault();
39. }, true);
40. // the last option dispatches the event to the listener first,
41. // then dispatches event to window
42.   function Back(){
43.     location.href = "/Login.php/?ID=<?php echo $Table; ?>";
44.   };
45. </script>
46.
```

E. InQueue.php

```
<?php
$Table = $_GET['ID'];
$tilkobling = mysqli_connect("localhost","Andy","Halla123","Backend");

if ($tilkobling -> connect_errno) {
    echo "Failed to connect:" . $tilkobling -> connect_error;
};

$user_query = "SELECT TableUserID FROM TableList WHERE TableID = '$Table'";
$user_data = $tilkobling -> query($user_query);
$user_data_array = mysqli_fetch_array($user_data);
$UserID = $user_data_array["TableUserID"];
$name_query = "SELECT UserName FROM UserCred WHERE UserID = '$UserID'";
$name_data = $tilkobling -> query($name_query);
$name_data_array = mysqli_fetch_array($name_data);
$Name = $name_data_array["UserName"];
$queueCheck = "SELECT QueueSolved FROM Queue WHERE QueueTableID = '$Table'";
$queueCheckData = $tilkobling -> query($queueCheck);
$row = mysqli_fetch_array($queueCheckData);
$Check = $row["QueueSolved"];
if ($Check == 1) {
  header("Location: /main.php/?ID=". $Table .""); // sender deg tilbake til main om studass fjernet deg fra køen
}elseif (!isset($UserID)) {
  header("Location: /Login.php/?ID=". $Table .""); //kaster brukeren ut om studass logger alle av
};
$queueSql = "SELECT QueueSessionID, QueueTableID FROM Queue WHERE QueueSolved = '0' ORDER BY
QueueSessionID ASC, QueueTimeStamp ASC";
$queueData = $tilkobling -> query($queueSql);
?>
<!DOCTYPE html>
<script>
function LogOut(){
    if(confirm("Logg ut?")){
                location.href = "/LogOff.php/?ID=<?php echo $Table; ?>";
    };
}
function startTime() { //klokkefunksjon, ikke bygd selv
  var today = new Date();
  var h = today.getHours();
  var m = today.getMinutes();
  m = checkTime(m);
  document.getElementById('clock').innerHTML =
  h + ":" + m;
  var t = setTimeout(startTime, 500);
}
function checkTime(i) {
  if (i < 10) {i = "0" + i};
  return i;
}
</script>
<style>
<?php include'RasPi.css'; ?>
</style>
<html>
<body onload="startTime()">
<element class ="Header">
    <img src="../Norge.png" height="120px">
    <element class="info1">
                Time: <div width = "40px"id="clock"></div> <br>
                BordNummer: <?php echo $Table; ?>
    </element>
    <element class="info2">
                Navn: <?php echo $Name; ?>
    </element>
    <element class="Logout">
     <input type="image" src="../LoggAv.png" id="1" onclick="LogOut()" height="120px" />
```

```
66.    </element>
67.  </element>
68.  <element class="Interact">
69.      <element class="Interact1">
70.                      <?php
71.                      $count = 0;
72.                      while($rad = mysqli_fetch_array($queueData)){
73.                                      $QueueTableID = $rad["QueueTableID"];
74.                                      if ($QueueTableID != $Table) { //skriven plassen din i kø. bruker index i tables så 0 --> 1
     i køen og dermed øke med 1
75.                                                      $count++;
76.                                      }else{
77.                                                      echo "Din plass i køen er: ". ++$count ."";
78.                                      };
79.                      }
80.                      ?>
81.      </element>
82.      <form method="GET"  action="/RemoveQueue.php">
83.                      <input type="hidden" name="ID" value="<?php echo $Table ?>">
84.        <input class="Interact2" id="0" type="submit" value="Trekk deg fra kø" />
85.      </form>
86.  </element>
87.  </body>
88.  </html>
89.  <script type="text/javascript">
90.  var loc = 0; // 0 = kø knapp 1 = logg av knapp
91.  document.getElementById('0').focus();
92.  window.addEventListener("keydown", function (event) {
93.    if (event.defaultPrevented) {
94.      return; // Do nothing if the event was already processed
95.    }
96.     switch (event.key) {
97.       case "ArrowDown":
98.         document.getElementById('0').focus();
99.         loc = 0;
100.        break;
101.      case "ArrowUp":
102.        document.getElementById('1').focus();
103.        loc = 1;
104.        break;
105.      default:
106.        return; // Quit when this doesn't handle the key event.
107.    }
108.
109.   // Cancel the default action to avoid it being handled twice
110.   event.preventDefault();
111. }, true);
112. // the last option dispatches the event to the listener first,
113. // then dispatches event to window
114. setTimeout(function(){
115.    window.location.reload(1); //Refresher siden hvert 15 sekund
116. }, 15000);
117. </script>
118.
```

F. Login.php

```
1.  <!DOCTYPE html>
2.  <?php
3.      $Table = $_GET["ID"]; // Hvert bord vil ha ett bordnummer som legges i URL via startopp program
4.  ?>
5.  <html>
6.  <style>
7.  <?php include'RasPi.css'; ?>
8.  </style>
9.  <head>
10. </head>
11. <body>
12. <form method="post" action="/LoginAction.php">
13.     <input type="hidden" value = "<?php echo $Table; ?>" name="TableID">
14.   <label>
15.     Key <br>
16.     <input type="password" name="UserKey" id="input">
17.   </label> <br>
18.   <label>
19.   <input type="submit" value="Logg Inn" name="LogVal"> <br>
20. </form>
21. </body>
22. </html>
23. <script type="text/javascript">
24. window.addEventListener("keydown", function (event) {
25.   if (event.defaultPrevented) {
26.     return; // Do nothing if the event was already processed
27.   }
28.     switch (event.key) {
29.       case "Control":
30.         document.getElementById('input').focus();
31.         break;
32.       default:
33.         return; // Quit when this doesn't handle the key event.
34.     }
35.   // Cancel the default action to avoid it being handled twice
36.   event.preventDefault();
37. }, true);
38. // the last option dispatches the event to the listener first,
39. // then dispatches event to window
40. </script>
41.
```

G. LogOff.php

```
1.  <?php
2.      $tilkobling = mysqli_connect("localhost","Andy","Halla123","Backend");
3.
4.      if ($tilkobling -> connect_errno) {
5.                      echo "Failed to connect:" . $tilkobling -> connect_error;
6.      };
7.
8.      $Table = $_GET['ID'];
9.
10.     $user_query = "UPDATE TableList SET TableUserID = NULL WHERE TableID = '$Table'";
11.     $user_data = $tilkobling -> query($user_query);
12.     $remove_query = "UPDATE Queue SET QueueSolved = '1' WHERE QueueTableID = '$Table'";
13.     $tilkobling -> query($remove_query);
14.     header("Location: /Login.php/?ID=". $Table ."");
15. ?>
16.
```

H. LoginAction.php

```
1.  <?php
2.      $tilkobling = mysqli_connect("localhost","Andy","Halla123","Backend");
3.
4.      if ($tilkobling -> connect_errno) {
5.                      echo "Failed to connect:" . $tilkobling -> connect_error;
6.      };
7.
8.      $Table = $_POST["TableID"];
9.      $Key = $_POST["UserKey"];
10.
11.     $user_query = "SELECT UserLevel, UserName, UserID FROM UserCred WHERE UserKey = '$Key'"; //sjekk at
    brukeren eksisterer og hent id
12.     $user_data = $tilkobling -> query($user_query);
13.     $user_data_array = mysqli_fetch_array($user_data);
14.     $userID = $user_data_array["UserID"];
15.
16.     $table_query = "SELECT TableUserID FROM TableList WHERE TableID = '$Table'"; //sjekk at det ikke er noen på
    det bordet allerede
17.     $table_data = $tilkobling -> query($table_query);
18.     $table_data_array = mysqli_fetch_array($table_data);
19.     $table_userID = $table_data_array["TableUserID"];
20.
21.
22.     if (isset($table_userID)) {
23.                     echo "That table is occupied, or table number:" . $Table . " does not exist";
24.                     echo "<button onclick='Back()'>Tilbake</button>";
25.     }else{
26.                     if ($user_data_array["UserLevel"] == 3) {
27.                                     header("Location: /Admin.php/?ID=". $Table .."");
28.                     }elseif ($user_data_array["UserLevel"] > 0 AND isset($user_data_array["UserName"])) { // kan evt
    byttes ut med ENUM
29.                                     $data = "UPDATE TableList SET TableUserID = $userID WHERE TableID = '$Table'";
    //sett deg på bordet
30.                                     $tilkobling -> query($data);
31.                                     header("Location: /main.php/?ID=". $Table .."");
32.                     }else{
33.                                     echo "No user with that key exists";
34.                                     echo "<button onclick='Back()'>Tilbake</button>";
35.                     };
36.     };
37. ?>
38. <script type="text/javascript">
39.     function Back(){
40.                     location.href = "/Login.php/?ID=<?php echo $Table; ?>";
41.     };
42. </script>
43.
```

I. main.php

```
<?php
$Table = $_GET['ID'];
$tilkobling = mysqli_connect("localhost","Andy","Halla123","Backend");

if ($tilkobling -> connect_errno) {
    echo "Failed to connect:" . $tilkobling -> connect_error;
};

$user_query = "SELECT TableUserID FROM TableList WHERE TableID = '$Table'";
$user_data = $tilkobling -> query($user_query);
$user_data_array = mysqli_fetch_array($user_data);
$UserID = $user_data_array["TableUserID"];
$name_query = "SELECT UserName FROM UserCred WHERE UserID = '$UserID'"; //Henter info om bruker.
$name_data = $tilkobling -> query($name_query);
$name_data_array = mysqli_fetch_array($name_data);
$Name = $name_data_array["UserName"];
if (!isset($UserID)) {
  header("Location: /Login.php/?ID=". $Table .""); //kaster brukeren ut om studass logger alle av
};
?>
<!DOCTYPE html>
<script>
function LogOut(){
    if(confirm("Logg ut?")){
                location.href = "/LogOff.php/?ID=<?php echo $Table; ?>"; //logg av skript som bekrefter
    };
}
function startTime() { //klokkefunksjon, ikke bygd selv
  var today = new Date();
  var h = today.getHours();
  var m = today.getMinutes();
  m = checkTime(m);
  document.getElementById('clock').innerHTML =
  h + ":" + m;
  var t = setTimeout(startTime, 500);
}
function checkTime(i) {
  if (i < 10) {i = "0" + i};
  return i;
}
</script>
<style>
<?php include'RasPi.css'; ?>
</style>
<html>
<body onload="startTime()">
<element class ="Header">
    <img src="../Norge.png" height="120px">
    <element class="info1">
                Time: <div width = "40px"id="clock"></div> <br>
                BordNummer: <?php echo $Table; ?>
    </element>
    <element class="info2">
                Navn: <?php echo $Name; ?>
    </element>
    <element class="Logout">
     <input type="image" src="../LoggAv.png" id="1" onclick="LogOut()" height="120px" />
  </element>
</element>
<element class="Interact">
    <element class="Interact1"></element>
    <form method="GET" action="/AddQueue.php">
                <input type="hidden" name="ID" value="<?php echo $Table ?>">
      <input class="Interact2" id="0" type="submit" value="Legg til i kø" />
    </form>
</element>
```

```
</body>
</html>
<script type="text/javascript">
var loc = 0; // 0 = kø knapp 1 = logg av knapp
document.getElementById('0').focus();
window.addEventListener("keydown", function (event) {
  if (event.defaultPrevented) {
    return; // Do nothing if the event was already processed
  }
    switch (event.key) {
      case "ArrowDown":
        document.getElementById('0').focus();
        loc = 0;
        break;
      case "ArrowUp":
        document.getElementById('1').focus();
        loc = 1;
        break;
      default:
        return; // Quit when this doesn't handle the key event.
    }

  // Cancel the default action to avoid it being handled twice
  event.preventDefault();
}, true);
// the last option dispatches the event to the listener first,
// then dispatches event to window
setTimeout(function(){
   window.location.reload(1); //Refresher siden hvert minutt
}, 60000);
</script>

```

J. RasPi.css

```
body{
    display: grid;
    height: 460px;
    width: 780px;
    grid-template-rows: 1fr 4fr;
    border: black 1px solid;
    grid-template-areas: "Header" "Interact";

}

html,body {overscroll-behavior: none;}

.Header{
    grid-area: "Header";
    display: grid;
    grid-template-columns: 2fr 3fr 3fr 1fr;
    grid-template-areas: "Language info1 info2 Logout";
}

.Interact{
    grid-area: "Interact";
    display: grid;
    grid-template-columns: 1fr 1fr;
    grid-template-areas: "Interact1 Interact2"
}

.Interact1{
    grid-area: "Interact1";
}
```

```
30.
31.  .Interact2{
32.      grid-area: "Interact2";
33.      height: 336px;
34.      width: 390px;
35.  }
36.
37.  .Language{
38.      grid-area: "Language";
39.  }
40.
41.  .info1{
42.      grid-area: "info1";
43.  }
44.
45.  .info2{
46.      grid-area: "info2";
47.  }
48.
49.  .Logout{
50.      grid-area: "Logout";
51.  }
52.
```

K. Reg.php

```
1.   <?php
2.       $tilkobling = mysqli_connect("localhost","Andy","Halla123","Backend");
3.
4.       if ($tilkobling -> connect_errno) {
5.                       echo "Failed to connect:" . $tilkobling -> connect_error;
6.       };
7.
8.       $Key = $_POST["UserKey"];
9.       $Name = $_POST["UserName"];
10.      $Table = $_POST["ID"];
11.
12.      $ver = "SELECT UserKey FROM UserCred WHERE UserKey = '$Key'";
13.      $ver_data = $tilkobling -> query($ver);
14.      $data_array = mysqli_fetch_array($ver_data);
15.      $ver_info = $data_array["UserKey"];
16.      if(isset($ver_info)){
17.                      echo "Dette kortet er allerede registrert";
18.      }elseif ($Name != "" AND $Key != "") {
19.                      $data = "INSERT INTO UserCred (UserKey,UserLevel,UserName) VALUES ('$Key','1','$Name')";
     //legg inn bruker som student nivå.
20.                      $tilkobling -> query($data);
21.                      echo "Følgende navn lagt inn:" . $Name ."Som student";
22.      }else{
23.                      echo "Ingenting ble lagt inn. prøv på nytt";
24.      };
25.      echo "<br><button onclick='Back()'>Tilbake</button>";
26.  ?>
27.  <script type="text/javascript">
28.      function Back(){
29.                      location.href = "/Admin.php/?ID=<?php echo $Table; ?>";
30.      };
31.  </script>
32.
```

L. RemoveQueue

```
1.  <?php
2.      $tilkobling = mysqli_connect("localhost","Andy","Halla123","Backend");
3.
4.      if ($tilkobling -> connect_errno) {
5.                      echo "Failed to connect:" . $tilkobling -> connect_error;
6.      };
7.
8.      $Table = $_GET['ID'];
9.
10.     $remove_query = "UPDATE Queue SET QueueSolved = '1' WHERE QueueTableID = '$Table'"; //setter
    queuesolved = 1 slik at den fjernes fra studass sin GUI
11.     $tilkobling -> query($remove_query);
12.     header("Location: /main.php/?ID=". $Table ."");
13. ?>
14.
```

M. SessionInsert.php

```
1.  <?php
2.      $tilkobling = mysqli_connect("localhost","Andy","Halla123","Backend");
3.
4.      if ($tilkobling -> connect_errno) {
5.                      echo "Failed to connect:" . $tilkobling -> connect_error;
6.      };
7.
8.      $Plasser = $_POST["PlassValg"];
9.      $sessionID = $_POST["sessionID"];
10.     $select_query = "SELECT TableID FROM TableList WHERE TableSessionID = '$sessionID'";
11.     $select_data = $tilkobling -> query($select_query);
12.     while($rad = mysqli_fetch_array($select_data)){
13.                     $Plass = $rad["TableID"];
14.                     $remove_query = "UPDATE TableList SET TableSessionID = NULL WHERE TableID = '$Plass'";
    //fjerner alle tidligere bord med den session
15.                     $tilkobling -> query($remove_query);
16.     };
17.     $update_query = "UPDATE TableList SET TableSessionID = '$sessionID' WHERE TableID in ($Plasser)"; //legger
    til de nye bordene
18.     $tilkobling -> query($update_query);
19.
20.     header("Location: /StudassMain.php");
21. ?>
22.
```

N. Studass.css

```
1.  .mainBody{
2.  display: grid;
3.      min-width: 1366px;
4.      min-height: 700px;
5.      grid-template-columns: 3fr 1fr 1fr 1fr 1fr;
6.      grid-template-rows: 1fr 3fr;
7.      border: black 1px solid;
8.  }
9.
10. .QueueClass{
11.     grid-column-start: 1;
12.     grid-column-end: 2;
13.     grid-row-start: 1;
14.     grid-row-end: 4;
```

```
15.  }
16.
17.  .SeatMap{
18.      grid-column-start: 2;
19.      grid-column-end: 6;
20.      grid-row-start: 2;
21.      grid-row-end: 3;
22.  }
23.
24.  .LogOut{
25.      grid-column-start: 5;
26.      grid-column-end: 6;
27.      grid-row-start: 1;
28.      grid-row-end: 2;
29.  }
30.
31.  .ChangeSession{
32.      grid-column-start: 4;
33.      grid-column-end: 5;
34.      grid-row-start: 1;
35.      grid-row-end: 2;
36.  }
37.
38.  .accountForm{
39.      grid-column-start: 2;
40.      grid-column-end: 3;
41.      grid-row-start: 1;
42.      grid-row-end: 2;
43.      display: none;
44.      border: black 1px round;
45.  }
46.
47.  img {
48.      width: 867px;
49.      height: 520px;
50.      object-fit: fill;
51.  }
52.
```

O. StudassLoginAction

```
1.   <?php
2.       $tilkobling = mysqli_connect("localhost","Andy","Halla123","Backend");
3.
4.       if ($tilkobling -> connect_errno) {
5.                       echo "Failed to connect:" . $tilkobling -> connect_error;
6.       };
7.
8.       $UserName = $_POST["UserName"];
9.       $Password = $_POST["UserPwd"];
10.
11.      $user_query = "SELECT UserLevel, UserID FROM UserCred WHERE UserName = '$UserName' AND UserKey =
     '$Password'";
12.      $user_data = $tilkobling -> query($user_query);
13.      $user_data_array = mysqli_fetch_array($user_data);
14.      $userID = $user_data_array["UserID"];
15.
16.      $session_query = "SELECT SessionID FROM SessionList WHERE SessionUserID = '$userID'";
17.      $session_data = $tilkobling -> query($session_query);
18.      $session_data_array = mysqli_fetch_array($session_data);
19.      $sessionID = $session_data_array["SessionID"];
20.
21.
22.      if (isset($sessionID)) {
```

```
23.                     header("Location: /TableReg.php/?ID=". $sessionID .""); //hvis man har session id send de til
    bordregistrering
24.     }else{
25.                     if ($user_data_array["UserLevel"] > 1) { //kan evt byttes ut med ENUM
26.                             $data = "INSERT INTO SessionList(SessionUserID) VALUES ($userID)";
27.                             $tilkobling -> query($data);
28.                             $session_query = "SELECT SessionID FROM SessionList WHERE SessionUserID =
    '$userID'"; //ellers lag session og send de videre
29.                             $session_data = $tilkobling -> query($session_query);
30.                             $session_data_array = mysqli_fetch_array($session_data);
31.                             $sessionID = $session_data_array["SessionID"];
32.                             header("Location: /TableReg.php/?ID=". $sessionID ."");
33.                     }else{
34.                             echo "No user with that key exists";
35.                             echo "<button onclick='Back()'>Tilbake</button>"; //brukeren har ikke tilgang.
36.                     };
37.     };
38. ?>
39. <script type="text/javascript">
40.     function Back(){
41.                     location.href = "/StudassMain.php";
42.     };
43. </script>
44.
```

P. StudassMain.php

```
1.  <?php
2.      $tilkobling = mysqli_connect("localhost","Andy","Halla123","Backend");
3.
4.      if ($tilkobling -> connect_errno) {
5.          echo "Failed to connect:" . $tilkobling -> connect_error;
6.      };
7.
8.      $FjernID = $_GET['Fjern'];
9.      $FjernSessionID = $_GET['FjernSession'];
10.
11.     if(isset($FjernID)){ //Henter og setter slik at Hjelp på bordet er gitt.
12.         $fjernSql = "UPDATE Queue SET QueueSolved = '1' WHERE QueueTableID = '$FjernID' AND QueueSessionID
    = '$FjernSessionID'";
13.         $tilkobling->query($fjernSql);
14.         echo(mysqli_error($tilkobling));
15.     };
16.     $queueSql = "SELECT QueueSessionID, QueueTableID FROM Queue WHERE QueueSolved = '0' ORDER BY
    QueueSessionID ASC, QueueTimeStamp ASC"; //Henter alle bord som trenger hjelp og sorterer de med hensyn til
    Session og Tid
17.
18.     $queueData = $tilkobling -> query($queueSql);
19. ?>
20. <!DOCTYPE html>
21. <html>
22. <style>
23. <?php include'Studass.css'; ?>
24. </style>
25. <body class="mainBody">
26. <table class='QueueClass'>
27.     <tr>
28.     <th>Kø SessionID</th>
29.     <th>Kø Nummer</th>
30.     <th>Bord Nummer</th>
31.     <th>Fjern</th>
32.     </tr>
33. <?php
34.     $QueueNumber = 1;
35.     $LastSessionID = "";
36.     while($rad = mysqli_fetch_array($queueData)){ //Skriver ut dataen som hentes ettehvert som den blir sendt. slipper å
    bruke loops
37.     $QueueSessionID = $rad["QueueSessionID"];
```

```
38.     $QueueTableID = $rad["QueueTableID"];
39.         echo "<tr>";
40.         echo "<td>";
41.         echo $QueueSessionID;
42.         echo "</td>";
43.         echo "<td>";
44.         if ($LastSessionID == $QueueSessionID OR $LastSessionID == ""){
45.             echo $QueueNumber;
46.             $QueueNumber++;
47.         }else{
48.         $QueueNumber = 1;
49.         echo $QueueNumber;
50.         $QueueNumber++;
51.        };
52.        $LastSessionID = $QueueSessionID;
53.         echo "</td>";
54.         echo "<td>";
55.         echo $QueueTableID;
56.         echo "</td>";
57.         echo "<td>";
58.         echo "<a href=/StudassMain.php/?Fjern=" . $QueueTableID . "&FjernSession=". $QueueSessionID .">Fjern</a>";
59.         echo "</td>";
60.         echo "</tr>";
61.     };
62. ?>
63. </table>
64. <div class="accountForm" id="accountVer">
65. <form method="post" action="/StudassLoginAction.php">
66.     <label><strong>Logg inn for å skape eller endre sessions</strong></label><br>
67.     <label>
68.     Brukernavn
69.     <input type="text" name="UserName">
70.   </label>
71.   <label>
72.     Passord
73.     <input type="password" name="UserPwd">
74.   </label>
75.   <label>
76.   <input type="submit" value="Logg Inn">
77.   </label>
78. </form>
79. <button onclick="goBack();">Lukk Login</button>
80. </div>
81. <button class="ChangeSession" onclick="ChangeSession()">Endre Session</button>
82. <button class="LogOut" onclick="LogOut()">Avslutt Sessions</button>
83. <div class="SeatMap">
84. <img src="../Bord.png">
85. </div>
86. </body>
87. </html>
88. <script type="text/javascript">
89.     function ChangeSession(){
90.         document.getElementById("accountVer").style.display = "block";
91.     };
92.     function goBack(){
93.         document.getElementById("accountVer").style.display = "none";
94.     };
95.     function LogOut(){
96.         if (confirm("Avslutte? OBS: Dette vil slette alle aktive sessions og den aktive køen. Ønsker du låse opp bord, bruk
    Endre Session")) {
97.             location.href = "/ExitSessions.php";
98.         };
99.     };
100. setTimeout(function(){
101.    location.href="/StudassMain.php"; //Refresher siden hvert 30. sekund
102. }, 30000);
103. </script>
104.
```

Q. TableReg.php

```
1.  <!DOCTYPE html>
2.  <html>
3.  <style>
4.  <?php
5.    include'Studass.css';
6.    $tilkobling = mysqli_connect("localhost","Andy","Halla123","Backend");
7.
8.    if ($tilkobling -> connect_errno) {
9.      echo "Failed to Connect:" . $tilkobling -> connect_errno;
10.   };
11.   $sessionID = $_GET["ID"];
12. ?>
13. </style>
14. <head>
15.   <title></title>
16. </head>
17. <body onload="loadVals()">
18. <form method="POST" action="/SessionInsert.php">
19.   <div>fagkode?</div>
20.   <div>start/slutt?</div>
21.   <input type="hidden" name="sessionID" <?php echo "value = '". $sessionID ."'"; ?>>
22.   <input type="text" id="PlassValgBoks" name="PlassValg"><br>
23.   <input type="submit" value="Velg Seter" > <br>
24. </form>
25. <div class="Selector">
26. <input type="button" class="Sel" id="37" value="37">
27. <input type="button" class="Sel" id="38" value="38">
28. ____________________________________________
29. <input type="button" class="Sel" id="39" value="39">
30. <input type="button" class="Sel" id="40" value="40"><br><br>
31.
32. <input type="button" class="Sel" id="36" value="36">
33. <input type="button" class="Sel" id="35" value="35">
34. <input type="button" class="Sel" id="34" value="34">
35. <input type="button" class="Sel" id="33" value="33">
36. _____________________________
37. <input type="button" class="Sel" id="45" value="45">
38. <input type="button" class="Sel" id="44" value="44">
39. <input type="button" class="Sel" id="43" value="43">
40. <input type="button" class="Sel" id="42" value="42">
41. <input type="button" class="Sel" id="41" value="41"><br>
42.
43. <input type="button" class="Sel" id="29" value="29">
44. <input type="button" class="Sel" id="30" value="30">
45. <input type="button" class="Sel" id="31" value="31">
46. <input type="button" class="Sel" id="32" value="32">
47. _____________________________
48. <input type="button" class="Sel" id="46" value="46">
49. <input type="button" class="Sel" id="47" value="47">
50. <input type="button" class="Sel" id="48" value="48">
51. <input type="button" class="Sel" id="49" value="49">
52. <input type="button" class="Sel" id="50" value="50"><br><br>
53.
54. <input type="button" class="Sel" id="28" value="28">
55. <input type="button" class="Sel" id="27" value="27">
56. <input type="button" class="Sel" id="26" value="26">
57. <input type="button" class="Sel" id="25" value="25">
58. _____________________________
59. <input type="button" class="Sel" id="56" value="56">
60. <input type="button" class="Sel" id="55" value="55">
61. <input type="button" class="Sel" id="54" value="54">
62. <input type="button" class="Sel" id="53" value="53">
63. <input type="button" class="Sel" id="52" value="52">
64. <input type="button" class="Sel" id="51" value="51"><br>
65.
66. <input type="button" class="Sel" id="21" value="21">
```

```
67.  <input type="button" class="Sel" id="22" value="22">
68.  <input type="button" class="Sel" id="23" value="23">
69.  <input type="button" class="Sel" id="24" value="24">
70.  _____
71.  <input type="button" class="Sel" id="57" value="57">
72.  <input type="button" class="Sel" id="58" value="58">
73.  <input type="button" class="Sel" id="59" value="59">
74.  <input type="button" class="Sel" id="60" value="60">
75.  <input type="button" class="Sel" id="61" value="61">
76.  <input type="button" class="Sel" id="62" value="62"><br><br>
77.
78.  <input type="button" class="Sel" id="20" value="20">
79.  <input type="button" class="Sel" id="19" value="19">
80.  <input type="button" class="Sel" id="18" value="18">
81.  <input type="button" class="Sel" id="17" value="17">
82.  _____
83.  <input type="button" class="Sel" id="68" value="68">
84.  <input type="button" class="Sel" id="67" value="67">
85.  <input type="button" class="Sel" id="66" value="66">
86.  <input type="button" class="Sel" id="65" value="65">
87.  <input type="button" class="Sel" id="64" value="64">
88.  <input type="button" class="Sel" id="63" value="63"><br>
89.
90.  <input type="button" class="Sel" id="13" value="13">
91.  <input type="button" class="Sel" id="14" value="14">
92.  <input type="button" class="Sel" id="15" value="15">
93.  <input type="button" class="Sel" id="16" value="16">
94.  _____
95.  <input type="button" class="Sel" id="69" value="69">
96.  <input type="button" class="Sel" id="70" value="70">
97.  <input type="button" class="Sel" id="71" value="71">
98.  <input type="button" class="Sel" id="72" value="72">
99.  <input type="button" class="Sel" id="73" value="73">
100. <input type="button" class="Sel" id="74" value="74"><br><br>
101.
102. <input type="button" class="Sel" id="12" value="12">
103. <input type="button" class="Sel" id="11" value="11">
104. <input type="button" class="Sel" id="10" value="10">
105. <input type="button" class="Sel" id="9" value="9">
106. <input type="button" class="Sel" id="8" value="8">
107. _____
108. <input type="button" class="Sel" id="75" value="75">
109. <input type="button" class="Sel" id="76" value="76">
110. <input type="button" class="Sel" id="77" value="77">
111. <input type="button" class="Sel" id="78" value="78">
112. <input type="button" class="Sel" id="79" value="79">
113. <input type="button" class="Sel" id="80" value="80"><br>
114.
115. <input type="button" class="Sel" id="3" value="3">
116. <input type="button" class="Sel" id="4" value="4">
117. <input type="button" class="Sel" id="5" value="5">
118. <input type="button" class="Sel" id="6" value="6">
119. <input type="button" class="Sel" id="7" value="7">
120. _____
121. <input type="button" class="Sel" id="81" value="81">
122. <input type="button" class="Sel" id="82" value="82">
123. <input type="button" class="Sel" id="83" value="83">
124. <input type="button" class="Sel" id="84" value="84">
125. <input type="button" class="Sel" id="85" value="85">
126. <input type="button" class="Sel" id="86" value="86"><br><br>
127.
128. <input type="button" class="Sel" id="2" value="2">
129. <input type="button" class="Sel" id="1" value="1">
130. _____
131. <input type="button" class="Sel" id="88" value="88">
132. <input type="button" class="Sel" id="87" value="87"><br>
133. </div>
134. </body>
135. <?php
136.     $SeatSQL = "SELECT TableID, TableSessionID FROM TableList WHERE TableSessionID IS NOT NULL";
      //Henter listen over bord som allerede er med i en session
```

```
137.    $data = $tilkobling -> query($SeatSQL);
138.    echo "<script type='text/javascript'>";
139.    echo "var Vals = [];";
140.    while($rad = mysqli_fetch_array($data)){
141.    $TableID = $rad["TableID"];
142.    $tableSessionID = $rad["TableSessionID"];
143.    if ($tableSessionID != $sessionID) {
144.     echo "document.getElementById('".$TableID."').disabled = true;"; //Skrur av knappenen for bord som allerede er
      med i session
145.    }else{
146.      echo "Vals.push(". $TableID ."');";
147.    };
148.   };
149.   echo "</script>";
150.
151. ?>
152. <script type="text/javascript">
153.   // Systemet er lagd slik at man kan bare legge til flere knapper hvor verdiene er i samsvar med setenummereringen i
      databasen og at de har class = "Sel" så vil det fungere.
154.   var Plass = document.getElementsByClassName('Sel'); // Finner alle knappene som skal brukes for valg av seter
155.   var PlassSel = []; // Lager en tom rad for å midlertidelig lagre dataen
156.   for (let i = 0, l = Plass.length; i <= l; i++) { // Lager plass i raden og styringsfunksjon ValgSete() for hver knapp
157.     PlassSel[i] = 0;
158.       Plass[i].addEventListener('click', function VelgSete() {
159.     Sete = Plass[i].id;
160.     if (PlassSel[Sete] == 0 ){
161.       PlassSel[Sete] = 1;
162.       Plass[i].style.backgroundColor = "#7FFF00"; // ved trykk endres fargen og korrisponderende index i PlassSel blir
      endret til 1
163.
164.     } else if (PlassSel[Sete] == 1) {
165.       PlassSel[Sete] = 0;
166.       Plass[i].style.backgroundColor = "#FFFFFF";
167.     };
168.     let LocString = "";
169.       var LocPlassSel = PlassSel.slice(); // lager lokale verdier for å ikke ødelegge brukerens valg
170.       var CurrentIndex = PlassSel.indexOf(1);
171.       while(CurrentIndex != -1){
172.         LocString = LocString + CurrentIndex;
173.         LocPlassSel[CurrentIndex] = 0; // bygger string som skal inn i php form og legger inn dynamisk
174.         CurrentIndex = LocPlassSel.indexOf(1);
175.         if (CurrentIndex != -1) { //avslutter når det ikke lenger er noen flere 1 i PlassSel
176.           LocString = LocString + ",";
177.         };
178.       };
179.       document.getElementById('PlassValgBoks').value = LocString;
180.       });
181.     };
182.     function loadVals(){
183.       for (var i = Vals.length - 1; i >= 0; i--) { // skal preloade alle bord som du allerede har registert før på deg.
184.         document.getElementById(Vals[i]).click();
185.       };
186.     };
187. </script>
188. </html
```

R. read.py

```
1.  import RPi.GPIO as GPIO
2.  import keyboard
3.
4.  from mfrc522 import SimpleMFRC522
5.  from time import sleep
6.
7.
8.  GPIO.setwarnings(False)
9.  reader = SimpleMFRC522()
10.
11. rfid_Y = True;
12.
13. while rfid_Y:
14.       try:
15.
16.
17.           id, text = reader.read()
18.           keyboard.send('ctrl')
19.           keyboard.write(str(id))
20.           keyboard.send('enter')
21.
22.           sleep(5)
23.
24.       except:
25.             GPIO.cleanup()
26.
```

S. buttons_to_keyboard.py

```
1.  import keyboard
2.  from gpiozero import Button
3.  from time import sleep
4.
5.  delay_after_buttonpress = 0.5
6.
7.  left_button = Button(17)
8.  right_button = Button(27)
9.  up_button = Button(22)
10. down_button = Button(5)
11. confirm_button = Button(6)
12.
13. #event loop
14. while True:
15.     if left_button.is_pressed:
16.         keyboard.send("left")
17.         sleep(delay_after_buttonpress)
18.
19.     elif right_button.is_pressed:
20.         keyboard.send("right")
21.         sleep(delay_after_buttonpress)
22.
23.     elif up_button.is_pressed:
24.         keyboard.send("up")
25.         sleep(delay_after_buttonpress)
26.
27.     elif down_button.is_pressed:
28.         keyboard.send("down")
29.         sleep(delay_after_buttonpress)
30.
31.     elif confirm_button.is_pressed:
32.         keyboard.send("enter")
33.         sleep(delay_after_buttonpress)
```

T. Komponentkostnader

RASPBERRY PI 4B/2GB

Datablad

Digi-Keys delenummer	1690-RASPBERRYPI4B/2GB-ND
Produsent	Raspberry Pi
Produsentens delenummer	RASPBERRY PI 4B/2GB
Beskrivelse	RASPBERRY PI 4 MODEL B 2GB SDRAM
Kundereferanse	Kundereferanse

Pris og anskaffelse

Forespør om varsel når varen er på lager

ANTALL

1

Legg til i handlevognen

Alle prisene er i EUR.

PRISREDUKSJON	ENHETSPRIS	UTVIDET PRIS
1	28,95000	€28,95

ENHETSPRIS

€28,95000 Eks. merverdiavgift (VAT)

€36,18750 Inkl. merverdiavgift (VAT)

Dokumenter og media

Datablad	Raspberry Pi 4 Model B Brief Raspberry Pi 4 Model B Datasheet
Utvalgt produkt	Trion® T20 MIPI D-PHY/CSI-2 Development Kit
Prosjekter	Make a Pi Trash Classifier with Machine Learning and Lobe

RFID-RC522 RF IC Card Sensor Modul

Antall tilgjengelig: 44

Pris: Kr 39,00
Eks. moms: Kr 31,20

3 eller mer Kr 34,00

Antall: 1 Kjøp - ELLER - Legg i ønskeliste

PCB

Charge Details

Engineering fee	$8.00
Surface Finish	$5.00
Testing	$36.20
Film	$5.10
Board	$155.90
Confirm Production file	$0.54

Build Time

PCB: 5-6 days $0.00

Calculated Price $210.74

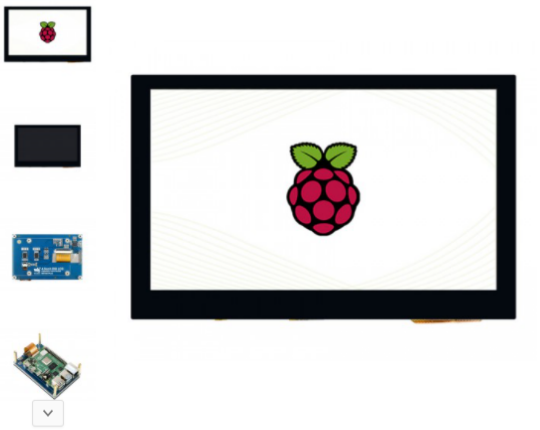
Additional charges may apply for special cases

Weight 7.76kg

SAVE TO CART

Shipping Estimate $71.64

FedEx International Priority 6-8 business days



4.3inch Capacitive Touch Display for Raspberry Pi, DSI Interface, 800×480

SKU: 16239
Part Number: 4.3inch DSI LCD
Brand: Waveshare

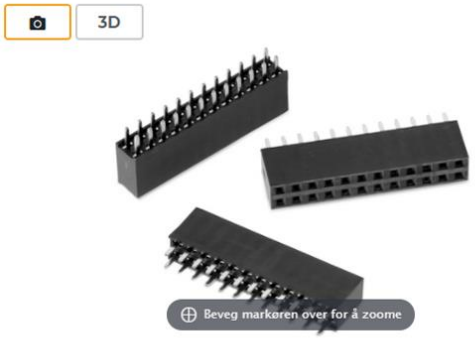
$39.99 1 ADD TO CART

$39.39	10+
$39.09	50+
$38.97	100+

Related Products:

Bildet er kun til illustrasjon. Se produktbeskrivelsen.

Legg til i handleliste Sammenlign



Bildet er kun til illustrasjon. Se produktbeskrivelsen.

Legg til i handleliste Sammenlign

Bildet er kun til illustrasjon. Se produktbeskrivelsen.

Legg til i handleliste Sammenlign

1469 på lager for omgående utsendelse

Pris per stykk Produktserie >

NOK 1,09 (ekskl. MVA)

25 +	NOK 1,09
50 +	NOK 0,8466
250 +	NOK 0,6559
1000 +	NOK 0,3488
5000 +	NOK 0,3152
10000 +	NOK 0,2611

– 25 + Legg i handlevogn

25 Min.bestilling

394 på lager for omgående utsendelse

Pris per stykk Produktserie >

NOK 16,00 (ekskl. MVA)

1 +	NOK 16,00	
25 +	NOK 14,60	Kvantumsrabatt NOK 1,40
50 +	NOK 13,40	Kvantumsrabatt NOK 2,60
100 +	NOK 12,60	Kvantumsrabatt NOK 3,40

– 1 + Legg i handlevogn

1 Min.bestilling

585 på lager for omgående utsendelse

Pris per stykk Produktserie >

NOK 11,50 (ekskl. MVA)

3 +	NOK 11,50
50 +	NOK 10,70
100 +	NOK 10,40

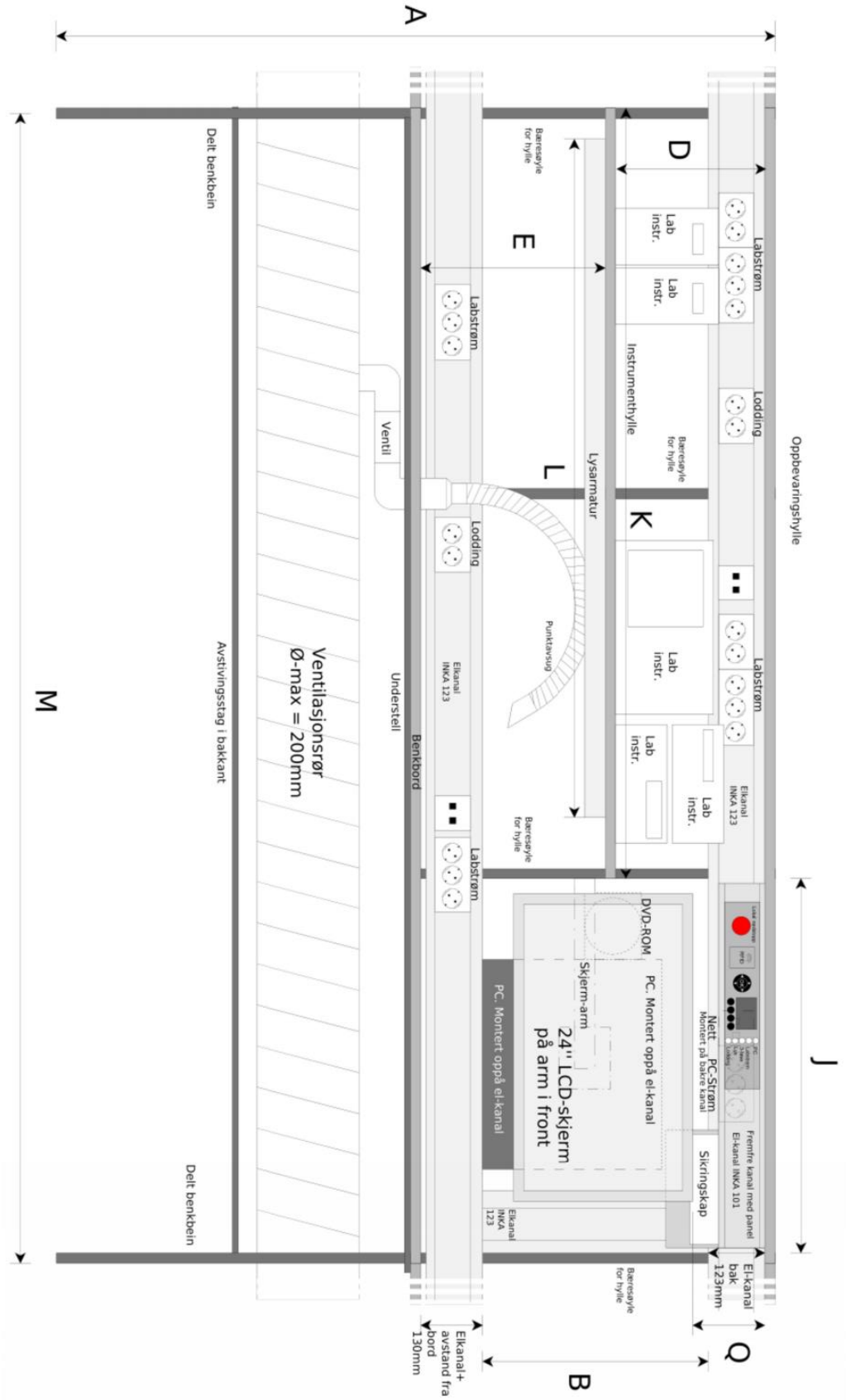
– 3 + Legg i handlevogn

3 Min.bestilling

Komponent	URL
Skjerm	https://www.waveshare.com/4.3inch-HDMI-LCD-B.htm
Raspberry Pi	https://www.digikey.no/product-detail/no/raspberry-pi/RASPBERRY-PI-4B-2GB/1690-RASPBERRYPI4B-2GB-ND/10258782
RFID	https://www.kultogbillig.no/Elektronikk/RFID-RC522-RF-IC-Card-Sensor-Modul
PCB	https://jlcpcb.com/

Ytterlige komponenter	https://www.elfadistrelec.no/no/nettboks-for-panelmontering-poles-lumberg-connect-gmbh-neb-21/p/30068534?track=true&no-cache=true&marketingPopup=false https://www.elfadistrelec.no/no/wr-phd-rett-hunn-kretskortkontakt-hullmontert-rekker-40-kontakter-54mm-pitch-wuerth-elektronik-61304021821/p/30024958?track=true&no-cache=true&marketingPopup=false https://www.elfadistrelec.no/no/motstand-600mw-470ohm-vishay-mbb02070c4700fct00/p/16059206?queryFromSuggest=true

U. Teknisk-mekanisk tegning for labplass



V. A3 Poster

NTNU
Kunnskap for en bedre verden

Bacheloroppgave Elektroingeniør

Serverbasert køsystem med adgangskontroll

Kandidat:
Andreas Glomsrud
Pragaash Mohan
Sondre Ravnås Vik
Jesper Oddaker

Veileder:
Olav Aleksander Myrvang IES, NTNU
Ingulf Helland IES, NTNU

Bakgrunn

Institutt for elektroniske systemer (IES) er underlagt NTNU, som underviser og forsker elektroniske systemer samt videreutvikling av dette. Under dette har IES også ansvaret for undervisningslaboratorier, som elektrolaboratoriet. Denne er primært brukt av 1. og 2.års studenter og har en teoretisk kapasitet på 264 studenter fordelt på 88 labplasser.

Det er identifisert grunnet laboratoriets utforming samt tidvis lav kapasitet av studentassistenter føres til forsinkelser og uheldige ventetider for studentene. Dette kan resultere i en skjev fordeling av hvem som får hjelp, og kan forverre erfaringen for både studenter og assistenter samt redusere læringsutbytte. I denne forbindelsen er det ønskelig å anvende et køsystem som skal være parallelt med PLS-systemet på hver labplass. Dette skal sikre rettferdig adgang til hjelp fra Studentassistenter.

Tekniske rammer

Hver arbeidsplass har en INKA101 elkanal installert. Denne er oppkoblet over en PC-skjerm og er tiltenkt kø- og styresystemet. Arbeidsplassen er utstyrt med en 40-leder-flatkabel, hvorav 10-15 ledere er reservert PLS-styresystemet, en CAT-6 nettverkskabel samt en 12-15V DC forsyningskabel.

Figur 1: Bilde av kanal på lab plass.

Systemstruktur

Vi har valgt å bygge et køsystem designet for å passe in INKA101 elkanalen. Dette køsystemet skal ha mulighet for framtidig utvidelse for PLS-systemet.

Delene brukt i design av systemet er følgende:

Datamaskin	Raspberry Pi 4B
Skjerm	4.3'' Kapasitiv touch LCD
RFID-leser	MFRC522 Modul
Navigasjonsknapper	Navimec 5G 1ZB/1ZCS
PCB	Tilpasset design

Tabell 1: Komponentliste

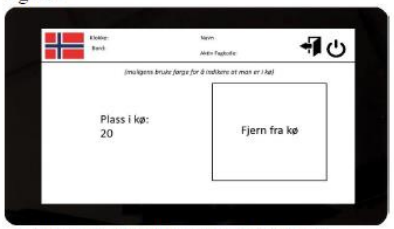
Systemet er designet for tilkobling på en PCB som skal settes inn i INKA101 elkanalen. Fullstendig oppkobling vises i Figur 2

Figur 2: Modell av ferdig oppkoblet køsystem

Køsystem

Køsystemet er bygd på tilkobling til et lokalt nettsidehierarki med tilhørig database via Chromium nettleser. Raspberry Pi systemene kjører en modifisert versjon av Raspberry Pi OS-Lite som starter opp systemet og kobler til nettsidene i kiosk-modus. Deretter kan man logge inn på nettsidene ved hjelp av RFID lesning av NTNU studentkort.

Når man er logget inn og er i en Lab økt kan man forespørre hjelp fra Studentassistent ved et enkelt trykk på skjermen. Enten ved bruk av touch eller knappetrykk. Du vil da få tilbakemelding om din plass i køen slik som i figur 3



Figur 1: Grafisk framstilling av systemet for bruker

Prototype

På grunn av tilgangsmangel på ønskede deler, har vi bygget en forenklet prototype for testing av systemets funksjonalitet. Prototypen er satt sammen av 2 oppkoblede Raspberry pi systemer koblet til en alternativ skjerm. Disse to Raspberry Pi systemene er deretter koblet sammen med Ethernet til en server og en administrator PC for testing.

Figur 2: Oppkoblet prototype

Konklusjon

Oppgaven for prosjektet var å designe et køsystem for å løse problemstillingen knyttet til studentenes bruk av elektrolaboratoriet til IES. Gruppen tok som mål å utvikle en prototype som ville danne et fleksibelt grunnlag for fremtidige implementasjoner av systemet. Prosjektet har innfridd målene gitt gruppens avgrensinger og presenterer et levedyktig konsept som løsning på utfordringen knyttet til oppgaven.

Mai 2021

Mohan, Vik, Oddaker, Glomsrud

Serverbasert køsystem med adgangskontroll

NTNU
Kunnskap for en bedre verden