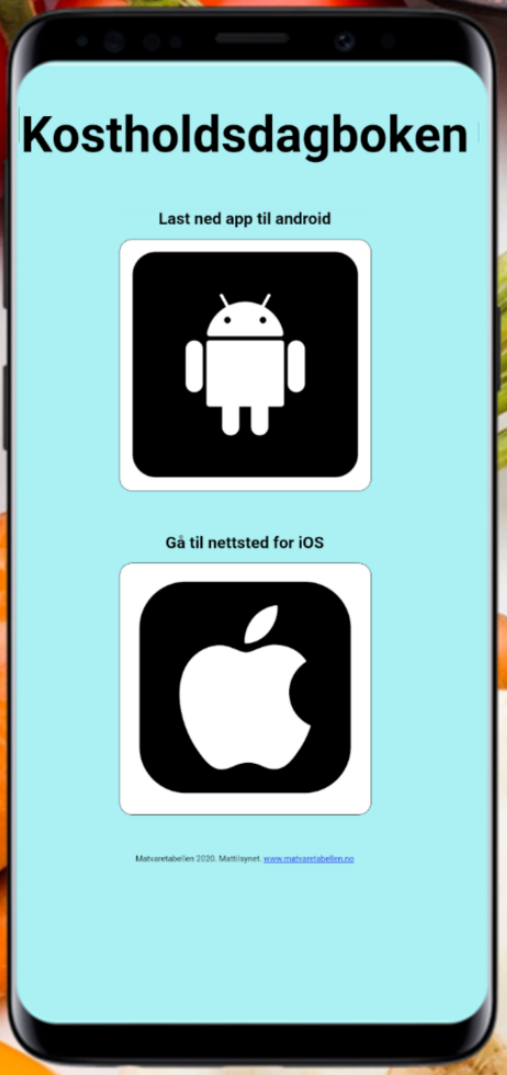
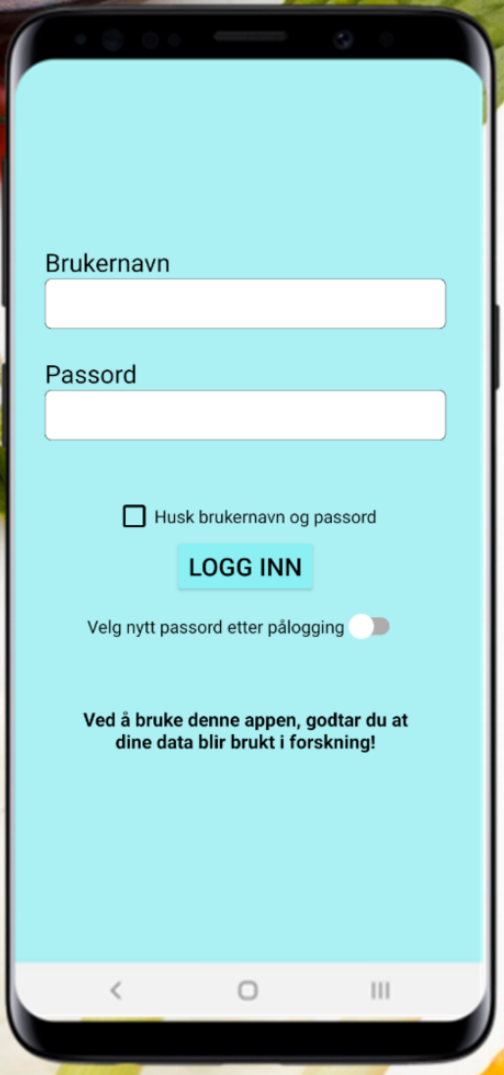
Bachelorprosjekt 2021





MEDDOC research

Ernæringsapplikasjon for Meddoc Research AS

Einar Kvam Lundberg

Institutt for Informasjonsteknologi
Postadresse: Postboks 4 St. Olavs plass, 0130 Oslo
Besøksadresse: Holbergs plass, Oslo

Telefon: 22 45 32 00

PROSJEKT NR.
26

TILGJENGELIGHET
Åpen

BACHELORPROSJEKT

HOVEDPROSJEKTETS TITTEL	DATO
Kostholdsdagboken	**26.05.2021**
	ANTALL SIDER / BILAG **117**
PROSJEKTDELTAKERE **Einar Kvam Lundberg, s331416**	INTERN VEILEDER **Trym Lindell**

OPPDRAGSGIVER	KONTAKTPERSON(ER)
Meddoc Research AS	**Stig Larsen** **Hans Fagertun**

SAMMENDRAG

Dette produktet består av en mobilapplikasjon for idrettsutøvere i et forskningsprosjekt arrangert av Meddoc Research AS i samarbeid med Norges Idrettshøgskole. Den går ut på at utøverne skal registrere kostholdet sitt. Applikasjonen er utformet i to versjoner, én for android og én for iOS, hvor hensikten er å slippe manuell registrering på papir. Denne rapporten tar for seg planlegging, utvikling og testing gjennom prosjektperioden.

3 STIKKORD
Kostholdsregistrering
Mobilapplikasjon
Databaseintegrasjon

Forord

Denne rapporten dokumenterer arbeidet rundt mitt hovedprosjekt gjennom våren 2021 ved OsloMet. Prosjektet har vært krevende, men også spennende, morsomt og ikke minst veldig lærerikt. Jeg har fått testet mye av det jeg har lært i løpet av årene på studiet, da jeg har måttet bruke kunnskapen for å sette sammen alt til et ferdigstilt sluttprodukt.

Rapporten tar for seg metodene og verktøyene jeg har brukt for å komme frem til det ferdige produktet. Beslutninger som er tatt underveis og endringene som er gjort i løpet av prosjektperioden, blir reflektert i rapporten. Rapporten er optimalisert for pc, da det er mulig å navigere rundt i dokumentet ved å klikke på overskriftene i innholdslista.

Jeg har gjennomført prosjektet sammen med min oppdragsgiver og arbeidsgiver Meddoc Research AS, etter ønske fra Norges Idrettshøgskole. Takk til daglig leder Stig Larsen og leder for biometri Hans Fagertun ved Meddoc, for å ha gitt meg mulighet til å gjennomføre og utvikle dette produktet.

Takk til min veileder Trym Lindell for å ha gitt meg god veiledning gjennom hele prosjektperioden.

Innholdsfortegnelse

1. Innledning

1.1 Om meg

Mitt navn er Einar Kvam Lundberg og jeg har studert dataingeniør ved OsloMet siden jeg startet på studiet høsten 2018. Jeg valgte å jobbe alene på dette prosjektet da jeg fikk tilbudt en oppgave fra arbeidsgiveren min, og tenkte derfor at det var mest hensiktsmessig å være alene da jeg allerede var kjent med systemene deres. Jeg var klar over at det å være alene ville føre til mye arbeid, men samtidig ville det gi meg økt fleksibilitet og bedre oversikt over arbeidet.

1.2 Om oppdragsgiver

Min oppdragsgiver og arbeidsgiver, Meddoc Research AS, er et medisinsk forsknings-firma som driver klinisk forskning av medisinske produkter på oppdrag fra bioteknologisk og farmasøytisk industri. Arbeidsoppgavene består i planlegging og gjennomføring av kliniske studier. Dette innbefatter statistisk studiedesign, utvikling av forsøksplaner og Case Record Forms (CRF), klinisk monitorering, databasekonstruksjon og datainnsamling. Etter at studiedata er samlet inn, ferdigstilles og dokumenteres prosjektets database og statistisk analyse utføres. Studierapporter og andre dokumenter brukes deretter av oppdragsgiver når produktet skal godkjennes av regulatoriske myndigheter for salg. Meddoc arbeider internasjonalt og har operasjoner i Europa, Asia og f.o.m. år 2020 også i USA.

1.3 Om oppgaven

På oppdrag fra Norges Idrettshøgskole (NIH) ønsker Meddoc en applikasjon hvor brukerne (idrettsutøvere) kan registrere data over kostholdet sitt i en elektronisk dagbok. Hensikten med denne applikasjonen er å slippe manuell registrering i form av papir. Ved å bruke en app blir det enklere både for brukerne, og også for forskerne som får dataene sendt direkte til databasen sin. Slike løsninger finnes det flere av, men et problem NIH har hatt med disse, er at brukeren kan se kaloriinntaket sitt etter registrering. Mange idrettsutøvere synes dette tallet kan virke stort og skremmende, og velger å spise mindre for å redusere kaloriene. For en

toppidrettsutøver er det kritisk å få i seg en rikelig mengde næring for å kunne prestere optimalt.

Mitt prosjekt går dermed ut på å utvikle en applikasjon hvor brukerne kan registrere kostholdet sitt, uten å kunne se kaloriinntakene sine og andre ernæringsdata etter registrering.

1.4 Rammebetingelser

I dette prosjektet skal jeg lage en applikasjon hvor brukerne kan registrere data for ernæringsstatus i en elektronisk dagbok. Det er ønskelig at applikasjonen er brukervennlig, enten i form av en webløsning tilpasset for mobil, eller en app som kan installeres på mobilen. Dataene skal så lagres og sendes til en database på webserver. Her skal dataene kunne hentes ut og brukes videre i forskningen til Meddoc.

Utover dette var det ikke mange rammer, så jeg hadde stor mulighet til å velge hvordan løsningen skulle bli. Disse rammene, og mine egne krav, står det mer om under kravspesifikasjonen i avsnitt **2.2.1**.

1.5 Teknologiske rammer

Meddoc hadde ingen krav til applikasjonen når det gjaldt teknologiske rammer, så lenge dataene blir sendt til en Microsoft SQL Server-database. Jeg har erfaring med SQL fra faget Databaser, så dette passet meg ypperlig. Siden det her heller ikke ble satt så mange rammer, kunne jeg selv velge de språkene og verktøyene jeg ønsket å benytte. Disse står det mer om under avsnitt **2.1.4** og **2.1.5**.

1.6 Mål

Målet med prosjektet er å utvikle en digital løsning som erstatter manuell registrering i form av papir for deretter å legge dataene manuelt inn i database. Dette øker effektiviteten, i tillegg til at Meddoc slipper å arrangere møter med idrettsutøverne for at de skal levere papirskjemaene.

I tillegg er det ønsket av Norges idrettshøgskole at utøverne ikke skal kunne se kalorimengden sin, da det har vært et problem at utøvere kutter ned på næringsinntaket sitt hvis de føler at de spiser for mye.

2. Prosessdokumentasjon

2.1 Planlegging og metode

Denne delen av rapporten tar for seg bakgrunnen for prosjektet, planleggingen, gjennomføringen og verktøy som har blitt brukt underveis.

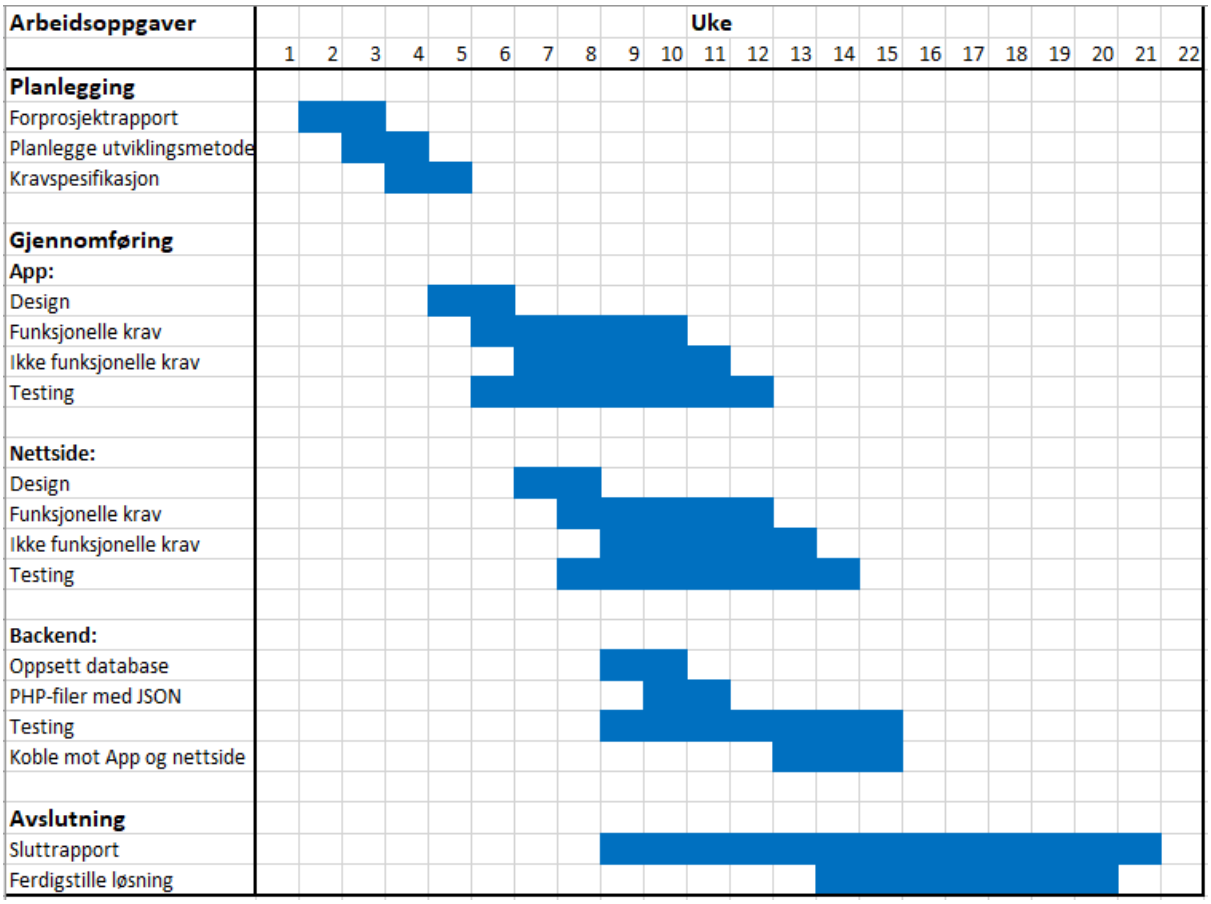
2.1.1 Bakgrunn for prosjektet

August 2019 begynte jeg i jobb hos Meddoc. Allerede da var vi i dialog angående bacheloroppgave hos dem, og jeg la frem hvilke kunnskaper jeg hadde. De fortalte at de hadde noen mulige aktuelle oppgaver til meg. Det ble drøftet litt frem og tilbake utfra behov, og i november var oppgaven delvis klar. Meddoc ønsket en mobilvennlig applikasjon for å registrere data direkte til database, istedenfor å registrere manuelt på papir og deretter innsending til databasen. Dette ville spare mye tid. I løpet av desember og frem til nyttår var vi i dialog med Norges Idrettshøgskole (NIH), som også skulle være med på prosjektet, angående hvilke krav de hadde og hva de ønsket av applikasjonen. Oppgaven virket veldig spennende, men jeg var spent på hvordan det ville la seg gjennomføre da den virket svært omfattende.

2.1.2 Planlegging

I starten av januar begynte jeg planleggingen av prosjektet. Da det ikke var så mange krav til prosjektet, annet enn de som ble nevnt i avsnitt **1.4** og **1.5**, begynte jeg med å kartlegge alt som måtte gjøres i planleggingsfasen, gjennomføringsfasen og sluttfasen.

Etter kartleggingen lagde jeg et Gantt-diagram for å ha en viss oversikt over hva som skulle gjøres til hvilken tid:



Figur 2.1.2.1: Gantt-diagram over tenkt fremdriftsplan

2.1.3 Metode

Her vil jeg ta for meg hvordan jeg gjennomførte oppgaven og løste problemer som oppstod i løpet av prosjektperioden. Dette ved å forklare min utviklingsmetode, hvordan jeg har kommunisert med oppdragsgiver og hvilke valg jeg har gjort rundt teknologi.

2.1.3.1 Utviklingsmetode

Jeg baserte min utviklingsmetode på metoden “Personlig Ekstrem programmering” (PXP) [1] som igjen er basert på “Ekstrem Programmering” (XP). XP er en utviklingsmetode som er utformet for å være så fleksibel som mulig. Man setter ikke så mange krav i starten, men har tett oppfølging med kunden hele veien. Man lager små utgaver av systemet, og forbedrer litt etter litt. En annen viktig del av XP er parprogrammering. Hensikten med dette er å fjerne distraksjoner og forbedre fokus. Det er her PXP skiller seg mest fra XP, ved at man jobber alene. Istedenfor å jobbe i par, er det viktig å holde seg fokusert og ikke bli distrahert. Ettersom jeg er alene, har

jeg vært nødt for å holde meg disiplinert nok til å ha fokus og ta jevnlige pauser slik at produktet ble best mulig.

Ettersom dette var en utviklingsmetode basert på å jobbe alene, tenkte jeg at denne utviklingsmetoden passet meg bra. Jeg er veldig fleksibel og kan lett endre på produktet underveis. Når jeg eller oppdragsgiver kom på noe som burde legges til eller endres, skrev jeg det opp i en liste i applikasjonen Trello, og fikset dette så fort det lot seg gjøre.

I forprosjektrapporten skrev jeg at jeg ville bruke Scrum, en annen fleksibel utviklingsmetode. I Scrum blir arbeidsoppgavene delt inn i sprinter (tidsperiode på 2-4 uker). I Gantt-diagrammet over, har jeg basert tidsperiodene på dette. Grunnen til at jeg raskt endret utviklingsmetode, er fordi man i Scrum ikke kan endre på noe man har startet på før sprinten er over. Dessuten er det i Scrum definert en del roller som passer bedre for flere personer. Det var dette som gjorde at jeg endret metode. I PXP har jeg mulighet til å endre noe på veldig kort varsel. Det oppstod flere situasjoner hvor jeg fant ut at det var bedre å gjøre noe på en annen måte enn jeg først hadde tenkt.

2.1.3.2 Kommunikasjon med oppdragsgiver

Jeg hadde god dialog med oppdragsgiver under hele prosjektperioden, enten i form av å fysisk være tilstede hos Meddoc, eller ved å sende bilder og videosnutter gjennom Skype. Meddoc ga meg øyeblikkelig respons i arbeidstiden, men responsen var rask utenom dette også, selv i helger. Det gjorde det enkelt for meg å få oppklart usikkerheter eller delt tanker om mulige løsninger og annet, og jeg fikk raskt jobbet videre med prosjektet.

2.1.3.3 Valg av teknologi

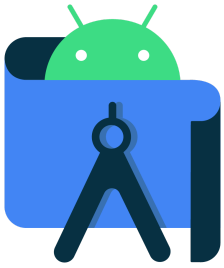
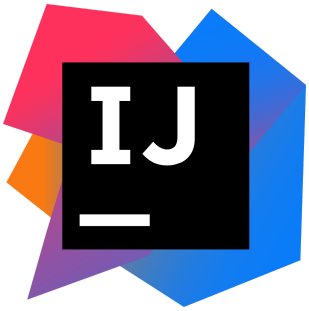
Ettersom applikasjonen skal kunne brukes for både android og iOS, var jeg tidlig inne på tanken om å bruke React Native [27]. Dette er en kryssplattform-teknologi som gjør at man kan benytte samme kode til begge de mobile operativsystemene. Dette var imidlertid ikke noe jeg hadde kunnskap om fra før av. Det viktigste for meg var å få et ferdig sluttprodukt, så istedenfor å lære meg et nytt rammeverk, lot jeg det ligge.

Jeg har erfaring med android-utvikling fra tidligere, og jeg bestemte meg raskt for å starte med android-applikasjonen for å få et produkt som jeg kunne bruke som mal til å lage applikasjonen for iOS. For android ville jeg bruke Java og XML for å bygge applikasjonen. Applikasjoner for iOS har jeg ikke noen erfaring med, så jeg bestemte meg heller for å lage en nettside beregnet for mobilt bruk i HTML, CSS og JavaScript. Dersom det skulle bli tid, ville jeg se om jeg klarte å lage en applikasjon i React Native istedenfor.

2.1.4 Verktøy

Gjennom dette prosjektet har jeg brukt flere nyttige verktøy, og noen av de viktigste er listet opp under.

Jeg har hovedsaklig brukt Android Studio og IntelliJ gjennom prosjektet til selve kodingen. Disse to er de IDEene [32] (integrert utviklingsmiljø) jeg har benyttet mest gjennom studiet. Android Studio har jeg brukt til android-utvikling og IntelliJ til alt annet.

 *Figur 2.1.4.1: Android Studio*	**Android Studio** - Android Studio er et integrert utviklingsmiljø (IDE) laget spesifikt for android-utvikling. Det er den offisielle IDEen til dette.
 *Figur 2.1.4.2: IntelliJ*	**IntelliJ** - IntelliJ IDEA er et integrert utviklingsmiljø (IDE) for utvikling av programvare. Dette har vært min hoved-IDE gjennom hele studiet.

 *Figur 2.1.4.3: Google Docs*	**Google Docs** - Google Docs er en nettbasert tekstbehandler. Jeg har brukt dette for at veileder lett skulle kunne sjekke rapporten min underveis og kommentere.
 *Figur 2.1.4.4: Skype*	**Skype** - Skype er et dataprogram for å kunne ha samtaler med mennesker over internett. Jeg brukte dette for å ha kontakt med oppdragsgiver.
 *Figur 2.1.4.5: Zoom*	**Zoom** - Zoom er et program for å arrangere møter over internett med andre. Jeg brukte dette for å ha veiledningsmøter med min veileder.
 *Figur 2.1.4.6: MSSQL SMS*	**MSSQL SMS** - Microsoft SQL Server Management Studio er et program som brukes til å administrere en SQL Server fra Microsoft. Det er dette programmet som blir brukt for å administrere serveren hos Meddoc, så jeg brukte det hos dem i tillegg til testing på min egen datamaskin.
 *Figur 2.1.4.7: Xampp*	**Xampp** - Xampp er en programvare for å kjøre en lokal server med database på egen datamaskin. Jeg brukte dette sammen med MSSQL SMS for å teste databasekoden før jeg la de inn på Meddoc sin server.

 *Figur 2.1.4.8: Trello*	**Trello** - Trello er en nettbasert applikasjon for å holde oversikt på oppgaver som skal gjøres, hva man holder på med og hva som er ferdig. Jeg brukte dette mye for å minne meg selv på å gjøre endringer, samtidig som det ga meg en fin flyt i arbeidet. Ved å ha en liste over arbeidsoppgaver, visste jeg til en hver tid hva jeg skulle gjøre.
 *Figur 2.1.4.9: GitHub*	**GitHub** - GitHub er et system som brukes for versjonskontroll av kode. Jeg brukte dette aktivt for å ha en “backup” dersom noe skulle skjære seg i koden min. Ved å integrere GitHub i prosjektet, får man en sikkerhet og i tillegg kan man se alle endringer som er gjort.

2.1.5 Teknologier og rammeverk

2.1.5.1 Frontend

For android-applikasjonen har jeg brukt Java for funksjonalitet og XML for oppsett av komponenter og design. Java har jeg erfaring med fra flere fag i løpet av studiet, og i forbindelse med android har jeg brukt det sammen med XML i faget Apputvikling.

For iOS-applikasjonen har jeg brukt HTML og CSS for struktur og design, og JavaScript for funksjonalitet. Dette har jeg også hatt erfaring med siden første semester, og videreutviklet kunnskapen i faget Webapplikasjoner.

2.1.5.2 Backend

For å kommunisere med databasen har jeg brukt PHP. Gjennom en url kan man både hente ut data og sende spørringer basert på ønsket input. Jeg valgte å bruke PHP da jeg har brukt det litt fra før av i Apputvikling, og tenkte at dette enkelt kunne overføres også til nettsiden.

2.1.5.3 Oversikt over språk og rammeverk

Under er en liste over språkene og rammeverkene jeg har brukt i prosjektet.

 *Figur 2.1.5.1: Java*	**Java** - Java er et objektorientert programmeringsspråk. Jeg brukte java-kode i android-programmeringen.
 *Figur 2.1.5.2: XML*	**XML** - Extensible Markup Language er et kodespråk for å definere regler for koding av dokumenter. Jeg brukte dette for å designe android-applikasjonen min.
 *Figur 2.1.5.3: HTML*	**HTML** - Hyper Text Markup Language brukes til å lage strukturen til et brukergrensesnitt. I mitt tilfelle ble det brukt til oppsett av nettside-delen av produktet.
 *Figur 2.1.5.4: JavaScript*	**JavaScript** - JavaScript er et programmeringsspråk for å lage dynamikk på nettsider. Jeg brukte dette til alle funksjoner på nettsiden.

 *Figur 2.1.5.5: jQuery*	**jQuery** - jQuery er et JavaScript-bibliotek for forenkling av HTML-skripting. Jeg brukte dette for å redusere en del av JavaScript-koden.
 *Figur 2.1.5.6: CSS*	**CSS** - Cascading Style Sheet brukes til å designe applikasjoner skrevet i HTML. Dette var også mitt tilfelle for nettsiden.
 *Figur 2.1.5.7: PHP*	**PHP** - PHP er et skriptspråk for utvikling av dynamiske nettsider. Jeg brukte dette for å koble applikasjonene mine til database og sende data mellom databasen og programmet.
 *Figur 2.1.5.8: SQL*	**SQL** - Structured Query Language er et spørrespråk for databaser for å hente ut eller legge til ønsket data. Jeg brukte dette for databasedelen av produktet.
 *Figur 2.1.5.9: Node.js*	**Node.js** - Node.js er et system for å kjøre JavaScript-kode på server. Det kommer også med mange moduler. Jeg benyttet meg av Node.js for å bruke en modul til kryptering av passord.

 *Figur 2.1.5.10: Browserify.js*	**Browserify.js** - Browserify.js er et JavaScript-verktøy som lar deg kjøre Node.js-moduler i nettleseren. For at jeg skulle få brukt krypteringsmodulen til Node.js, trengte jeg å kjøre det i nettleseren.

2.1.5.4 Hvorfor ikke bare én app i nettleser?

Jeg utviklet som nevnt over en android-applikasjon som kan installeres på telefonen og en nettside for iOS. Grunnen til at jeg valgte å lage en egen android-applikasjon, istedenfor å bare lage en felles nettside, er på grunn av tid. Dersom det hadde blitt mer tid, ville jeg ha utviklet en egen app også for iOS.

2.2 Utviklingsprosessen

Her vil jeg skrive om selve prosessen gjennom prosjektet. Dette innebærer kravspesifikasjonen, produktutviklingen og en evaluering av hvordan utviklingsprosessen har vært.

2.2.1 Kravspesifikasjon

Ettersom jeg ikke fikk så mange krav til hvordan denne applikasjonen skulle være, hadde jeg mulighet til å bestemme mye selv. Jeg lagde meg en grov kravspesifikasjon over de punktene jeg fikk, og hva jeg selv ønsket å få med i applikasjonen. Jeg estimerte også hvor stor prioritet de ulike punktene hadde.

#	Funksjonelle krav	Prioritet
1	Databasen skal inneholde en kryptert versjon av passordet	Høy
2	Bruker skal kunne endre passord	Middels

3	Applikasjonen skal kunne huske brukernavn og passord	Middels
4	Bruker skal kunne registrere kostholdet sitt	Høy
5	Bruker skal kunne se når man har registrert kosthold	Middels
6	Bruker skal ikke kunne se kaloriinntaket sitt	Høy
7	Bruker skal kunne velge hvilken enhet de ønsker å bruke for en matvare	Middels
8	Applikasjonen skal sende data til Meddoc sin database	Høy

Figur 2.2.1.1: Funksjonelle krav

#	Ikke-funksjonelle krav	Prioritet
1	Applikasjonen skal kunne brukes både for Android og iOS	Høy
2	Applikasjonen skal være brukervennlig	Middels
3	Logoens farger skal være gjennomgående i appen	Middels

Figur 2.2.1.2: Ikke-funksjonelle krav

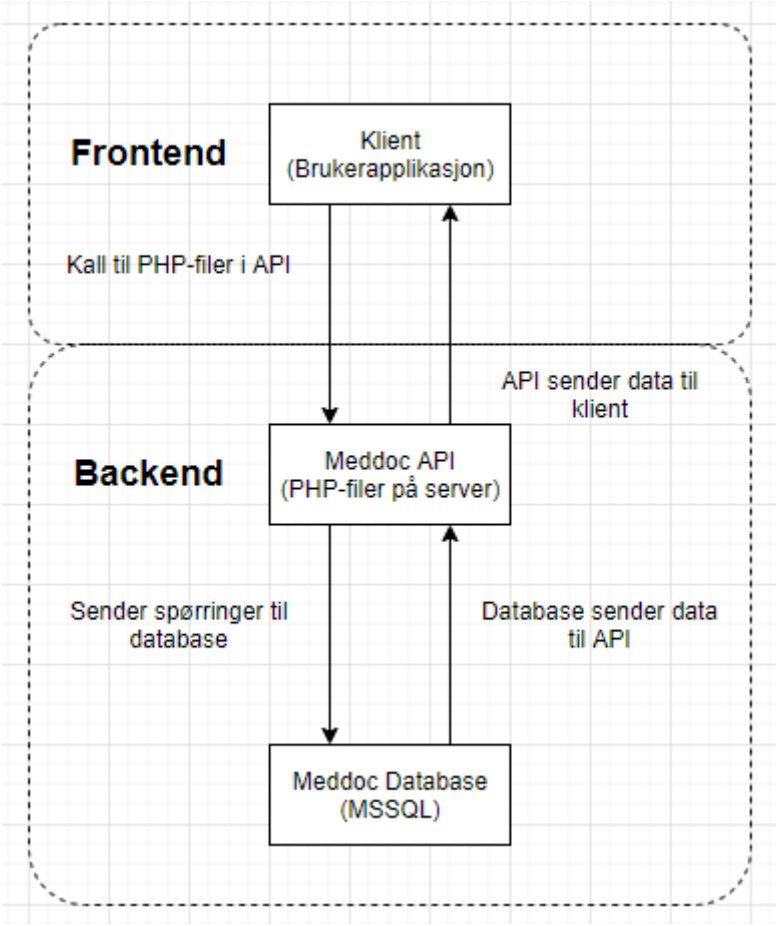
Med denne kravspesifikasjonen på plass, ble det lettere for meg å vite hva jeg måtte gjøre. Selv om kravspesifikasjonen ikke tar så stor plass i ekstrem programmering, valgte jeg å bruke god tid i starten av prosjektet til å lage denne, slik at det var lettere å komme i gang.

2.2.2 Produktutviklingen

Her vil jeg ta for meg hvilke valg jeg har tatt under selve utviklingen av produktet, samt skisser og modeller jeg har lagd underveis, Jeg vil også gi en begrunnelse for valg rundt design av applikasjonen.

2.2.2.1 Systemmodell

Her er en modell over hvordan de ulike delene av systemet henger sammen i sluttproduktet:



Figur 2.2.2.1: Modell over systemets struktur

Denne modellen lagde jeg i starten av utviklingsfasen ut fra hvordan jeg tenkte systemet kom til å bli. Jeg hadde en klar idé om hvordan jeg skulle lage produktet, og med denne modellen var det enklere å forstå hvordan systemet skulle settes opp.

Ut fra oppgaven jeg fikk, trengte jeg en applikasjon som skulle kommunisere med en database. I modellen kunne jeg altså tegne opp en klient (applikasjonen) og Meddoc sin database. For at applikasjon og database skulle kunne kommunisere med hverandre, trengte jeg et API (Application Programming Interface). Dette ønsket jeg lage med php-filer, en filtype som blant annet kan opprette forbindelse med en database og sende SQL-spørringer til den. Ved å tegne opp API'et på modellen, hadde jeg hele systemstrukturen jeg trengte.

2.2.2.2 Applikasjon

Det var ønskelig med brukervennlighet for denne applikasjonen. Det ble derfor viktig å velge farger som passet godt sammen, bruke ikoner kontra mye tekst der det lot seg gjøre og ha et enkelt og forståelig design. Merk at når jeg videre i rapporten skriver om "applikasjonen", henviser jeg stort sett til både android-applikasjonen og nettversjonen for iOS. Det er kun små forskjeller som skiller de ulike versjonene, men der det er forskjeller utdyper jeg hvilken versjon det gjelder.

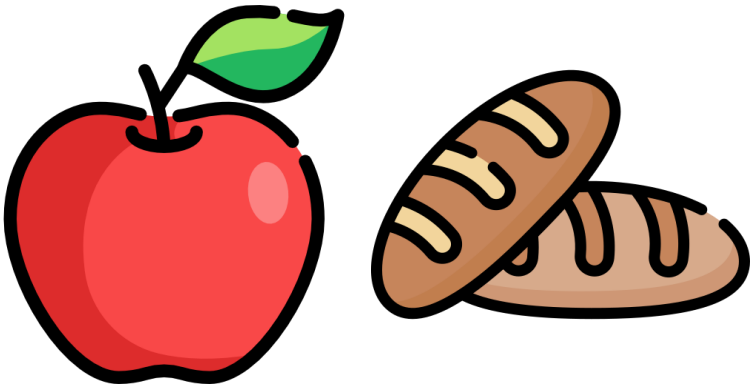
Universell utforming

I Norge er det bestemt at alle IKT-tjenester skal være universelt utformet [20]. Dette innebærer at nettsider og applikasjoner må følge minimumskravene fra WCAG 2.0 (Web Content Accessibility Guidelines 2.0) [21]. Gjennom prosjektet har jeg prøvd å holde meg til disse kravene for å ha en best mulig universell utforming.

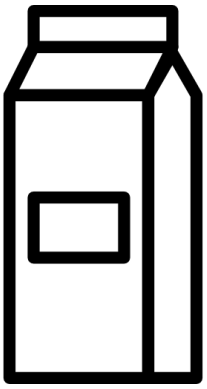
Logo

Da utviklingen av applikasjonen var kommet godt i gang, bestemte jeg meg for å lage en logo til applikasjonen. Jeg ville at den skulle være enkel og representere hva applikasjonen handler om, nemlig kosthold. Jeg fant derfor noen gratis matrelaterte ikoner fra nettstedet flaticon [3], endret farge og satte de sammen til en ferdig logo.

Ikonene jeg brukte i logoen er vist under.



Figur 2.2.2.2: Icons made by Freepik from www.flaticon.com



Figur 2.2.2.3: Icon made by Darius Dan from www.flaticon.com

Satt sammen, og med farge på melkekartongen, ble logoen slik:

Figur 2.2.2.4: Logo uten bakgrunn

Meddoc hadde ingen preferanser da det gjaldt fargevalg, så her måtte jeg prøve meg frem. Jeg testet først med en grønn bakgrunnsfarge, da grønn lett kan forbindes med helse og godt kosthold, men jeg følte ikke at matvare-objektene i logoen passet til fargen.

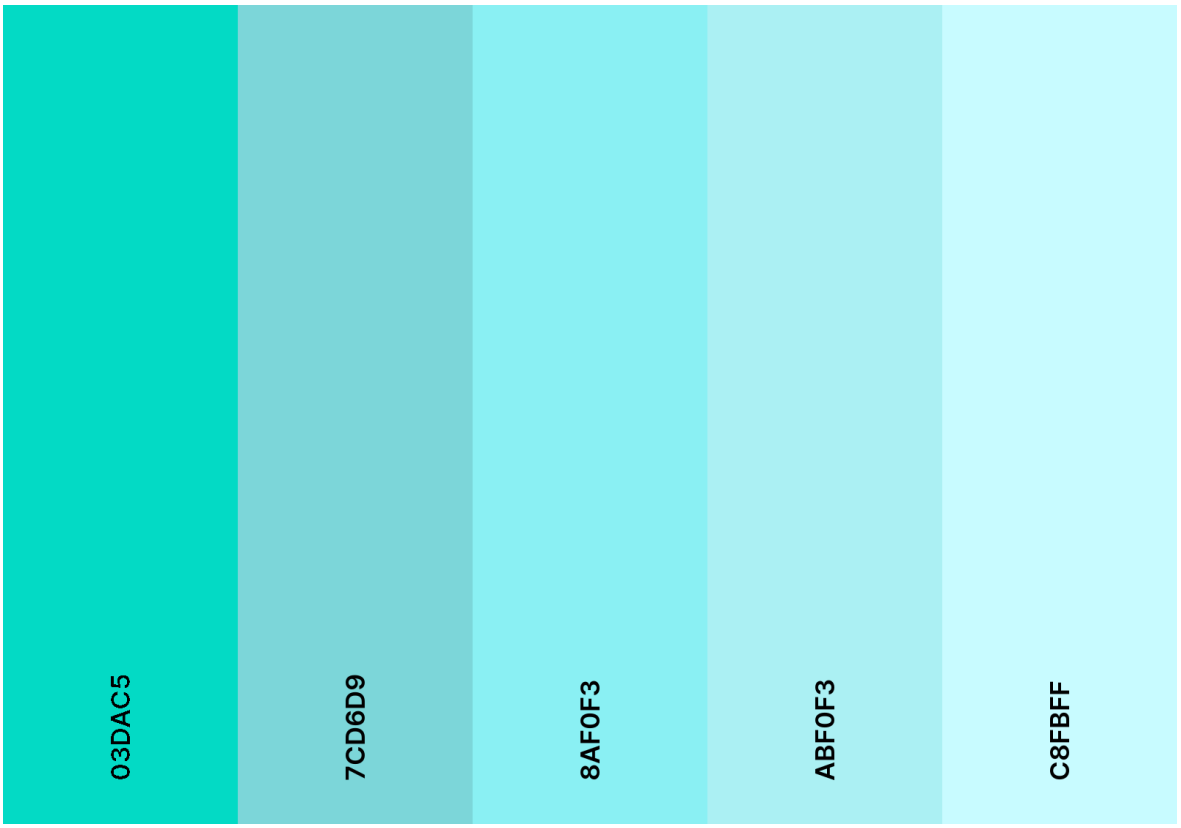
Figur 2.2.2.5: Første forsøk med bakgrunnsfarge

Jeg prøvde deretter en lyseblå farge, noe som jeg syntes ga et mye friskere utseende, og det resulterte i logoen vist under:

Figur 2.2.2.6: Applikasjonens endelige logo

Fargevalg

Nå som jeg hadde en logo, hadde jeg noe å gå ut i fra da det gjaldt farger. Ettersom logoen hovedsakelig består av lyseblå farge, valgte jeg å lage meg et fargespekter basert på denne blåfargen.

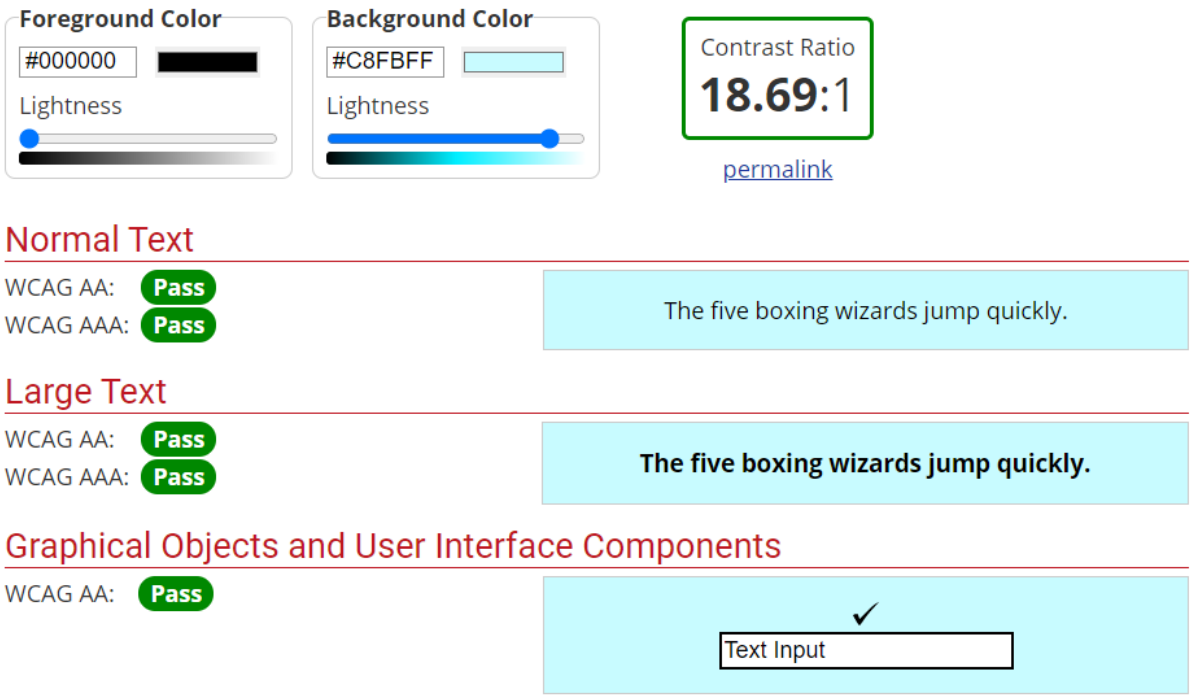


Figur 2.2.2.7: Fargepalett som ble brukt i applikasjonen

Jeg brukte disse fargene i tillegg til hvit og svart gjennomgående i applikasjonen. Det å samkjøre logoens farger med fargene i applikasjonen, fører til en bedre sammenheng og flyt. Logoen får en bedre tilhørighet til selve appen, og det forbedrer brukeropplevelsen. Dette er noe jeg hadde i fokus allerede fra planleggingsfasen.

Kontrast

Det er viktig med god kontrast i applikasjoner. For å tydelig kunne se hva som står på skjermen, spesielt for de med nedsatt syn, er dette veldig viktig. Jeg brukte et verktøy fra nettsiden *WebAIM* [2] for å sjekke kontraster mellom to farger ut fra WCAG sine krav, og sjekket alle fargene i fargepaletten vist over mot svart. Det viste seg at det var god kontrast for samtlige farger i paletten, og jeg kunne fint bruke disse fargene med svart tekst. I følge WCAG 2.0 må kontrastforholdet være minst 4.5:1 [22]. I testresultatet under er dette forholdet 18.69:1 for den gitte fargen og dermed godt innenfor kravet.



Figur 2.2.2.8: Testresultat fra kontrastverktøyet

Ikonbruk

Jeg ønsket å bruke ikoner fremfor tekst der jeg følte dette var hensiktsmessig. Ikoner gjør applikasjonen litt lettere fremfor å ha store mengder tekst, i form av at brukeren slipper å lese så mye. I tillegg er ikonbruk plassbesparende ved å gi alle komponenter mer luft, hvilket gjør helhetsinntrykket av applikasjonen bedre [16].

Når man bruker ikoner og bilder i applikasjoner, må man passe på å legge til en alternativ beskrivelse [23]. På denne måten kan brukere med nedsatt syn få hjelp av en skjermleser til å forklare hva ikonet er ment å symbolisere.

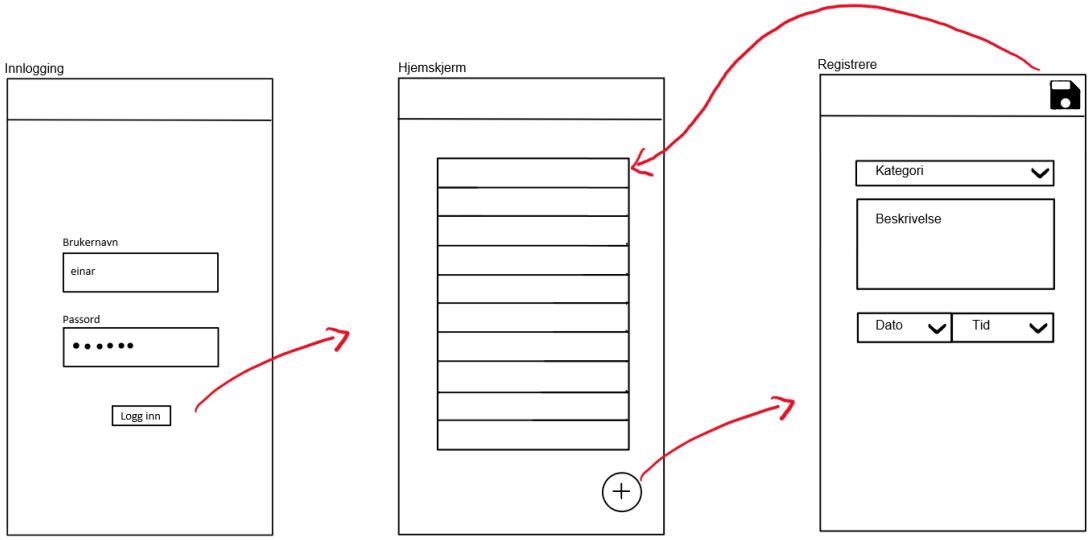
`android:contentDescription="Lagre"`	`alt="Lagre"`

Figur 2.2.2.9: Tekstlig beskrivelse av lagreknapp

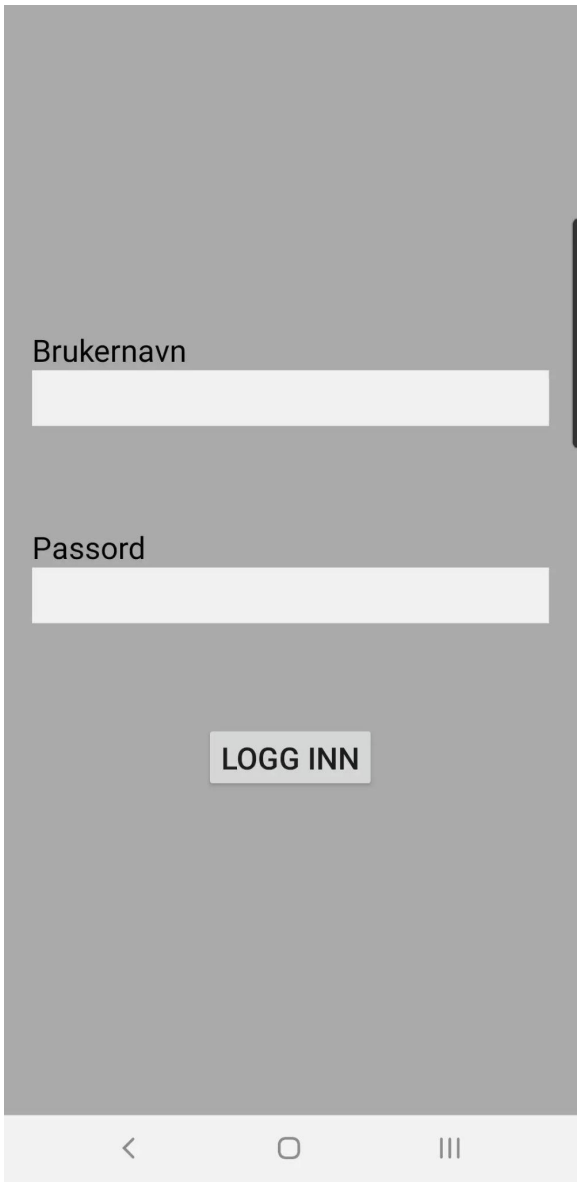
Design av applikasjonen

Jeg la tidlig frem et forslag over hvordan applikasjonen kunne struktureres. Jeg lagde en skisse over de ulike aktivitetene som trengtes, og grovt hva de skulle inneholde. Jeg tenkte å ha tre aktiviteter (sider): en innloggingsside, en oversiktsside

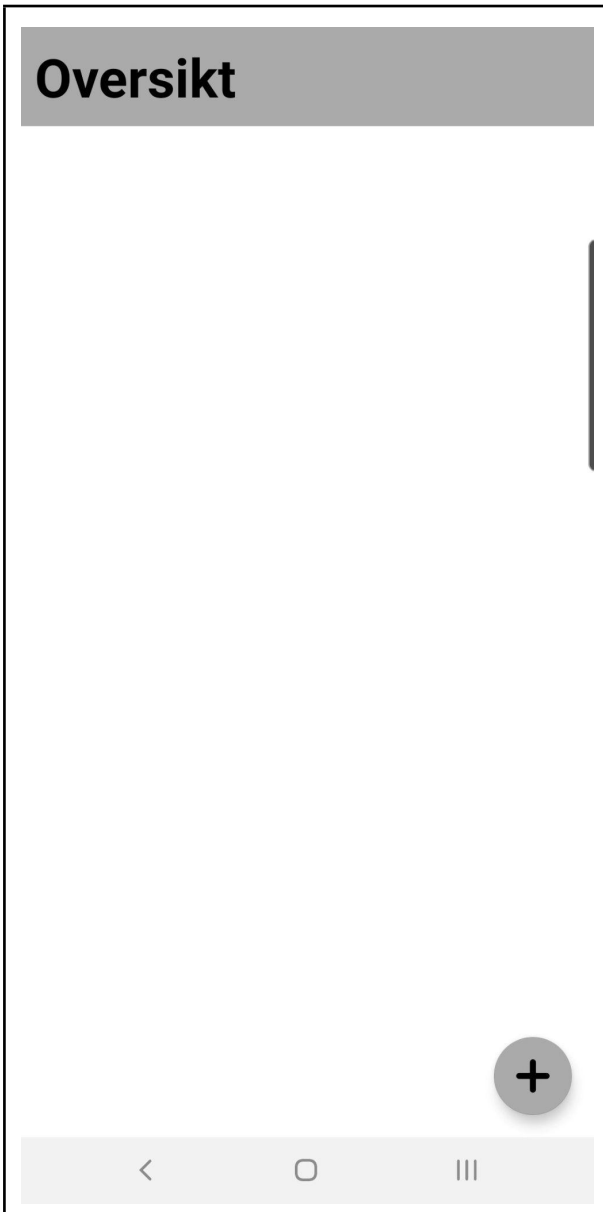
og en registreringsside. Meddoc syntes det virket som en god løsning, så jeg begynte designet av applikasjonen ut ifra denne skissen:



Figur 2.2.2.10: Skisse over aktiviteter

	**Innlogging:** Her er en tidlig utgave av innloggingssiden. Den bestod kun av to felter for å skrive inn brukernavn og passord, og en knapp for å utføre innloggingen.

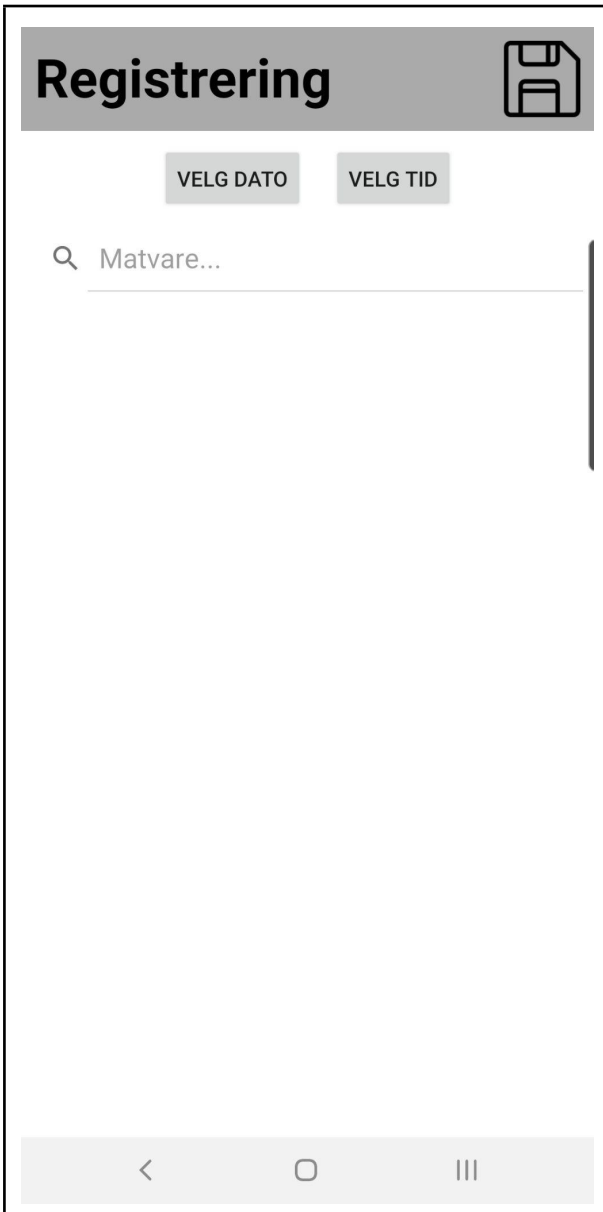
Figur 2.2.2.11: Tidlig utgave av innloggingssiden



Oversikt:

Dette er oversiktssiden fra den tidlige utgaven. Her var det ingenting utenom en knapp for å opprette en registrering.

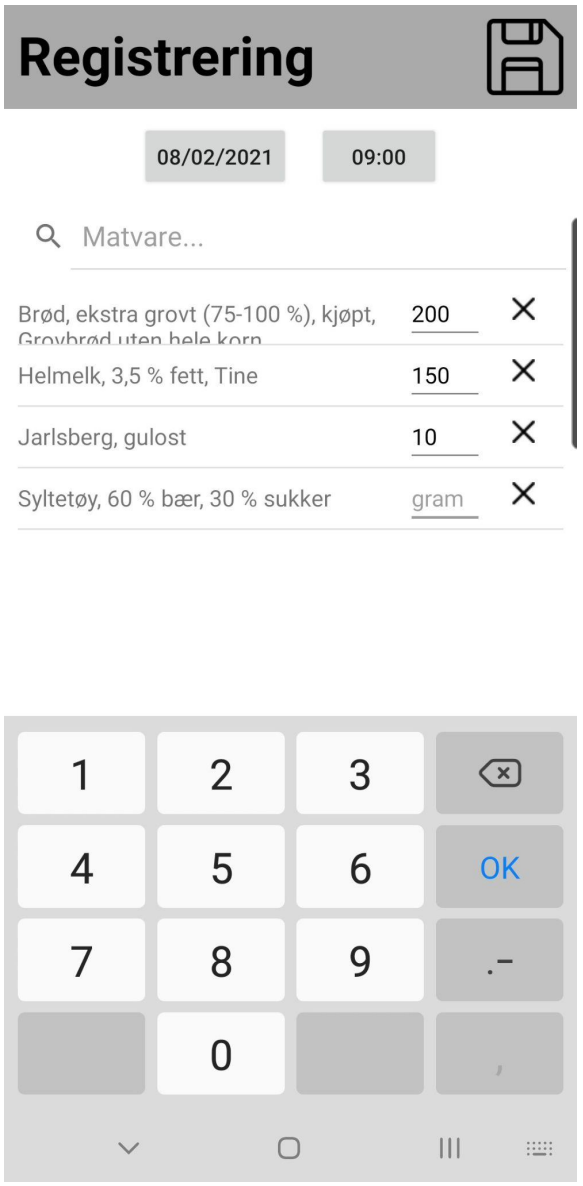
Figur 2.2.2.12: Tidlig utgave av oversiktssiden



Registrering:

I registreringssiden var det to knapper for å velge dato og tid, et søkefelt for å velge matvarer og en knapp for å lagre og sende måltidet til databasen.

Figur 2.2.2.13: Tidlig utgave av registreringssiden

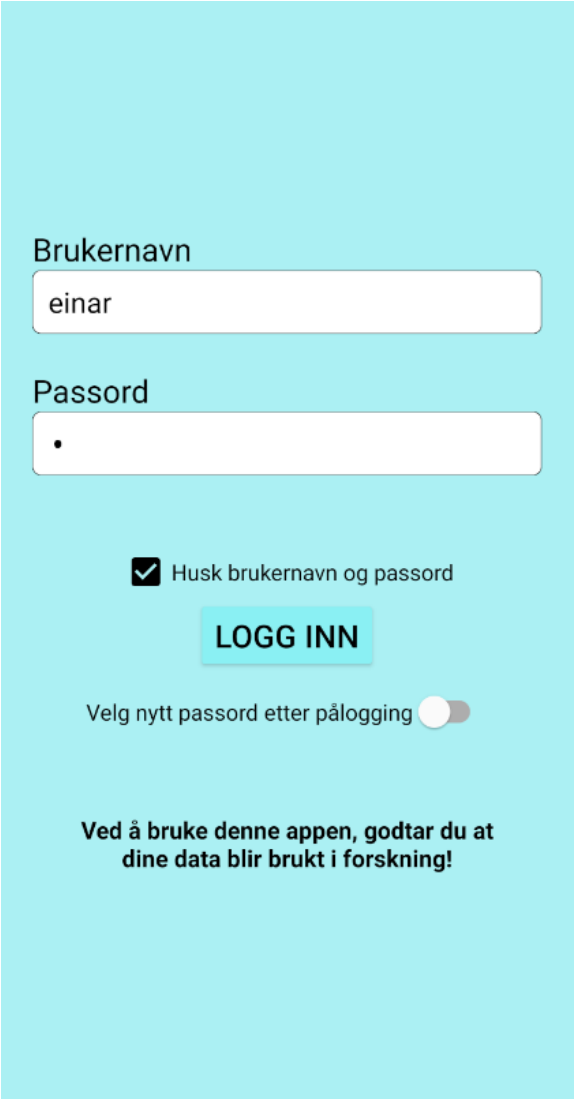


Registrering utfylt:

Når man hadde lagt til noen matvarer i lista, ble hver matvare listet opp og fikk et felt for å skrive inn antall gram av den matvaren og en sletteknapp (kryss).

Figur 2.2.2.14: Tidlig utgave av utfylte matvarer

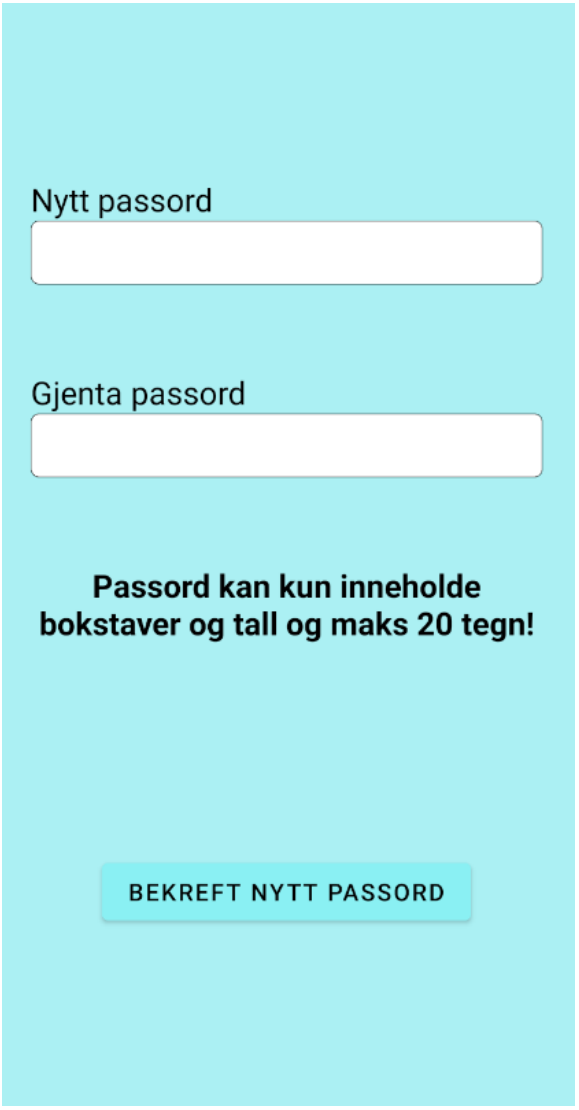
Mye av designet over ble beholdt, men forbedringer ble også gjort.



Innlogging:

I den endelige versjonen av applikasjonen har man mulighet til å la appen huske brukernavn og passord, slik at brukeren ikke trenger å skrive inn dette hver gang. I tillegg kan man trykke på en bryter dersom man ønsker å endre passord. En beskjed nederst på siden informerer om at dataen man sender vil bli brukt til forskning.

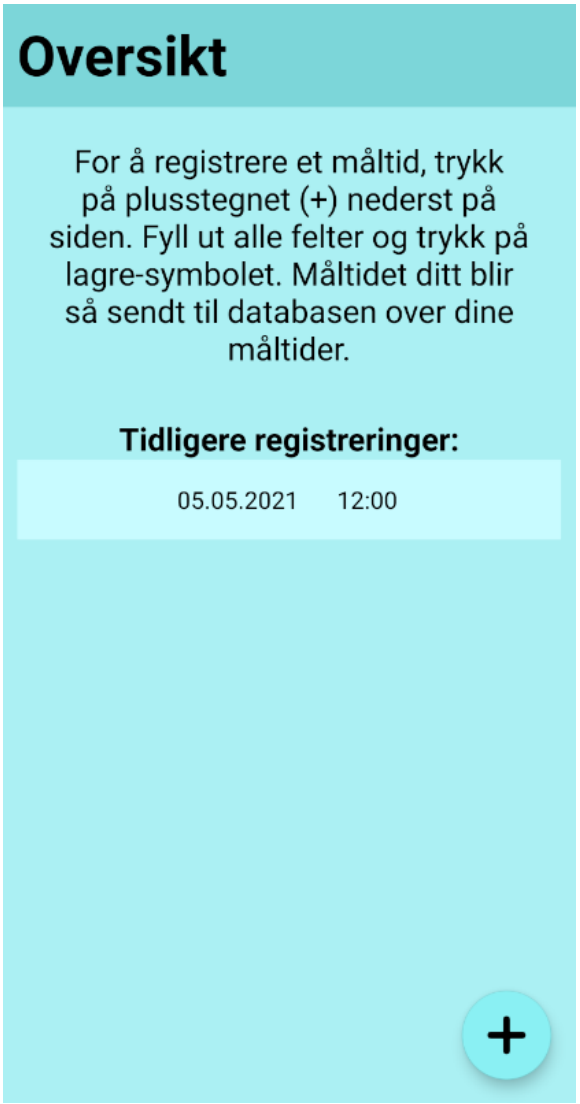
Figur 2.2.2.15: Endelig utgave av innloggingssiden



Nytt passord:

Dette er en ny aktivitet siden første utgave av applikasjonen. Her er det to felter for å skrive inn nytt passord og en knapp for å bekrefte dette.

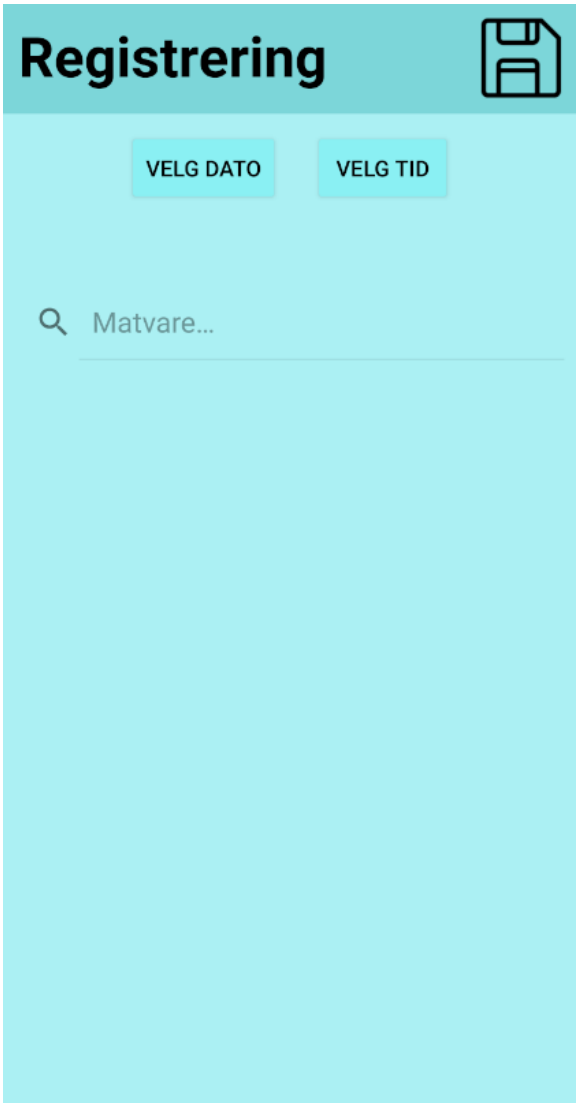
Figur 2.2.2.16: Aktivitet for å endre passord



Oversikt:

I oversiktssiden er det nå lagt til en liten instruksjon øverst på siden. I tillegg vil man få opp en liste etterhvert som man har registrert måltider. Den viser dato og klokkeslett for alle måltider som er registrert.

Figur 2.2.2.17: Endelig versjon av oversiktssiden



Registrering:

For selve registreringssiden er det ikke så mye annerledes enn fargeendringen.

Figur 2.2.2.18: Endelig versjon av registreringssiden

Registrering utfylt:

Når man har lagt til noen matvarer, kan man se noen endringer fra den tidlige versjonen. Istedenfor å bare ha gram som enhet, får man en liste over ulike enheter til høyre for inputfeltet for mengde.

Figur 2.2.2.19: Endelig versjon av utfylte matvarer

2.2.2.3 Administrasjonsløsning

I planleggingsfasen var det tenkt at jeg skulle lage en administrasjonsløsning hvor forskerne kunne hente ut dataene sendt fra applikasjonen. Dette kunne være en nettside hvor man kunne laste ned en fil med alle dataene, helst et Excel-dokument eller lignende. Dette ble det i midlertidig gått bort ifra. I stedet ble dataen sendt direkte til databasen til Meddoc. Ved å gjøre dette, slipper man å først laste ned dataen, for så å importere denne til databasen.

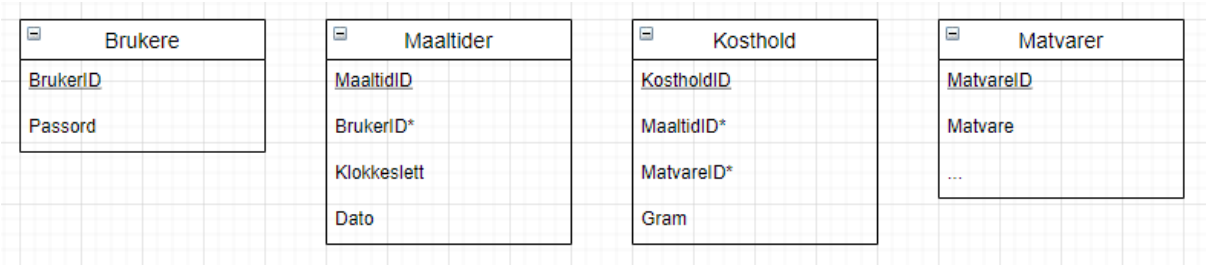
2.2.2.4 Database

Jeg var tidlig i dialog med Meddoc om hva slags database de anvendte. Jeg var forberedt på å måtte sette meg inn i det systemet de brukte. Heldigvis benyttet de seg av en Microsoft SQL Server-database (MSSQL). I denne databasen blir det

benyttet en modifisert versjon av SQL, et språk jeg har kjennskap til fra før. Selv om den er modifisert, kan man fortsatt skrive de fleste vanlige SQL-spørringer, så jeg hadde god erfaring med dette.

For å kommunisere med databasen, trengte jeg et API (Application Programming Interface) eller *programmeringsgrensesnitt*. API er et hjelpeverktøy som lar frontend kommunisere med backend, i dette tilfellet lar APIet applikasjonen kommunisere med databasen. For å få dette til, lagde jeg noen filer skrevet i PHP. PHP-filene opprettet forbindelse til databasen og sendte spørringer til den. Deretter ble data sendt tilbake via PHP-filene og videre til applikasjonen.

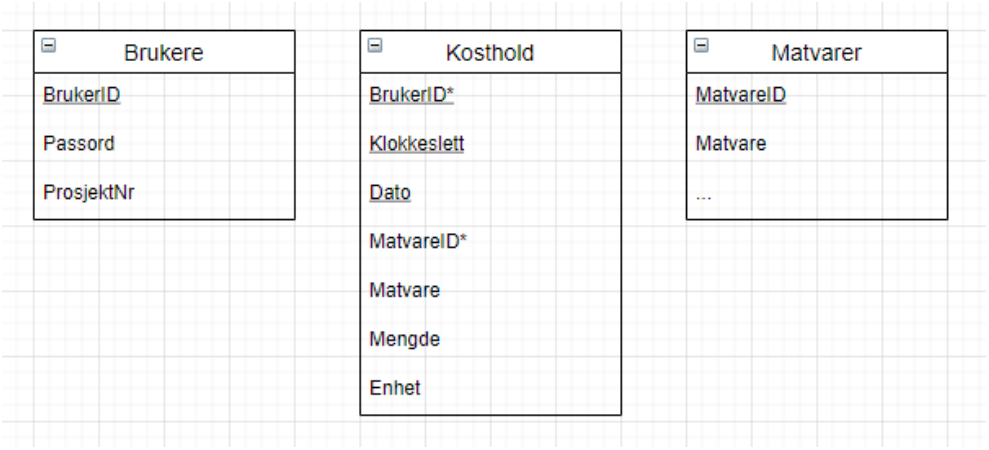
Før jeg satt opp de tabellene jeg trengte i databasen, lagde jeg meg et diagram over de ulike tabellene:



Figur 2.2.2.20: Førsteutkast av tabellene til databasen

Tanken bak tabellene var at hver bruker ble lagret i en egen tabell. Når en bruker registrerte et måltid i applikasjonen, fikk hver matvare fra måltidet en rad i tabellen "Kosthold". Dette var for å senere kunne finne ut hva som var registrert til hvilket måltid. "Matvarer" er en egen tabell med matvaredata hentet fra matvaretabellen.no[4].

Dette fungerte bra, men jeg kom over et hypotetisk problem som jeg deretter testet i praksis. *MaaltidID* var en automatisk økende ID, som ble én større for hver gang en bruker la noe inn i databasen. Dersom to brukere skulle ha registrert et måltid samtidig, ville kun den ene av dem fått måltidet sitt registrert. For å løse dette, endret jeg tabelloppsettet. Jeg fjernet "Maaltider"-tabellen og flyttet klokkeslett og dato over til kostholdstabellen. I tillegg ble gram erstattet med *Mengde* og det ble lagt inn et ekstra felt for *Enhet*. Dette var for å kunne bruke andre enheter enn gram der det var mer passende. For brukere ble det etterhvert lagt inn et *ProsjektNr* for å kunne skille brukere i fremtidige prosjekter. De nye tabellene ble slik:



Figur 2.2.2.21: Endelig versjon av tabellene i databasen

Utfordringer

Det var litt problematisk å få integrert PHP-filene med Meddoc sin server. Jeg fikk feilmeldinger da jeg prøvde å sende data til filene, og etter noen google-søk fant jeg ut at det ikke var installert drivere for PHP på serveren [18]. Jeg lastet ned og installerte en driver, og nå kom jeg litt lenger. Jeg fikk fortsatt ikke forbindelse til databasen, og etter noen dager med mer googling, fant jeg ut at jeg måtte laste ned en annen versjon av driveren til denne serveren. Da dette var i orden, fikk jeg endelig koblet meg til databasen, og jeg kunne jobbe videre.

2.2.2.5 GDPR

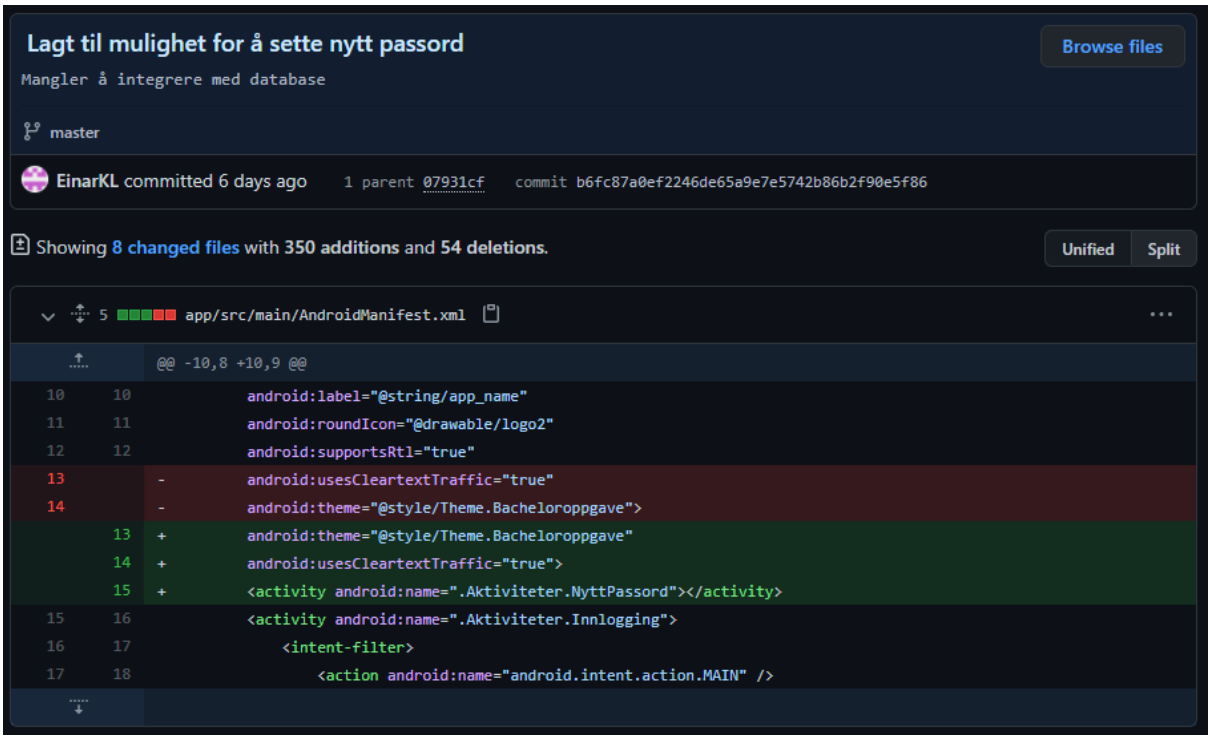
GDPR er en forordning som omfatter lagring og behandling av personopplysninger, og dette gjelder for EU og EØS [19]. Det er hovedsaklig Meddoc som tar seg av personopplysninger for deltakere i forskningsprosjektene deres. I applikasjonen lagres kun opplysninger om matinntaket til brukerne, samt passord og en forhåndsdefinert brukerID i form av et nummer. Utøverne som er med på forskningsprosjektet, har måttet signere papirer rundt bruk av deres data hos Meddoc før prosjektstart. I applikasjonen er det i tillegg lagt til en påminnelse ved innlogging om at brukerens data vil bli brukt til forskning dersom vedkommende velger å benytte appen.

2.2.2.6 Versjonshåndtering

Et versjonshåndteringssystem er programvare som kan registrere endringer i kode. Dette gir en slags historikk over koden hvor man kan gå tilbake og se ulike versjoner av koden.

Når man jobber med prosjekter, kan det lett oppstå problemer. Jeg valgte å bruke versjonshåndteringssystemet Git [33] via GitHub, nevnt i avsnitt **2.1.4**, for å ha en sikkerhetskopi av alle versjoner av koden. Hver gang jeg la til en funksjon eller et element i applikasjonen, sendte jeg koden til GitHub. På denne måten var jeg sikker på at hvis det skulle skje noe, så hadde jeg alltid en tidligere, fungerende kode tilgjengelig.

I tillegg til å ha en sikkerhetskopi, gjør GitHub det veldig enkelt å jobbe med samme kode flere steder. Dette kan blant annet brukes til samarbeid, men for meg var det praktisk for å kunne jobbe med prosjektet både hjemme og hos Meddoc. Hver gang jeg var fysisk tilstede hos Meddoc, hadde jeg på forhånd oppdatert koden min på GitHub, slik at jeg kunne fortsette å jobbe med den der.



Figur 2.2.2.22: Skjermdump fra GitHub

3. Produktdokumentasjon

3.1 Introduksjon

Denne delen av rapporten viser hvordan applikasjonen fungerer ved å vise til eksempler fra koden. Produktdokumentasjonen er skrevet slik at de fleste skal klare å forstå den. Det er fordel for leseren om vedkommende har kunnskap om koding, men det skal forklares hva de ulike kodesnuttene gjør.

3.2 Produktbeskrivelse

Sluttproduktet er en mobilvennlig applikasjon for registrering av kosthold. Produktet er todelt, og består av en nettversjon og en installerbar applikasjon for android-enheter.

Applikasjonen er beregnet for bruk av idrettsutøvere som skal delta i forskningsprosjekter. Utøverne som skal delta vil få tilsendt brukernavn og passord til innlogging. Etter innlogging kan man se en oversikt over tidspunkter for når brukeren har registrert tidligere kosthold, og man kan registrere nytt kosthold.

Samsvar mellom sluttproduktet og kravspesifikasjonen:

#	Funksjonelle krav	Beskrivelse	Krav møtt
1	Databasen skal inneholde en kryptert versjon av passordet	Passord blir kryptert før det sendes og lagres i databasen. Dette gjøres ved en implementasjon av hashing og salting med SHA256.	Ja
2	Bruker skal kunne endre passord	I innloggingssiden kan man huke av for å endre passord etter godkjent innlogging. Effekten er umiddelbar.	Ja

3	Applikasjonen skal kunne huske brukernavn og passord	Bruker kan velge å krysse av for at applikasjonen skal huske brukernavn og passord. Om dette er valgt husker applikasjonen dette ved hjelp av lokal lagring (local storage) på enheten.	Ja
4	Bruker skal kunne registrere kostholdet sitt	Brukeren kan registrere kostholdet sitt i registreringssiden. Brukeren velger matvarer fra en liste som filtreres etter søk og angir mengde og enhet for dette. Dataene sendes så til database.	Ja
5	Bruker skal kunne se når man har registrert kosthold	I oversiktssiden vil alle tidspunkter over registreringer vises i en liste, sortert etter nyeste øverst.	Ja
6	Bruker skal ikke kunne se kaloriinntaket sitt	Ingen kalkulering av kalorier eller annet næringsinnhold foregår i applikasjonen. Dette vil bli gått gjennom manuelt i ettertid for de ønskede periodene.	Ja
7	Bruker skal kunne velge hvilken enhet de ønsker å bruke for en matvare	Ved registrering kan brukere velge enheter fra en liste for hver matvare, ut fra hva som passer best. Dette er enheter som (g, ss, ml, stk osv.).	Ja

8	Applikasjonen skal sende data til Meddoc sin database	Applikasjonen kommuniserer med databasen ved å sende data om kosthold og henter data over tidligere registreringer.	Ja

Figur 3.2.1: Samsvar mellom sluttprodukt og kravspesifikasjon for funksjonelle krav

#	Ikke-funksjonelle krav	Beskrivelse	Krav møtt
1	Applikasjonen skal kunne brukes både for Android og iOS	For android er det laget en installerbar applikasjon. Det er laget en nettversjon av applikasjonen for iOS.	Ja
2	Applikasjonen skal være brukervennlig	Applikasjonen er laget så enkel som mulig. Ingen overflødige funksjoner. Applikasjonen gjør det den er ment å gjøre.	Ja
3	Logoens farger skal være gjennomgående i appen	Fargepaletten som ble laget i sammenheng med logoen, har blitt brukt for alle sider av applikasjonen.	Ja

Figur 3.2.2: Samsvar mellom sluttprodukt og kravspesifikasjon for ikke-funksjonelle krav

3.3 Systemarkitektur

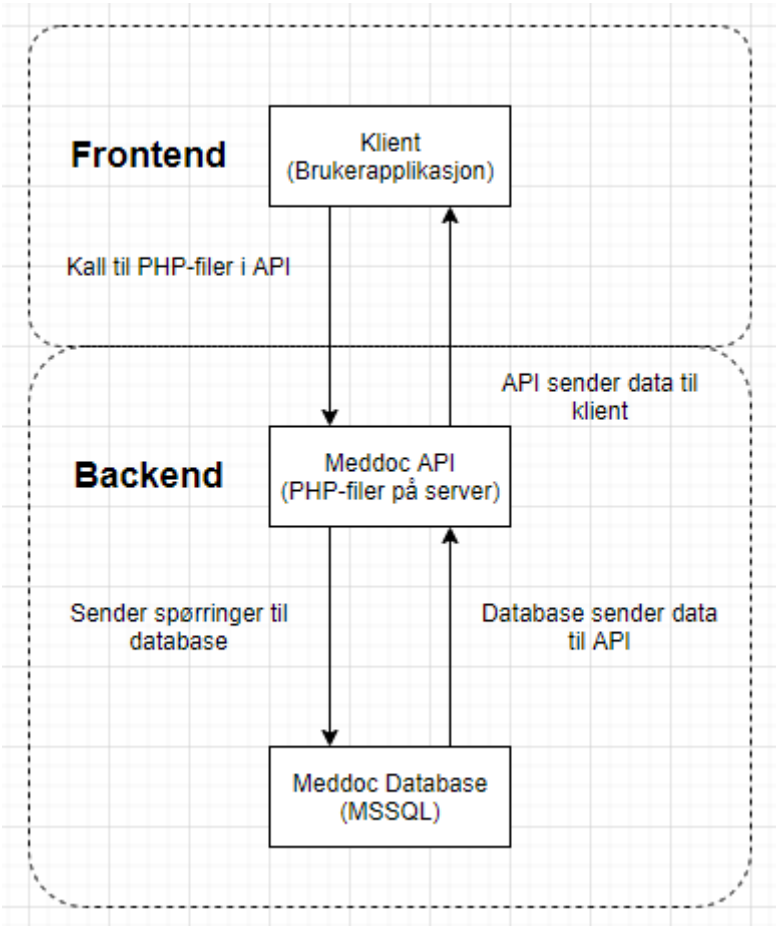
Her vil det gis et grundigere og mer detaljert innblikk i hvordan applikasjonen er bygd opp samt hvordan funksjoner fungerer. Applikasjonens frontend vil gjennomgås og det blir forklart hvordan filene i backend opererer mot databasen.

3.3.1 Systemet som helhet

Systemet mitt består av to versjoner applikasjoner, php-filer som utgjør et API på server og tabeller i Meddoc sin database. Med unntak av selve databasesystemet, er alt utviklet fra bunnen av. Applikasjonene sender data via php-filene som kobler seg opp mot databasen. Deretter sendes data tilbake fra databasen samme vei gjennom php-filene og ender opp i brukerens applikasjon.

3.3.1.1 Systemmodell

Jeg viser igjen til systemmodellen fra avsnitt **2.2.2.1**, som viser hvordan systemet henger sammen.



Figur 3.3.1.1: Systemmodell

Systemet mitt består hovedsakelig av to områder. Dette er en frontend for brukeren (applikasjonen), backend i form av php-filene på serveren (API) og i form av databasen.

Frontend [30] består av applikasjonene for android- og iOS-enheter. Alt brukerne ser og gjør, skjer her. Applikasjonene sender data ved å bruke JSON-forespørsler (JavaScript Object Notation). JSON er et tekstbasert format som brukes for å sende data i form av objekter.

Backend [31] består av php-filene som utgjør et API og databasesystemet. I API'et blir JSON-objektene tatt imot og videresendt til databasen. Det er slik alle endringer blir gjort mot databasen. Dersom det er forespurt, blir data sendt tilbake fra databasen via API'et og videre til brukerens applikasjon.

3.3.2 Applikasjon frontend

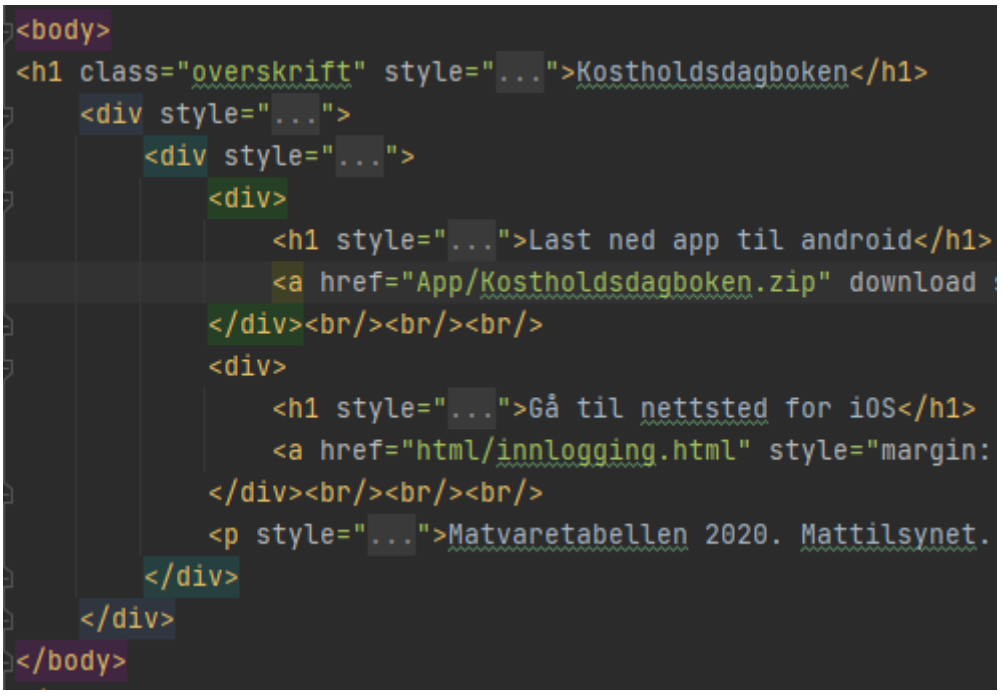
Her vil jeg gi en bedre innsikt i hva applikasjonen inneholder og hvordan den fungerer, dette med eksempler fra kode og bilder. Jeg vil vise dette for begge versjoner av applikasjonen, da det er noen forskjeller grunnet ulike programmeringsspråk, og elementer innenfor disse. Der begge versjonene vises, er android til venstre og iOS til høyre, og ellers vil jeg spesifisere hvilken versjon det er snakk om.

3.3.2.1 Startside

Startsiden er det første brukeren ser når vedkommende bruker applikasjonen. Hit kommer man ved å følge en link som alle brukerne vil få på mail. Her har man to store knapper, hvor man har mulighet til å velge den versjonen som passer sin enhet. Nederst på siden er det også linket til matvaretabellen [4] hvor alle matvarene er hentet fra.

Figur 3.3.2.1: Skjermdump fra startsiden

Ved å trykke på android-ikonet får man forespørsel om å laste ned en fil. Dette er en zippet mappe med en apk-fil (android application package). Dette er en filtype for android-applikasjoner. Denne kan så installeres på mobilen, og kjøres som en vanlig app. Ved å trykke på iOS-ikonet går man videre til innloggingssiden for iOS-versjonen. Figur **3.3.2.2** viser hvordan startsiden er satt opp.



Figur 3.3.2.2: Kodeutsnitt av startsiden

3.3.2.2 Innlogging

På innloggingssiden kan bruker skrive inn brukernavn og passord i sine respektive felter. *Logg inn*-knappen opprettes i JavaScript og krypterer det inntastede passordet før funksjonen *loggInn* kjøres. Denne funksjonen tar inn brukernavn og passord som brukeren har skrevet inn, og sjekker om begge stemmer med en bruker som er lagret i databasen.

```java
public static String encrypt(String strToEncrypt) {
    try {
        byte[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
        IvParameterSpec ivspec = new IvParameterSpec(iv);

        SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
        KeySpec spec = new PBEKeySpec(SECRET_KEY.toCharArray(), SALT.getBytes(),
                iterationCount: 65536, keyLength: 256);
        SecretKey tmp = factory.generateSecret(spec);
        SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), algorithm: "AES");

        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivspec);
        return Base64.getEncoder()
                .encodeToString(cipher.doFinal(strToEncrypt.getBytes(StandardCharsets.UTF_8)));
    } catch (Exception e) {
        System.out.println("Feil ved kryptering: " + e.toString());
    }
    return null;
}
```

Figur 3.3.2.3: Funksjon som krypterer passordet for android

I figur **3.3.2.3** ser vi funksjonen for kryptering av passordet. Den tar inn brukerinntastet passord og returnerer en kryptert tekststreng. Krypteringsmetoden er SHA256 som er mer beskrevet i avsnitt **3.3.4** om *passord*.

```javascript
const button = $('<button/>').html("Logg inn").on('click', function () {
    const crypto = require('crypto');

    const algorithm = 'aes-256-cbc';
    const secretKey = [redacted]
    const salt = [redacted]

    const key = crypto.pbkdf2Sync(secretKey, salt, 65536, 32, 'sha256');
    const iv = Buffer.from([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]);

    const cipher = crypto.createCipheriv(algorithm, key, iv);
    let encrypted = cipher.update($("#passord").val(), 'utf-8', 'base64');
    encrypted += cipher.final('base64');

    loggInn($("#brukernavn").val(), encrypted);
});

$("#btnlogginn").html(button);
```

Figur 3.3.2.4: Funksjon for opprettelse av "Logg inn"-knapp for iOS

I figur **3.3.2.4** blir innloggingsknappen opprettet. Her blir passordet kryptert før det sendes til *loggInn*-funksjonen som også tar inn brukernavnet. Krypteringsmetoden er igjen SHA256 som er mer beskrevet i avsnitt **3.3.4** om *passord*.

```
for (Bruker bruker : brukere) {
    if (b.equals(String.valueOf(bruker.getId())) &&
            String.valueOf(AES256.encrypt(p)).equals(bruker.getPassord())) {
        match = true;
        brukernavn = b;
        passord = p;
        editor.putString("brukernavn", brukernavn);
        editor.putString("passord", passord);
        editor.putString("brukerID", bruker.getId());
        editor.apply();
        if (nyttPassord) {
            tilNyttPassord();
        }
        else {
            loggInn();
        }
        break;
    }
}
```

Figur 3.3.2.5: Funksjonene som sjekker innloggingsinformasjonen for android

```
for (let i=0; i<brukere.length; i++) {
    if (b === brukere[i].brukerid && p === brukere[i].passord) {
        gyldig = true;
        feilB.css("display","none");
        feilP.css("display","none");
        feilBP.css("display","none");
        break;
    }
    else {
        feilBP.css("display","block");
    }
}
```

Figur 3.3.2.6: Funksjonene som sjekker innloggingsinformasjonen for iOS

I figur **3.3.2.5** og **3.3.2.6** går funksjonene gjennom alle brukerne i databasen og sjekker om det inntastede brukernavnet og det krypterte passordet stemmer med en eksisterende bruker. Hvis en bruker stemmer overens med

innloggingsinformasjonen, lagres brukernavn og passord med *shared preferences* [5], en måte å lagre nøkkel/verdi-par på enheten. Dette gjøres i tilfelle brukeren ønsker å endre passordet sitt etter innlogging.

```
sessionStorage.setItem("brukerID",bnavn);
location.replace( url: "oversikt.html");
```

Figur 3.3.2.7:

```
public void loggInn() {
    Intent intent = new Intent( packageContext: this, MainActivity.class);
    startActivity(intent);
    finish();
}
```

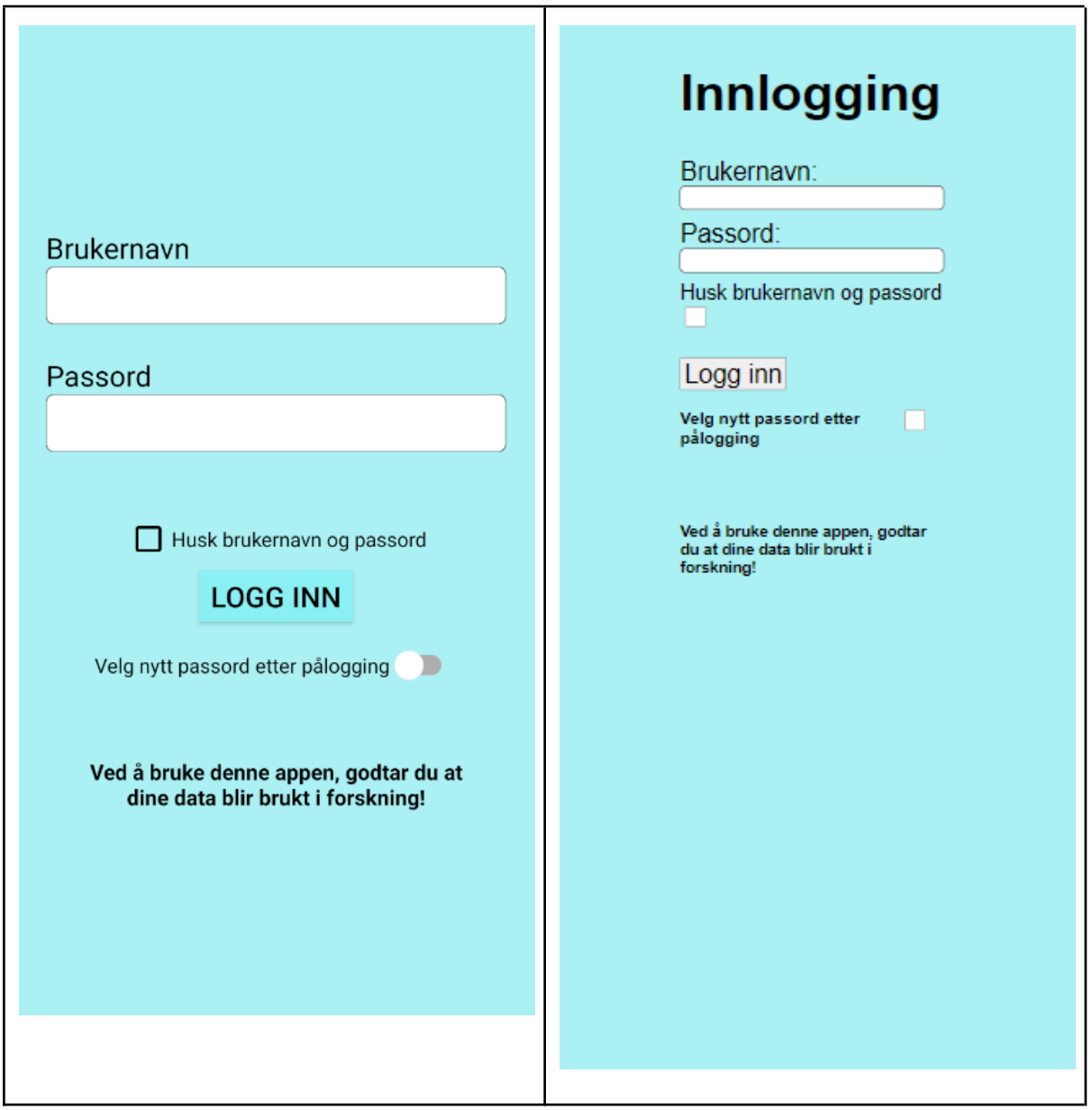
Figur 3.3.2.8:

```
if (byttPassord) {
    sessionStorage.setItem("brukerID",bnavn);
    location.replace( url: "nyttpassord.html");
}
```

Figur 3.3.2.9:

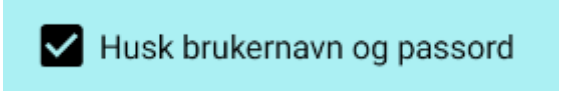
```
public void tilNyttPassord() {
    Intent intent = new Intent( packageContext: this, NyttPassord.class);
    startActivity(intent);
    finish();
}
```

Figur 3.3.2.10:

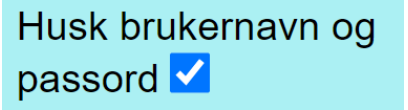


Figur 3.3.2.11: Innloggingssiden

Brukeren har mulighet til å la applikasjonen huske innloggingsinformasjonen for seg. Ved å huke av i sjekkboksen *Husk brukernavn og passord* lagres informasjonen lokalt på brukerens telefon eller nettleser.



Figur 3.3.2.12: Viser avhuking for å la appen huske innloggingsinformasjon for android



Figur 3.3.2.13: Viser avhuking for å la appen huske innloggingsinformasjon for iOS

```
cbHusk.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            husk = true;
            editor.putString("brukernavn", brukernavn);
            editor.putString("passord", passord);
        }
        else {
            husk = false;
            editor.putString("brukernavn", "");
            editor.putString("passord", "");
        }
        editor.putBoolean("husk", husk);
        editor.apply();
    }
});
```

Figur 3.3.2.14: Funksjonen som husker innloggingsinformasjonen for android

I figur **3.3.2.14** ser vi en funksjon som blir kjørt dersom sjekkboksen blir endret, om den blir valgt eller ikke. Den tar inn en boolsk variabel (en variabel som kan ha to verdier, enten sant eller usant) som forteller om sjekkboksen er valgt eller ikke. Hvis den er valgt, lagres brukernavn og passord gjennom shared preferences. En annen egendefinert boolsk variabel kalt *husk* blir satt til å være *sann* og lagres også. Hvis sjekkboksen ikke er valgt, blir innloggingen satt til å være tom, og *husk*-variabelen blir satt til *usann*.

```
if (document.getElementById( elementId: "cbHusk").checked) {
    localStorage.setItem("husk", "true");
    localStorage.setItem("huskB", $("#brukernavn").val());
    localStorage.setItem("huskP", $("#passord").val());
}
else {
    localStorage.clear();
}
```

Figur 3.3.2.15: Funksjonen som husker innloggingsinformasjonen for iOS

I figur **3.3.2.15** sjekkes det om sjekkboksen er valgt. Hvis den er det, lagres innloggingsinformasjonen i local storage (lokal lagring), og som i forrige figur blir en egendefinert boolsk variabel *husk* satt til å være sann.

```
husk = prefs.getBoolean( key: "husk",  defValue: false);
if (husk) {
    cbHusk.setChecked(true);
    brukernavn = prefs.getString( key: "brukernavn",  defValue: "");
    passord = prefs.getString( key: "passord",  defValue: "");
    etBrukernavn.setText(brukernavn);
    etPassord.setText(passord);
}
```

Figur 3.3.2.16: Funksjonen som sjekker om innloggingsinformasjon skal fylles ut av app for android

I figur **3.3.2.16** sjekkes det hva den boolske variabelen *husk* er satt til. Hvis den er satt til *sann*, fylles innloggingsfeltene med den informasjonen som er lagret.

```
husk = localStorage.getItem( key: "husk");
if (husk === "true") {
    $("#cbHusk").prop('checked', true);
    $("#brukernavn").val(localStorage.getItem( key: "huskB"));
    $("#passord").val(localStorage.getItem( key: "huskP"));
}
```

Figur 3.3.2.17: Funksjonen som sjekker om innloggingsinformasjon skal fylles ut av app for iOS

I figur **3.3.2.17** sjekkes det også om variabelen *husk* er satt til å være *sann*. Her fylles feltene i så fall ut med informasjonen som er lagret lokalt.

Feilmeldinger

Dersom det skulle forekomme en feil, enten ved at bruker ikke skriver riktige innloggingsopplysninger eller at det er noe feil med serveren, er det viktig at bruker blir informert om dette.

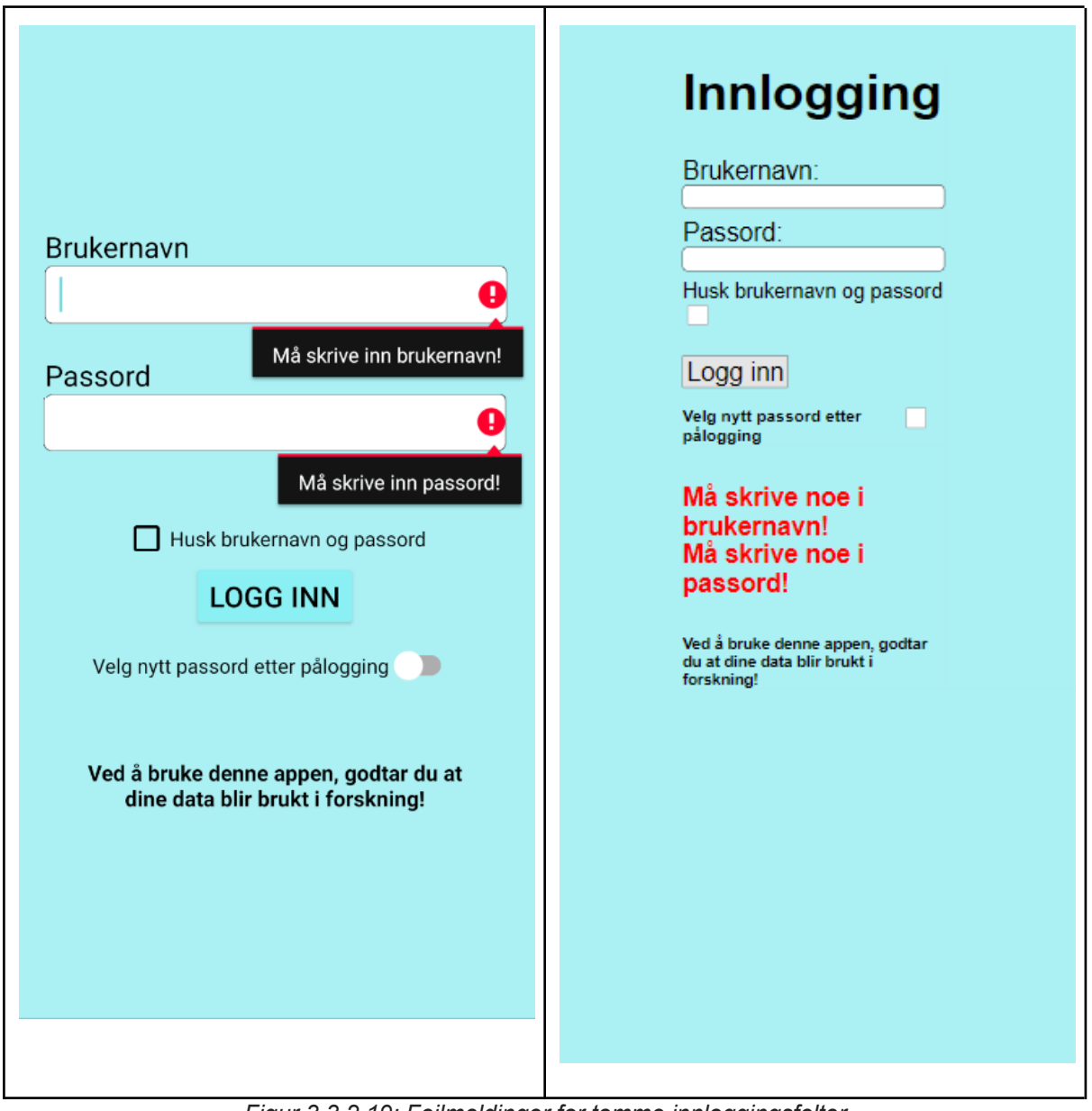
Første tilfelle er dersom brukeren prøver å logge inn uten å ha gitt noen opplysninger.

```
if (b.equals("") || p.equals("")) {
    if (b.equals("")) {
        etBrukernavn.setError("Må skrive inn brukernavn!");
    }
    if (p.equals("")) {
        etPassord.setError("Må skrive inn passord!");
    }
} else {
```

```
if (b === "" || $("#passord").val() === "") {
    if (b === "") {
        feilB.css("display","block");
    }
    if ($("#passord").val() === "") {
        feilP.css("display","block");
    }
}
```

Figur 3.3.2.18: Funksjonene sjekker om noen av feltene er tomme

I figur **3.3.2.18** sjekkes det først om noen av innloggingsfeltene er tomme. Dersom minst én av de er det, sjekkes det videre hvilken eller hvilke som er tomme. Hvis et felt er tomt, vises en beskrivende feilmelding. Resultatet blir som vist under:



Figur 3.3.2.19: Feilmeldinger for tomme innloggingsfelter

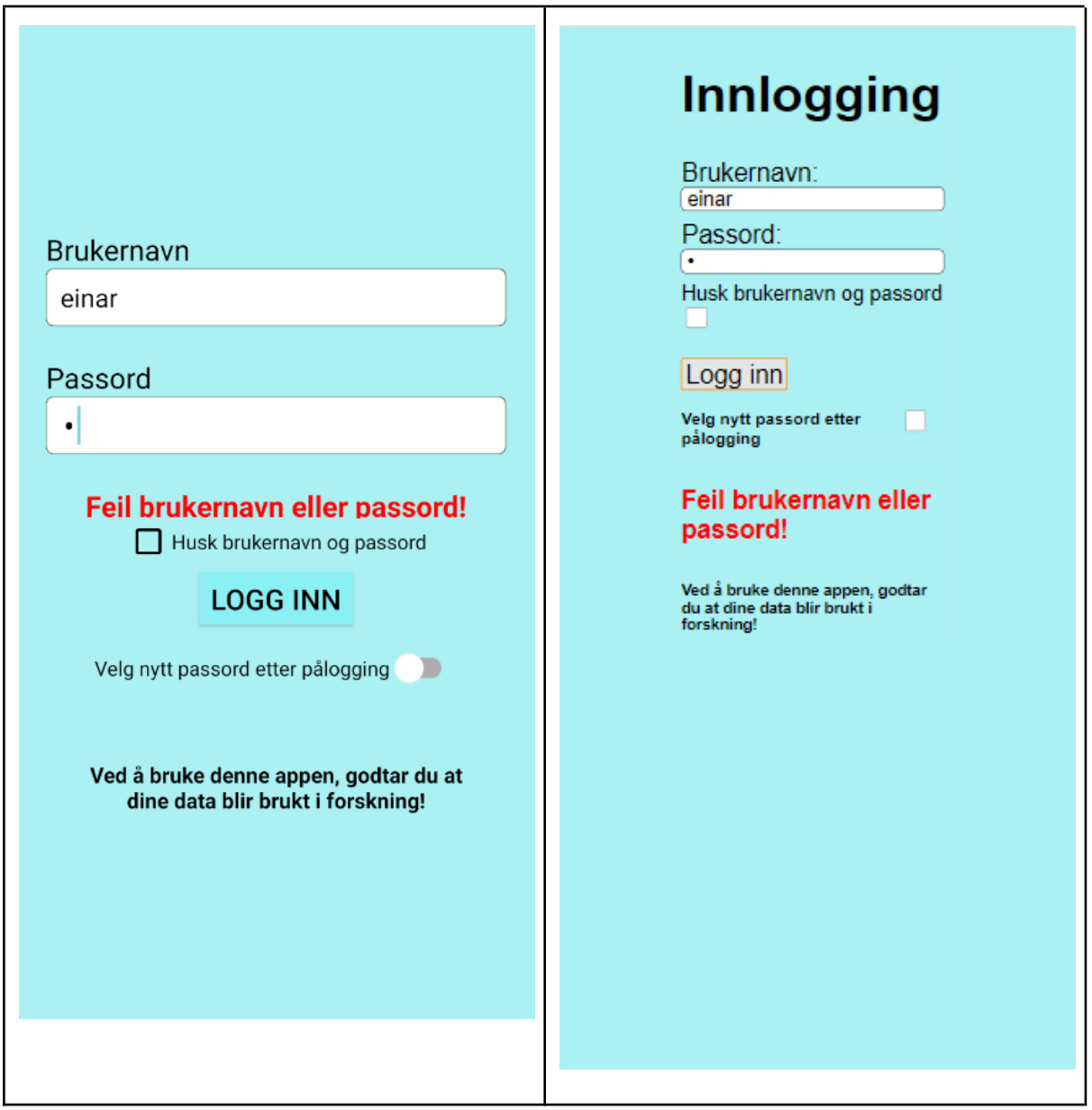
Neste tilfelle er hvis brukeren har skrevet inn innloggingsopplysninger som ikke stemmer overens med en bruker som er registrert i databasen.

```
if (!match) {
    tvFeil.setVisibility(View.VISIBLE);
}
```

```
else {
    feilBP.css("display","block");
}
```

Figur 3.3.2.20: Funksjonene sjekker om opplysningene stemmer

I figur **3.3.2.20** sjekker funksjonene om opplysningene er gyldige innloggingsopplysninger. Disse funksjonene er de nederste delene fra funksjonene vist i figurene **3.3.2.5** og **3.3.2.6**. Hvis opplysningene ikke stemmer, vises feilmeldingen som vist under:



Figur 3.3.2.21: Funksjonen sjekker om det finnes brukere registrert på serveren for iOS

Ved feil på serveren eller hvis det ikke kan oppnås forbindelse med serveren, vil brukeren få opp en varselboks.

```
if (brukere.size() == 0) {
    new AlertDialog.Builder( context: this)
            .setTitle("Feil på server")
            .setMessage("Prøv igjen senere")
            .setNeutralButton( text: "OK", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {

                }
            })
            .setIcon(android.R.drawable.ic_dialog_alert)
            .show();
}
```

Figur 3.3.2.22: Funksjonen sjekker om det finnes brukere registrert på serveren for android

```
if (brukere.length === 0) {
    alert("Feil på server! Prøv igjen senere...")
}
```

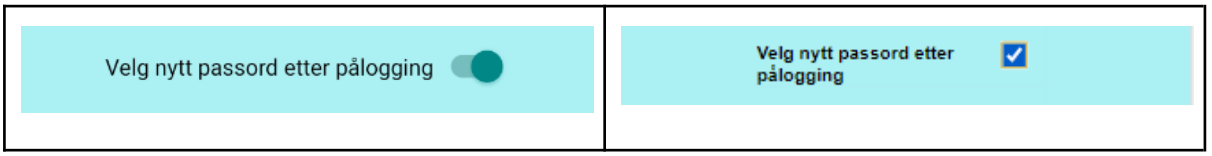
Figur 3.3.2.23: Funksjonen sjekker om det finnes brukere registrert på serveren for iOS

I figurene **3.3.2.22** og **3.3.2.23** sjekker funksjonene om det eksisterer brukere registrert på serveren. Hvis det ikke finnes noen brukere skal det vises en varsel-dialog med info om dette. Jeg valgte å sjekke om antall brukere er 0 for dersom man ikke får forbindelse med serveren, finner man heller ingen brukere. Ettersom jeg ønsket å sjekke server-forbindelsen når man prøver å logge inn, valgte jeg å sjekke om det eksisterer noen brukere der. Dersom man får forbindelse med serveren, men applikasjonen ikke klarer å hente brukerne, er dette også en feil. Varsel-dialogene ser slik ut:

Figur 3.3.2.24: Varsel-dialoger for feil på serveren

3.3.2.3 Nytt passord

Brukeren kan som nevnt velge et nytt passord etter innlogging. Dette gjøres ved å huke av for "*Velg nytt passord etter pålogging*" før man trykker på innloggingsknappen:



Figur 3.3.2.25: Viser avhuket valg for nytt passord

Hvis brukeren huker av for å endre passord slik det er vist i figur **3.3.2.25**, utføres følgende funksjoner:

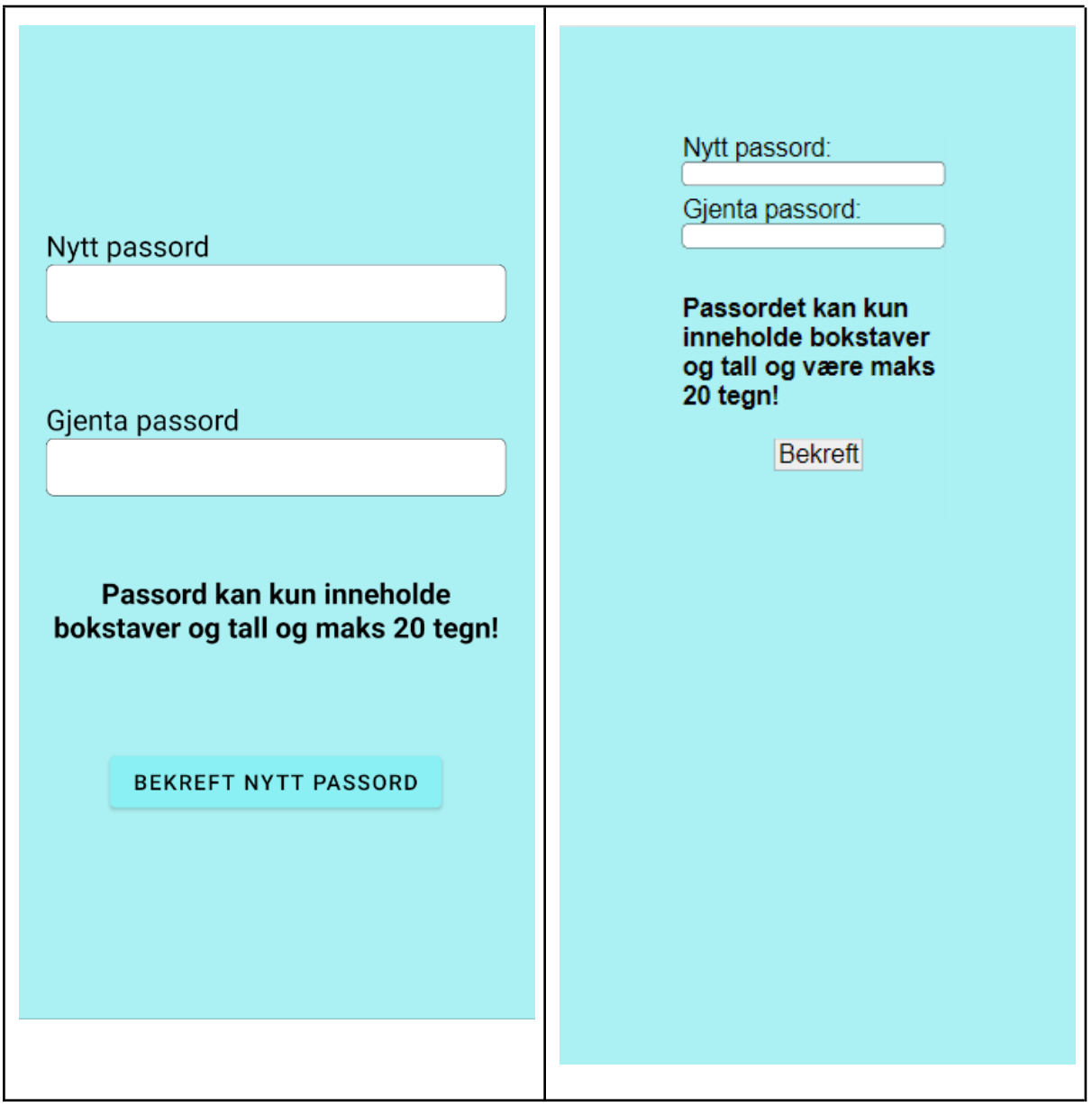
```
public void tilNyttPassord() {
    Intent intent = new Intent( packageContext: this, NyttPassord.class);
    startActivity(intent);
    finish();
}
```

Figur 3.3.2.26: Funksjonen åpner "Nytt passord"-siden for android

```
if (byttPassord) {
    sessionStorage.setItem("brukerID",bnavn);
    location.replace( url: "nyttpassord.html");
}
```

Figur 3.3.2.27: Funksjonen åpner "Nytt passord"-siden for iOS

I figurene **3.3.2.26** og **3.3.2.27** vises funksjonene som kjøres dersom brukeren har skrevet inn gyldig innloggingsinformasjon og har huket av for å bytte passord. Selve sjekken som finner ut om det skal byttes passord, er satt opp likt som funksjonene for å huske innlogging vist i figur **3.3.2.14** og **3.3.2.15**. Funksjonene åpner så en ny side i applikasjonen:



Figur 3.3.2.28: Viser "Nytt passord"-siden

Figur **3.3.2.28** viser utseendet til "*Nytt passord*"-siden. Her må brukeren skrive inn nytt ønsket passord to ganger, og deretter trykke *Bekreft*. Begrensningene for passordet er skrevet under feltene.

```
public void bekreft(String passord) {
    nyttPassord = passord;
    String url = urlP + "/?BrukerID=" + brukerID + "&Passord=" + (AES256.encrypt(nyttPassord))
            .replace( target: "+", replacement: "%2B");
    System.out.println(url);

    setPassordJSON task = new setPassordJSON();
    task.execute(url);
}
```

Figur 3.3.2.29: Funksjonen håndterer det nye passordet for android

```
function bekreft(passord) {
    nyttPassord = passord;
    const url = urlP + "/?BrukerID=" + brukerID + "&Passord=" + passord.replace("+","%2B");
    $.post( url, function (data) {
        //Kode her
        localStorage.setItem("huskP", $("#nyttPassord").val());
        location.replace( url: "innlogging.html");
    });
}
```

Figur 3.3.2.30: Funksjonen håndterer det nye passordet for iOS

I figurene **3.3.2.29** og **3.3.2.30** krypterer funksjonene det nye passordet før det sendes til en php-fil på server som kobler seg til databasen og erstatter det gamle passordet. Dette gjøres gjennom et JSON.post-kall, hvor brukernavnet og det krypterte passordet sendes sammen med url'en til php-filen. Når passordet er oppdatert, sendes brukeren tilbake til innloggingssiden igjen.

Funksjonen *.replace("+", "%2B")* gjør at eventuelle plusstegn (+) i det krypterte passordet blir gjort om til "%2B" (ASCII-verdi av plusstegnet). Dette gjøres fordi url'en kun leser ASCII-formatert tekst. Plusstegnet er ikke inkludert i ASCII-settet og må derfor gjøres om for å kunne bli tolket riktig. [7]

Feilmeldinger

Ved setting av nytt passord kan det også forekomme feil. Igjen er det viktig at brukeren får relevante feilmeldinger om noe er feil.

Første tilfelle er som for innloggingen, at brukeren prøver å bekrefte uten å ha skrevet inn noe i feltene.

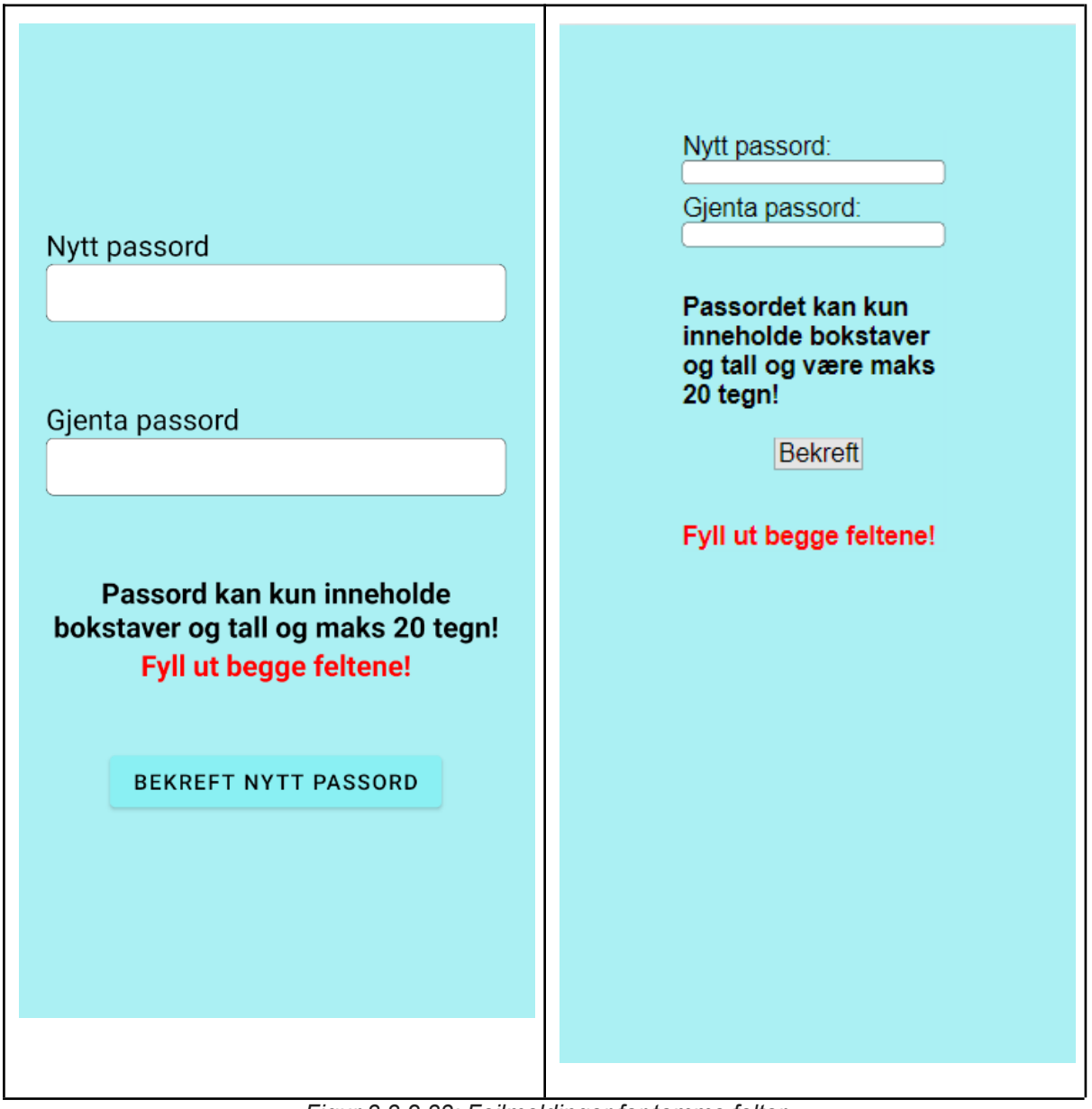
```
if (String.valueOf(etNyttPassord.getText()).equals("")
        || String.valueOf(etGjentaPassord.getText()).equals("")) {
    tvTom.setVisibility(View.VISIBLE);
    tvPassordStemmerIkke.setVisibility(View.INVISIBLE);
    tvPassordIkkeGyldig.setVisibility(View.INVISIBLE);
}
```

Figur 3.3.2.31: Funksjonen sjekker om det er skrevet noe inn i feltene for android

```
if (nyttPassord.val() === "" || gjentaPassord.val() === "") {
    $("#ugyldigPassord").css("display", "none");
    $("#tomPassord").css("display","block");
    $("#ulikePassord").css("display","none");
}
```

Figur 3.3.2.32: Funksjonen sjekker om det er skrevet noe inn i feltene for iOS

I figurene **3.3.2.31** og **3.3.2.32** sjekker funksjonene om det er skrevet noe inn i feltene. Hvis minst ett av feltene er tomme vises en feilmelding som ber brukeren fylle ut begge felter:



Figur 3.3.2.33: Feilmeldinger for tomme felter

Neste tilfelle er at brukeren har skrevet inn to forskjellige passord.

```
else if (likePassord) {
    tvTom.setVisibility(View.INVISIBLE);
    tvPassordStemmerIkke.setVisibility(View.INVISIBLE);
    tvPassordIkkeGyldig.setVisibility(View.INVISIBLE);
    bekreft(String.valueOf(etNyttPassord.getText()));
}
else {
    tvTom.setVisibility(View.INVISIBLE);
    tvPassordStemmerIkke.setVisibility(View.VISIBLE);
    tvPassordIkkeGyldig.setVisibility(View.INVISIBLE);
}
```

```
else if (likePassord) {
    $("#ugyldigPassord").css("display", "none");
    $("#tomPassord").css("display","none");
    $("#ulikePassord").css("display","none");

    bekreft(kp);
}
else {
    $("#ugyldigPassord").css("display", "none");
    $("#tomPassord").css("display","none");
    $("#ulikePassord").css("display","block");
}
```

Figur 3.3.2.34: Funksjonene sjekker om passordene er like

I figur **3.3.2.34** sjekker funksjonene om de to passordene som er skrevet inn er like. Er de det, kjøres *bekreft*-funksjonen vist i figur **3.3.2.30**. Hvis disse ikke stemmer vises følgende feilmeldinger:

Figur 3.3.2.35: Feilmeldinger for ulike passord

Siste tilfelle er at passordet som blir skrevet inn ikke stemmer med passord-betingelsene.

```
else if (!String.valueOf(etNyttPassord.getText()).matches(regex)
        || !String.valueOf(etGjentaPassord.getText()).matches(regex)) {
    tvTom.setVisibility(View.INVISIBLE);
    tvPassordStemmerIkke.setVisibility(View.INVISIBLE);
    tvPassordIkkeGyldig.setVisibility(View.VISIBLE);
}
```

Figur 3.3.2.36: Funksjonen sjekker om passordet er gyldig for android

```
else if (!gyldigPassord) {
    $("#ugyldigPassord").css("display", "block");
    $("#tomPassord").css("display","none");
    $("#ulikePassord").css("display","none");
}
```

Figur 3.3.2.37: Funksjonen sjekker om passordet er gyldig for iOS

I figurene **3.3.2.36** og **3.3.2.37** sjekker funksjonene om passordet stemmer med passord-betingelsene. Disse er definert ved hjelp av en *RegEx* [6]. I mitt tilfelle ser den slik ut:

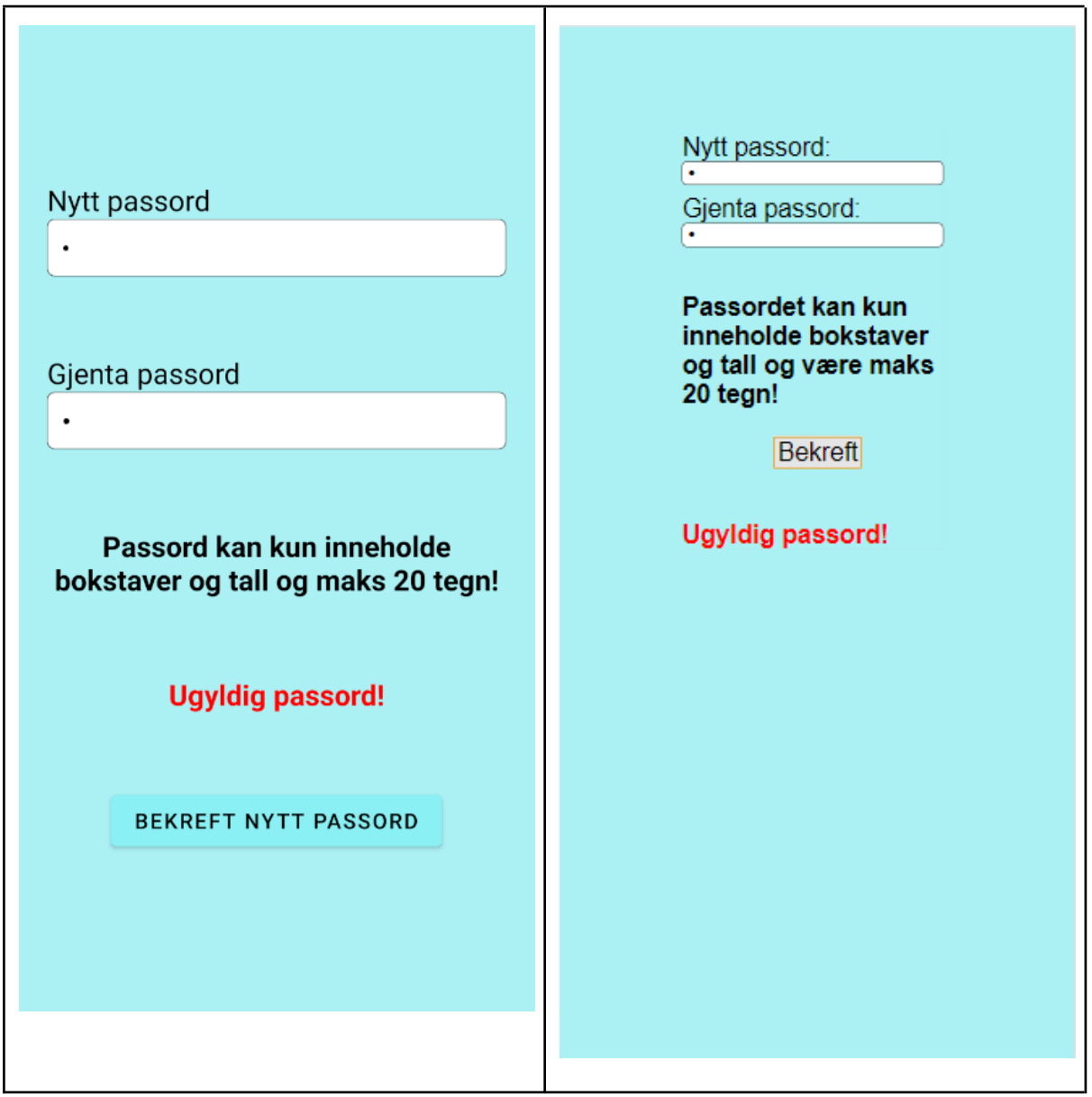
```
String regex = "^[a-zA-Z0-9ÆØÅæøå]{0,20}+$";
```

```
(/^([A-Za-z0-9ÆØÅæøå]{0,20})$/
```

Figur 3.3.2.38: RegEx for passord

Denne RegEx'en sier at passordet skal tillate tegnene a-z i små bokstaver (a-z), A-Z i store bokstaver (A-Z), tallene fra 0-9 (0-9), spesialtegnene æ, ø og å med små og store bokstaver (ÆØÅæøå), være maks 20 tegn langt ({0, 20}).

Dersom passordet ikke er gyldig i henhold til RegEx'en, vises følgende feilmeldinger:



Figur 3.3.2.39: Feilmeldinger for ugyldige passord

Dersom det nye passordet er gyldig, kommer brukeren tilbake til innloggingssiden. Brukeren kan umiddelbart logge på med det nye passordet.

3.3.2.4 Oversikt

Etter vellykket innlogging kommer brukeren til *oversiktssiden*. Her vil tidligere registreringer listes opp med dato og klokkeslett for når registreringen skjedde. Dette for at brukerne skal være sikre på at registreringen ble sendt, slik at de ikke registrerer dobbelt.

Figur 3.3.2.40: oversiktssiden

I figur **3.3.2.40** ser vi at nederst til høyre på oversiktssiden, er det en knapp. Ved å trykke på denne kjøres følgende funksjon:

```
Intent intent = new Intent( packageContext: MainActivity.this, Registrering.class);
startActivity(intent);
finish();
```

Figur 3.3.2.41: Funksjonen sender bruker til registrering for android

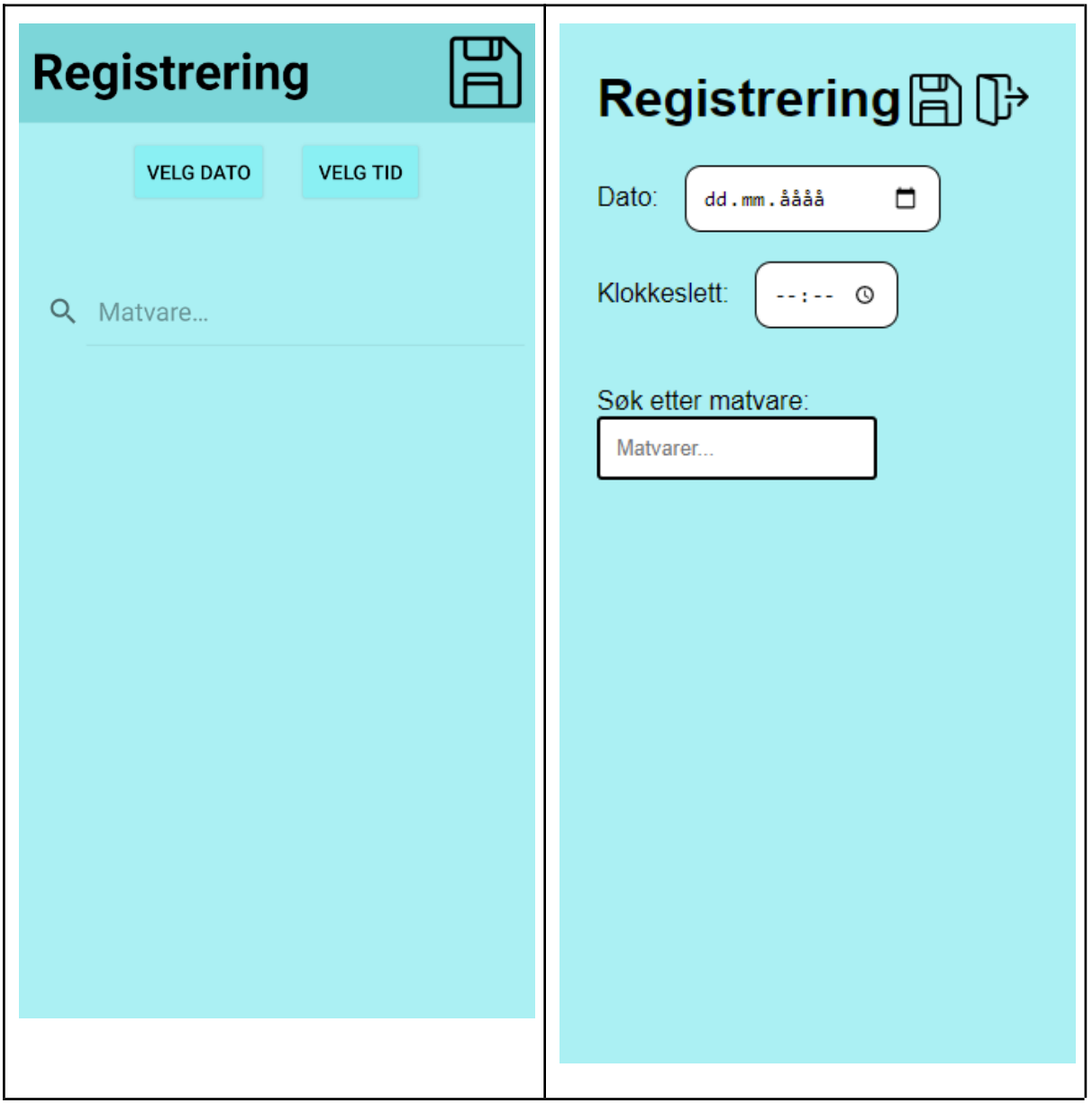
```
function registrer() {
    location.replace( url: "registrering.html");
}
```

Figur 3.3.2.42: Funksjonen sender bruker til registrering for iOS

I figurene **3.3.2.41** og **3.3.2.42** sender funksjonene brukeren videre til siden for registrering.

3.3.2.5 Registrering

Når brukeren trykker på registreringsknappen nevnt i **3.3.2.40**, kommer vedkommende til registreringssiden.



Figur 3.3.2.43: Registreringssiden

I figur **3.3.2.43** ser vi oppsettet for registreringssiden. Her er det en knapp for å lagre og et søkefelt for begge versjoner. For android er det enda to knapper for å velge dato og klokkeslett. For iOS er disse to inputfelter. I tillegg har iOS en egen "*gå*

tilbake”-knapp øverst til høyre. Dette var ikke nødvendig for android, da android-telefoner har en egen innebygd tilbakeknapp.

Dato og tid

Ved å trykke på knappene for dato og tid hos android, kjøres denne koden:

```
public void visDato() {
    DialogFragment datoFragment = new DatePickerFragment();
    datoFragment.show(getFragmentManager(), tag: "datePicker");
}

public void visTid() {
    DialogFragment tidFragment = new TimePickerFragment();
    tidFragment.show(getFragmentManager(), tag: "timePicker");
}
```

Figur 3.3.2.44: Funksjonene viser dialogbokser for android

I figur **3.3.2.44** lager funksjonene nye fragmenter (gjenbrukbar del av applikasjonens brukergrensesnitt) [28] for dato- og tid-dialogboksene og viser disse.

```java
public class DatePickerFragment extends DialogFragment
        implements DatePickerDialog.OnDateSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        final Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);

        return new DatePickerDialog(getActivity(), listener: this, year, month, day);
    }

    public void onDateSet(DatePicker view, int year, int month, int day) {
        month++;
        String dayStr;
        String monthStr;

        if (day < 10) {
            dayStr = "0" + day;
        }
        else {
            dayStr = String.valueOf(day);
        }
        if (month < 10) {
            monthStr = "0" + month;
        }
        else {
            monthStr = String.valueOf(month);
        }

        String dato= dayStr+"/"+monthStr+"/"+year;
        Button b = (Button) getActivity().findViewById(R.id.knappVelgDato);
        b.setText(dato);
        getActivity().findViewById(R.id.tvFeilDato).setVisibility(View.INVISIBLE);
    }
}
```

Figur 3.3.2.45: Funksjonene oppretter dato-dialogboks for android

I figur **3.3.2.45** ser vi klassen *DatePickerFragment* som er oppsettet for datovelgeren. Den returnerer dialogboksen som brukeren vil se. Før klassen returnerer denne, tar den inn nåværende årstall, måned og dag fra den lokale kalenderen og stiller kalenderen inn etter dette.

I funksjonen *onDateSet* ser vi det som skal skje når brukeren velger en dato. Først økes måneden med 1 (month++). Dette er fordi måned starter på 0 i Java, og ettersom jeg valgte å returnere datoen som tekst, er dette hensiktsmessig for at det blir riktig. Videre setter jeg en 0 foran dag og måned dersom disse er lavere enn 10. Dette er for at formatet skal bli likt hver gang.

Eksempel:
5. Mai 2021 skal være 05/05/2021 og ikke 5/5/2021.

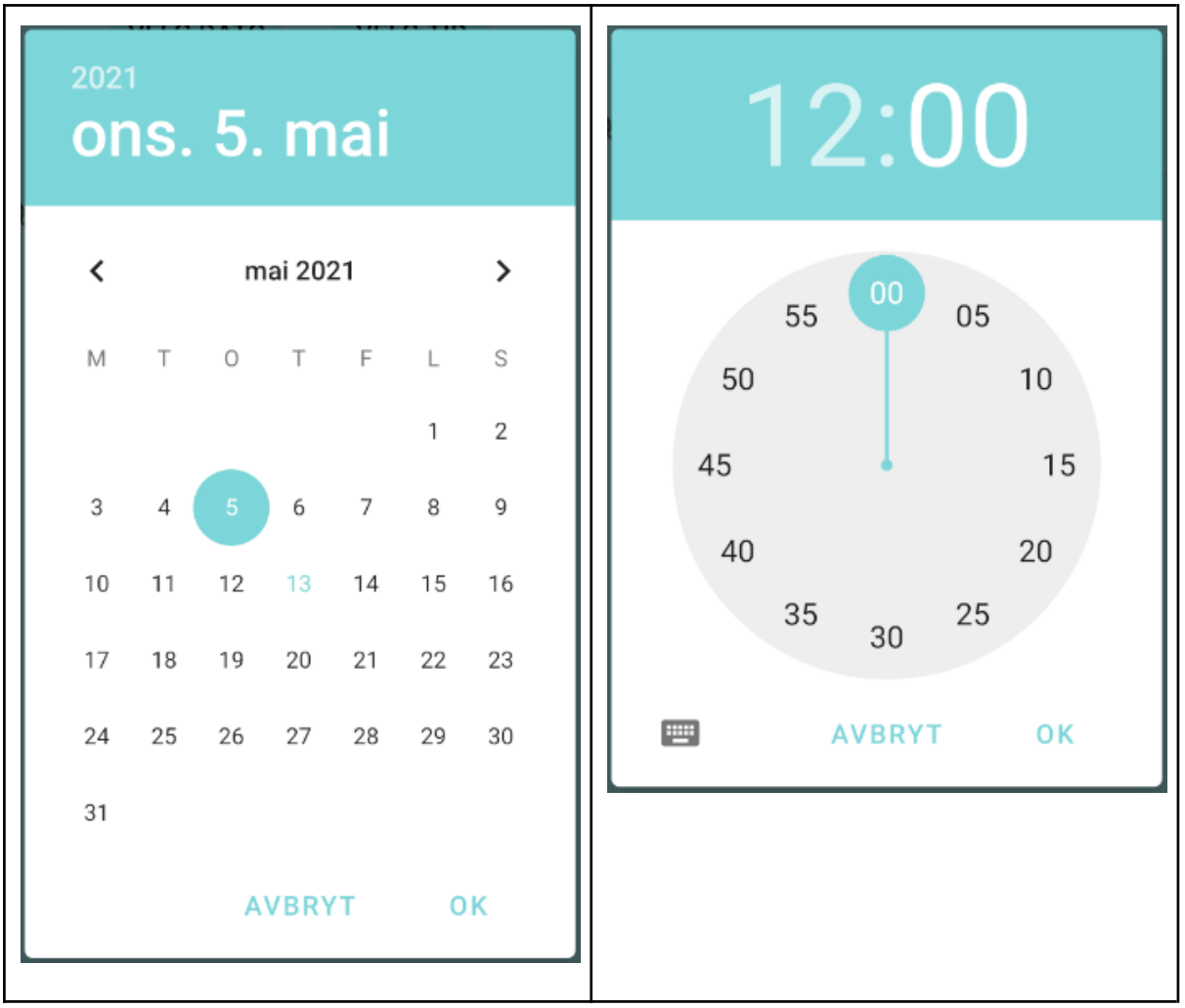
Til slutt finner vi knappen som skal få dato-teksten, og setter teksten til datoen som er valgt i formatet vist i eksempelet.

```
public class TimePickerFragment extends DialogFragment
        implements TimePickerDialog.OnTimeSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);


        return new TimePickerDialog(getActivity(), listener: this, hour, minute,
                DateFormat.is24HourFormat(getActivity()));
    }

    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        String minuteStr;
        String hourOfDayStr;

        if (minute < 10) {
            minuteStr = "0" + minute;
        }
        else {
            minuteStr = String.valueOf(minute);
        }
        if (hourOfDay < 10) {
            hourOfDayStr = "0" + hourOfDay;
        }
        else {
            hourOfDayStr = String.valueOf(hourOfDay);
        }

        String timeString = hourOfDayStr+":"+minuteStr;
        Button b = (Button) getActivity().findViewById(R.id.knappVelgTid);
        b.setText(timeString);
        getActivity().findViewById(R.id.tvFeilTid).setVisibility(View.INVISIBLE);
    }
}
```

Figur 3.3.2.46: Funksjonene oppretter tid-dialogboks for android

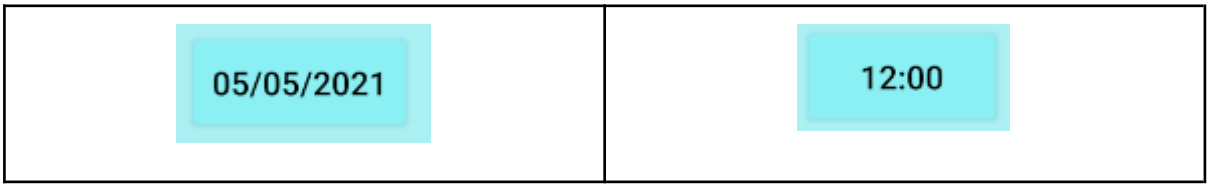
I figur **3.3.2.46** foregår omtrent det samme som for figur **3.3.2.45**. Forskjellen er at nå er det snakk om tid, ikke dato. Igjen henter klassen den lokale kalenderen, og timer og minutter fra denne. Jeg setter også klokke-formatet til 24 timer.

Funksjonen *onTimeSet* kjøres når brukeren har valgt ønsket tidspunkt. Som for dato, setter jeg en 0 foran timer eller minutter dersom disse er lavere enn 10. Knappen som skal få tid-teksten identifiseres og får teksten med tidsformatet *tt:mm*.



Figur 3.3.2.47: Dialogbokser for dato og tid for android

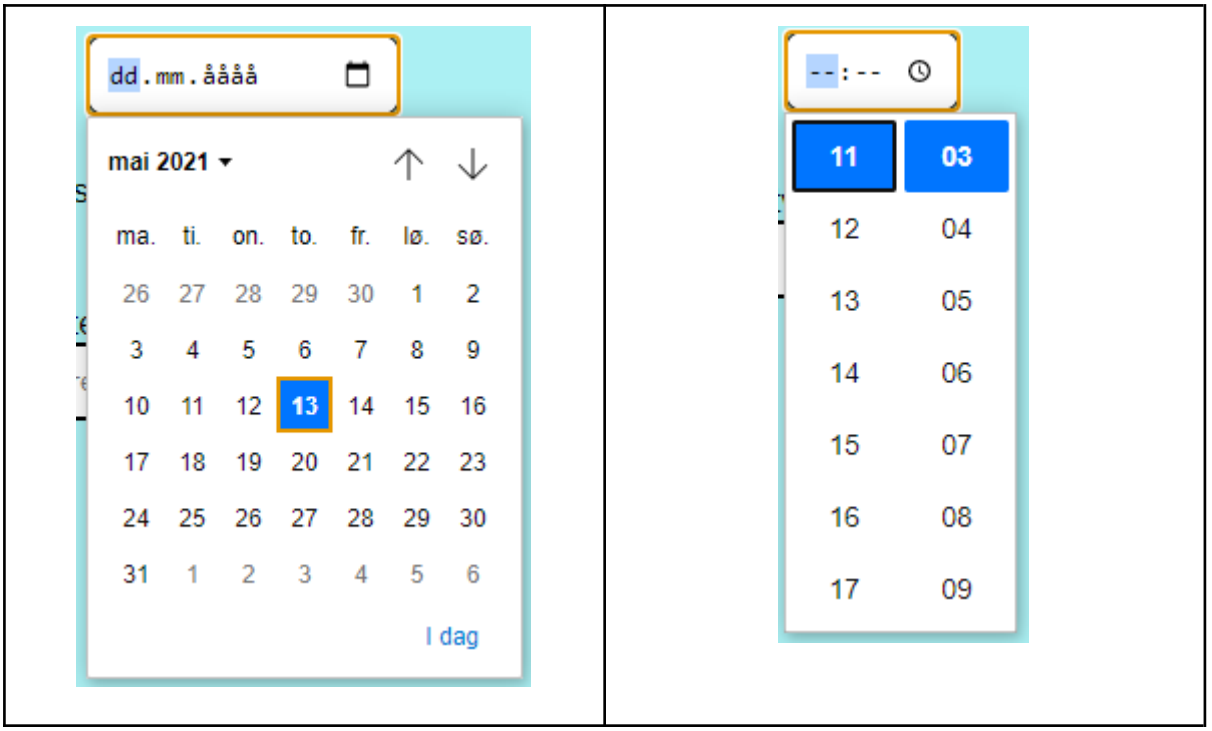
I figur **3.3.2.47** ser vi dialogboksene for dato og tid hos android, som også er standarden. Ved å trykke *OK* lagres dato og tid som tekst i deres respektive knapper.



Figur 3.3.2.48: Knappene for dato og tid etter dato og tid er valgt for android

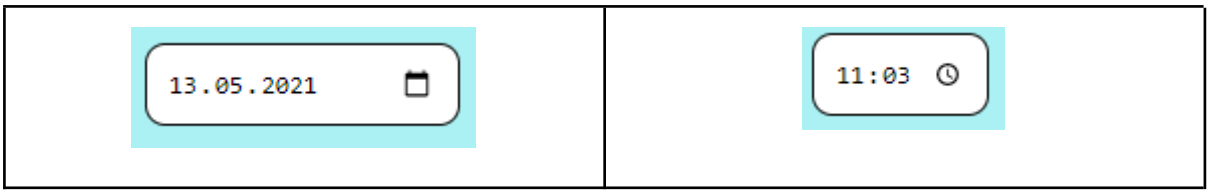
I figur **3.3.2.48** ser vi at valgene fra figur **3.3.2.47** har blitt lagret som tekst i knappene.

For iOS brukes inputfelter istedenfor knapper. Disse ser slik ut:



Figur 3.3.2.49: Inputfeltene for dato og tid for iOS

I figur **3.3.2.49** ser vi inputfeltene for dato og tid for iOS. Disse feltene er henholdsvis satt til typene *date* og *time*. De fungerer som et vanlig inputfelt, men man har også muligheten til å trykke på ikonet helt til høyre for å få opp kalender eller tidspunkter.



Figur 3.3.2.50: Inputfeltene for dato og tid etter dato og tid er valgt for iOS

I figur **3.3.2.50** ser vi feltene etter at dato og tid er valgt for iOS.

Søkefelt

Søkefeltet fungerer likt for både android og iOS. Her skriver du inn den matvaren du ønsker å registrere, og en liste med forslag vil dukke opp. Denne listen består av produkter som inneholder ordet du søker etter. Det er også lagt til en indikator på de

mest vanlige matvarene, slik at de vil komme høyere på listen enn mer uvanlige matvarer.

```
for (Matvare p : matvarer) {
    if (p.getMatvare().toLowerCase().contains(s.toLowerCase())) {
        filtrertListe.add(p);
    }
}

filtrertListe.sort(new Comparator<Matvare>() {
    @Override
    public int compare(Matvare o1, Matvare o2) {
        return Integer.compare(o1.getIndikator(), o2.getIndikator());
    }
});
```

Figur 3.3.2.51: Funksjonene legger til aktuelle matvarer i en liste og sorterer for android

I figur **3.3.2.51** legger den første funksjonen til alle aktuelle matvarer i forslagslisten. Dette er matvarer som inneholder søkeordet til brukeren. I den andre funksjonen sorteres den nye lista etter indikatoren som forteller om en matvare er mer vanlig enn en annen.

For iOS har jeg brukt eksempelkode fra w3schools [8]. Eksempelet deres fungerte omtrent slik jeg ønsket, og jeg tenkte det var bedre å ta det i bruk enn å skulle lage noe eget helt fra bunnen av. Jeg har derimot gjort noen modifikasjoner for at det skal fungere akkurat slik jeg ønsker.

I koden fra w3schools tar funksjonen inn teksten fra søkefeltet og et array (liste) med matvarene som skal søkes gjennom. Funksjonen sjekker gjennom matvarene om noen av dem inneholder teksten som bruker har skrevet inn i søkefeltet. Dersom de inneholder det, legges de relevante matvarene til i en ny liste som vises under søkefeltet.

```
    for (i = 0; i < arr.length; i++) {
        if (arr[i].toUpperCase().includes(val.toUpperCase())) {
            b = document.createElement( tagName: "DIV");
            b.innerHTML = arr[i].substr( from: 0, val.length);
            b.innerHTML += arr[i].substr(val.length);
            b.innerHTML += "<input type='hidden' value='" + arr[i] + "'>";
            b.addEventListener( type: "click", listener: function(e : MouseEvent ) {
                let verdi = this.getElementsByTagName("input")[0].value;
                inp.value = "";
                leggTilListe(verdi);
                closeAllLists();
            });
            a.appendChild(b);
        }
    }
});
```

Figur 3.3.2.52: Funksjonen legger aktuelle matvarer til i autofullføring for iOS

I figur **3.3.2.52** oppretter funksjonen et div-element for aktuelle matvarer og legger dette i en forslagsliste under søkefeltet dersom de inneholder teksten fra søkefeltet. Ved klikk på en matvare kalles funksjonen *leggTilListe* (se figur **3.3.2.57**) og *closeAllLists* (se figur **3.3.2.53**).

```
function closeAllLists(elmnt) {
    let x = document.getElementsByClassName( classNames: "autocomplete-items");
    for (let i = 0; i < x.length; i++) {
        if (elmnt != x[i] && elmnt != inp) {
            x[i].parentNode.removeChild(x[i]);
        }
    }
}
```

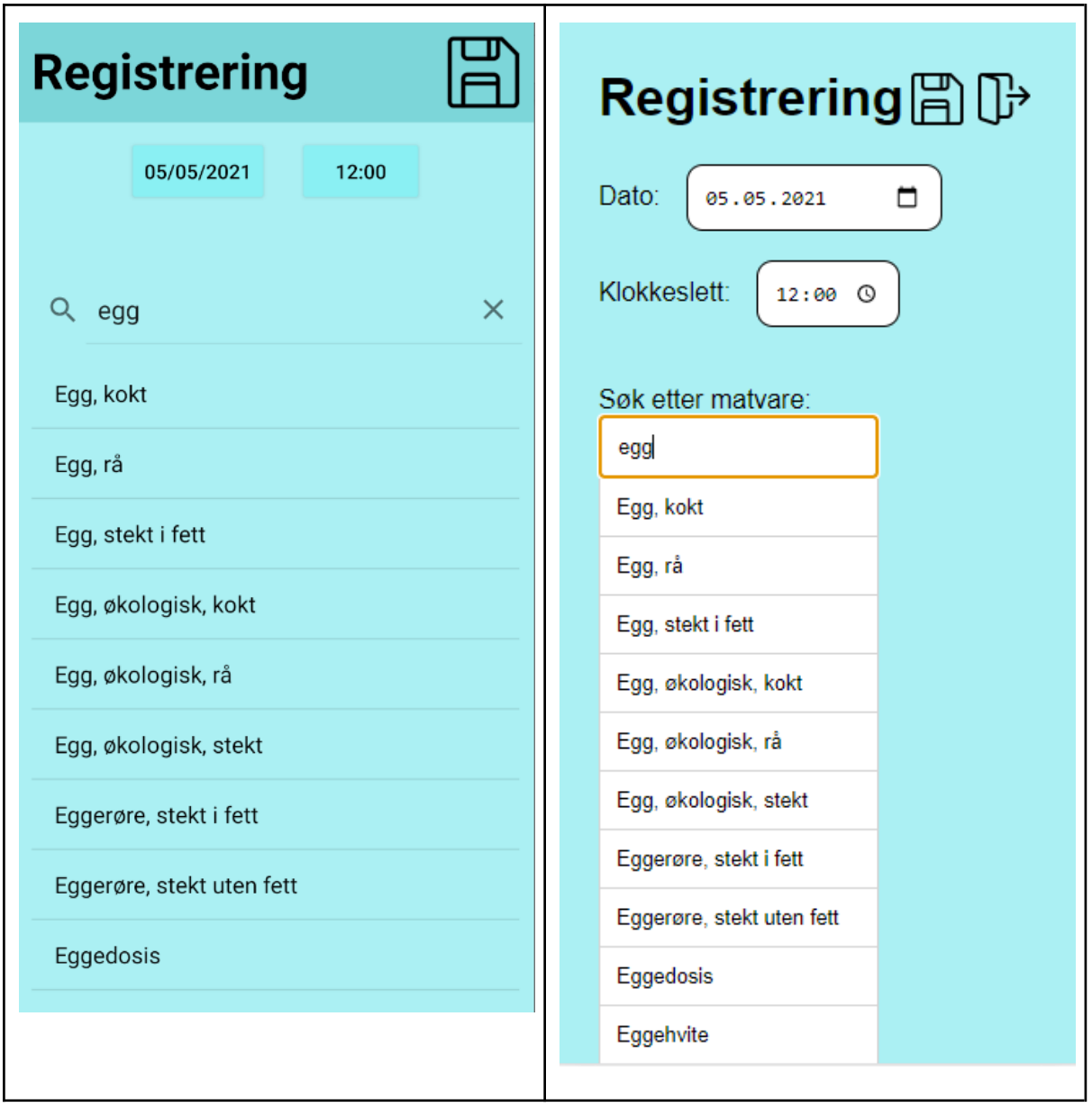
Figur 3.3.2.53: Funksjonen fjerner aktuelle matvarer når en er valgt for iOS

Funksjonen *closeAllLists* fjerner alle elementene fra forslagslista.

```
autocomplete(document.getElementById( elementId: "sokefelt"),
            produkter.sort( compareFn: function (a : string , b : string ) {
    indA = produkter.indexOf(a);
    indB = produkter.indexOf(b);
    return produktIndikatorer[indA] - produktIndikatorer[indB];
}));
```

Figur 3.3.2.54: Kall på autocomplete-funksjonen for iOS

I figur **3.2.54** blir autocomplete-funksjonen kalt. Den tar inn søkefeltet og en sortert liste med aktuelle matvarer. Da matvarelista allerede er sortert i alfabetisk rekkefølge, sorteres lista igjen på indikatoren her.



Figur 3.3.2.55: Registreringssiden ved søk etter matvarer

I figur **3.3.2.55** ser vi resultatet fra funksjonene i figurene **3.3.2.51**-**3.3.2.54**.

Når man trykker på ønsket matvare, blir den lagt til i en liste under. Dette gjøres ut fra følgende funksjoner:

```
public void leggTilListe(String matvareid, String matvare, int indikator) {
    registreringListe.add(new Matvare(matvareid, matvare, indikator));

    oppdaterListe();
}

public void oppdaterListe() {
    registreringAdapter = new RegistreringListeAdapter(
            context: this,
            R.layout.listeelement,
            registreringListe);

    lvElementer.setAdapter(registreringAdapter);
}
```

Figur 3.3.2.56: Funksjonene som legger til matvare i liste for android

I figur **3.3.2.56** legger den første funksjonen matvaren til i en *ArrayList*, en type liste som blir mye brukt i Java. Den kaller så på den nederste funksjonen, som oppdaterer hvilken liste som skal vises i et *ListView*, en listevisning som viser objekter fra en ønsket liste.

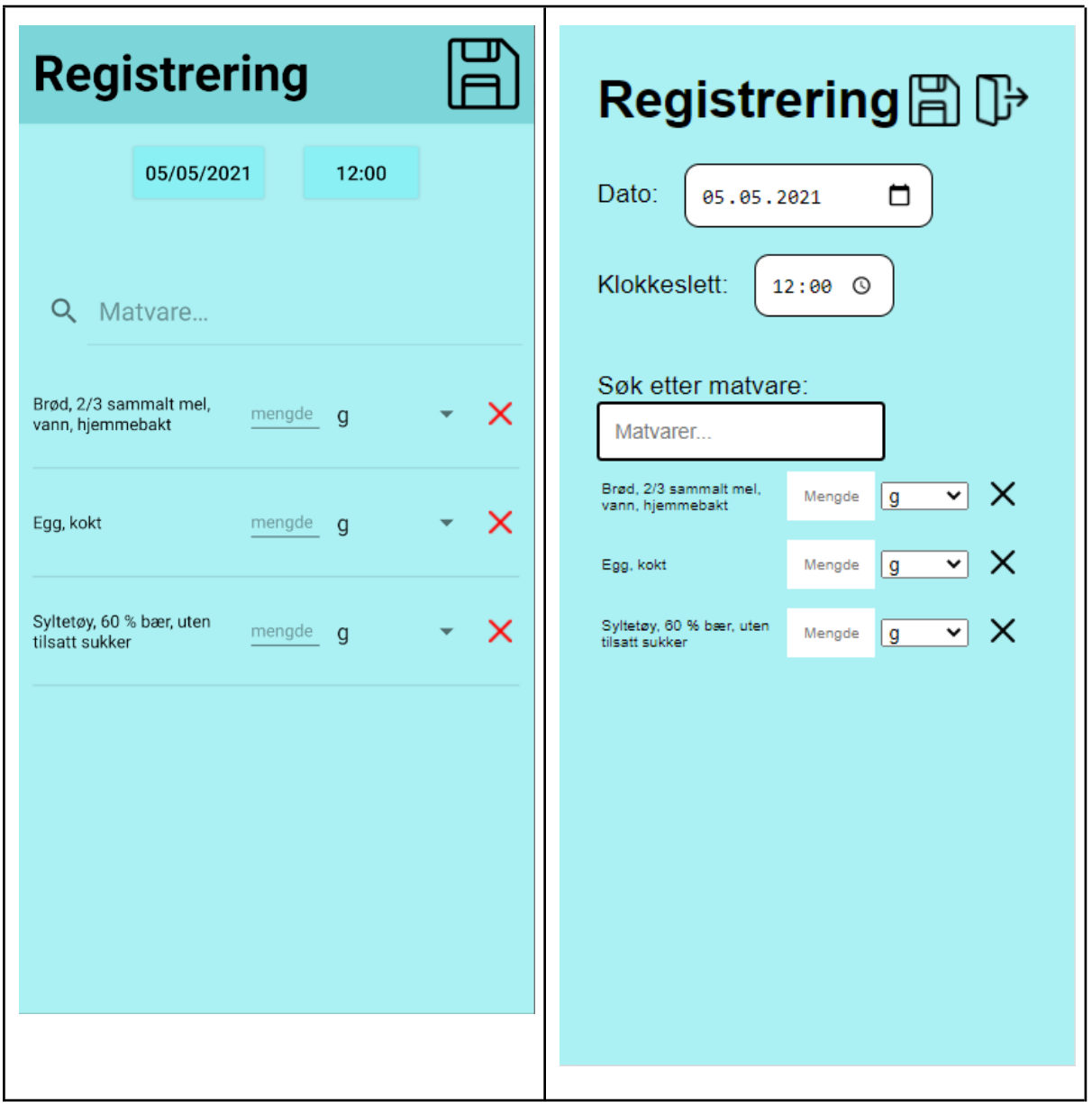
```
function leggTilListe(verdi) {
    liste[antallProdukter] = verdi;
    const s = liste.length-1;
    const trID = "trID"+id;

    const tr = document.createElement( tagName: "tr");
    tr.setAttribute( qualifiedName: "id",trID);
    const td1 = document.createElement( tagName: "td");
    td1.setAttribute( qualifiedName: "style", value: "width: 50%; font-size: 3vw");
    const td2 = document.createElement( tagName: "td");
    td2.setAttribute( qualifiedName: "style", value: "width: 25%");
    const td3 = document.createElement( tagName: "td");
    td3.setAttribute( qualifiedName: "style", value: "width: 10%");
    const td4 = document.createElement( tagName: "td");
    td4.setAttribute( qualifiedName: "style", value: "width: 5%");

    td1.innerHTML = verdi;
    td2.innerHTML = "<input style='background-color: white; width: 100%; " +
        "font-size: 3vw' value='' type='number' placeholder='Mengde'/>";
    td3.innerHTML = "<select><option>g</option><option>ml</option><option>dl</option>" +
        "<option>ts</option><option>ss</option><option>stk</option><option>skiver</option></select>";
    td4.innerHTML = "<input style='background-color: #ABF0F3'  type='image' " +
        "src='../bilder/close16.png' alt='Slett' onclick='slett(this.parentNode)'/>";

    tr.appendChild(td1);
    tr.appendChild(td2);
    tr.appendChild(td3);
    tr.appendChild(td4);

    document.getElementById( elementId: "tblListe").appendChild(tr);
    //list.parentNode.insertBefore(tr, list.nextSibling);

    id++;
    antallProdukter++;
}
```

Figur 3.3.2.57: Funksjonene som legger til matvare i liste for iOS

I figur **3.3.2.57** opprettes en tabellrad (*document.createElement("tr");*) med matvaren i tillegg til de feltene vi ønsker å ha med. Hvert felt i raden defineres med et td-element (table data cell).
I linjen *document.getElementById("tblListe").appendChild(tr)* legges denne tabellraden til en tabell som vi bruker for å vise matvarene som er lagt til.

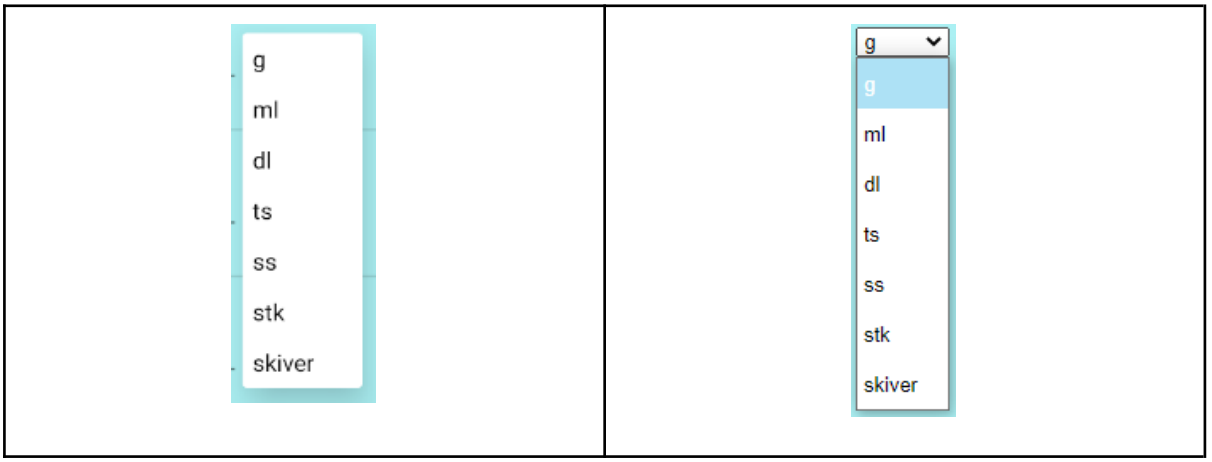
I tillegg til matvarene som blir valgt, blir det listet opp noen felter for hver matvare. Disse er mengden av hver matvare, og hvilken enhet mengden måles i. En knapp for å fjerne matvaren dersom den ble feilregistrert blir også lagt til.

Etter å ha lagt til ønskede matvarer, vil det se slik ut:



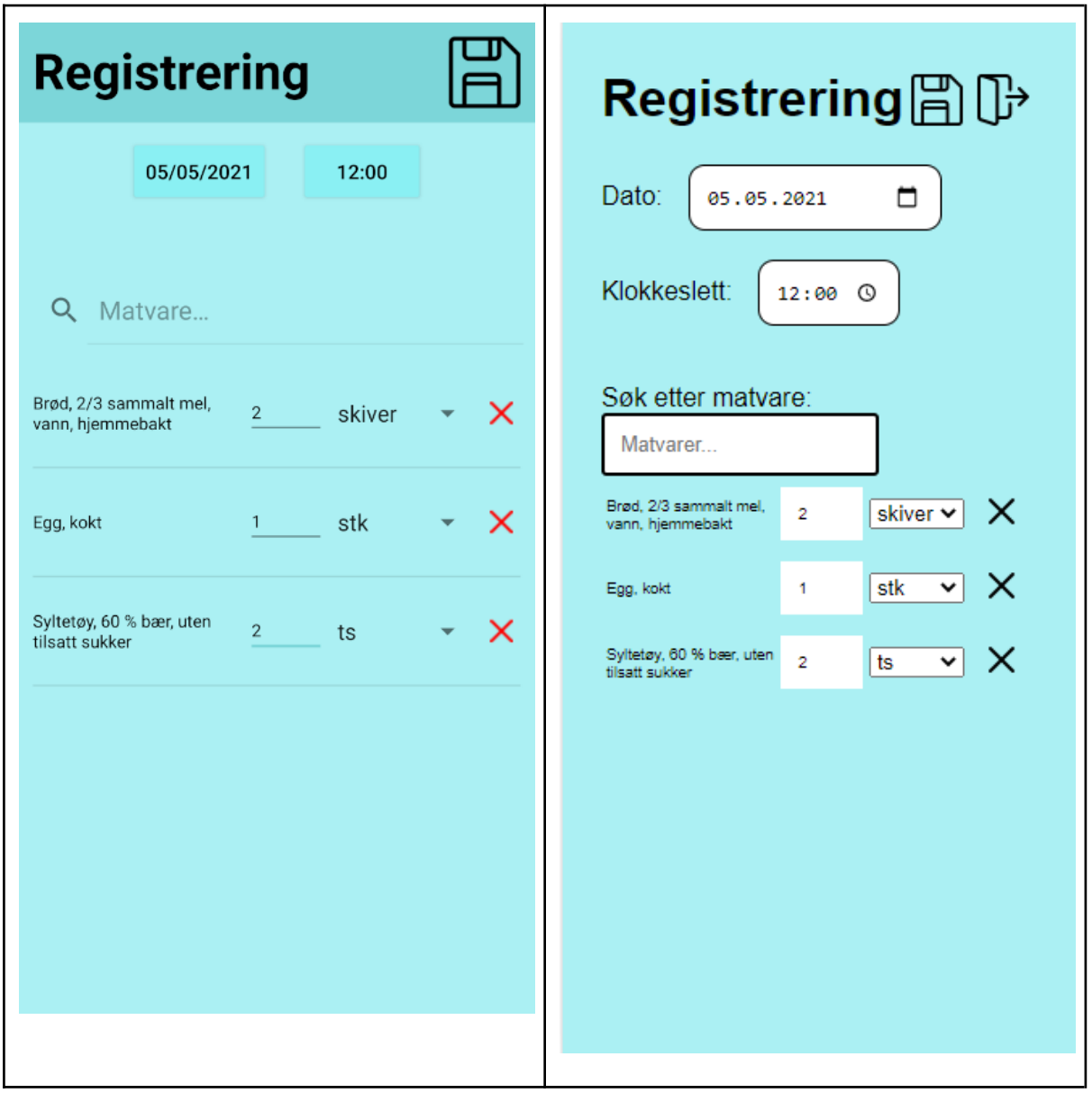
Figur 3.3.2.58: Registreringssiden etter å ha lagt til matvarer

Nå kan brukeren fylle ut feltene ved å skrive inn ønsket mengde i feltet for mengde, og velge enhet fra en liste:



Figur 3.3.2.59: Registreringssiden etter å ha lagt inn matvarer

Ferdig utfylt kan det slik ut:

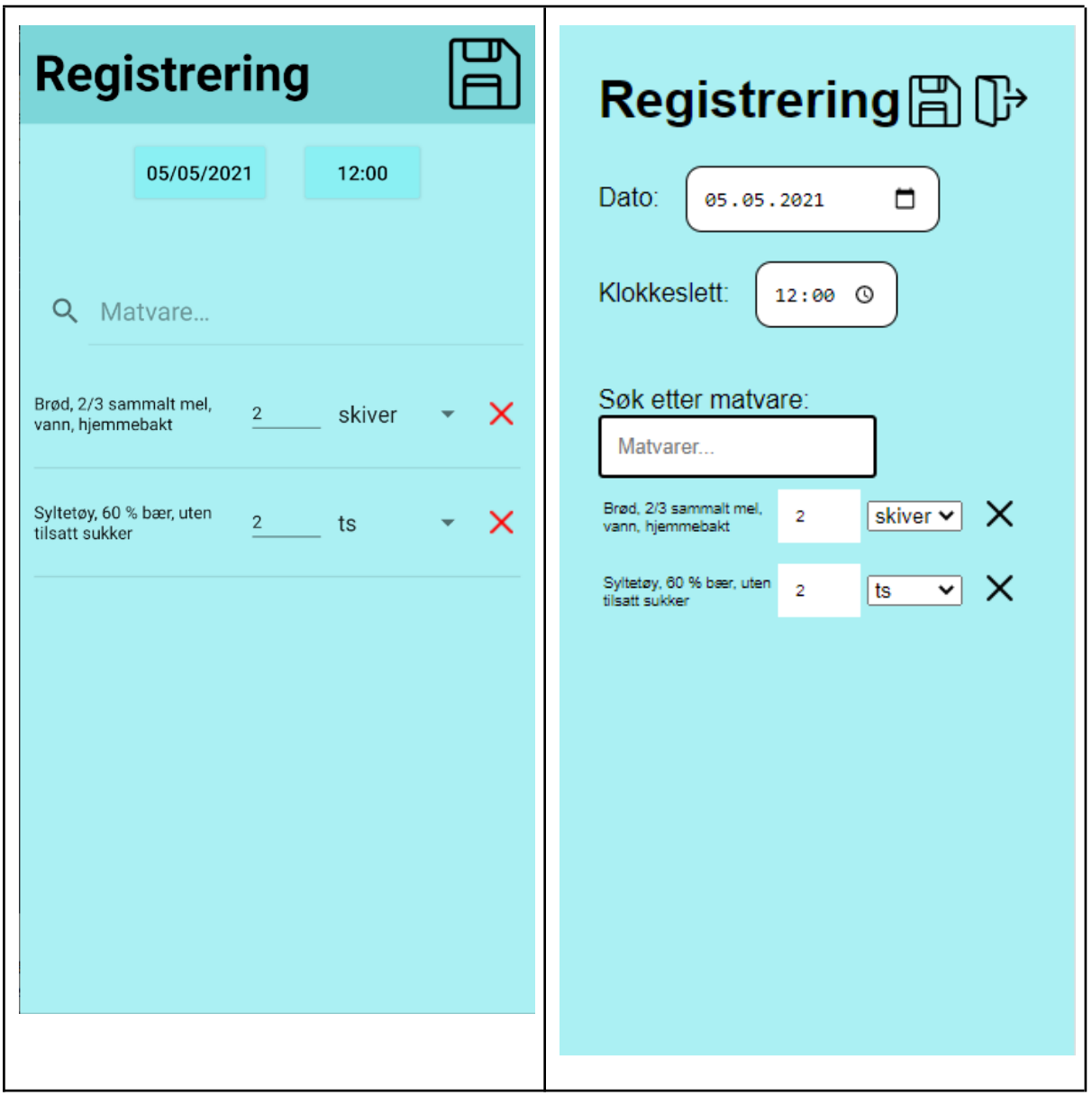


Figur 3.3.2.60: Registreringssiden etter å ha lagt inn matvarer

Sletting av matvarer

Dersom brukeren har lagt inn feil matvare, kan denne slettes ved å trykke på sletteknappen (X) helt til høyre for ønsket matvare.

Figur 3.3.2.61: Sletteknapper



Figur 3.3.2.62: Matvare slettet

I figur **3.3.2.62** er matvaren *Egg, kokt* slettet, og denne matvaren blir nå borte fra listen.

Når brukeren er fornøyd, trykker han/hun på lagreknappen øverst på siden og blir sendt tilbake til oversikten, hvor siste registrering nå blir listet opp.

Feilmeldinger

Når brukeren trykker på lagreknappen vil vedkommende, dersom det er noen feil, få en informativ feilmelding.

Første tilfelle er som tidligere hvis brukeren ikke har fylt ut feltene. Dette sjekkes med følgende funksjoner:

```
public void verifiser() {
    View view;
    EditText et;
    gyldig = true;
    if (knappVelgDato.getText().equals("Velg dato")) {
        tvFeilDato.setVisibility(View.VISIBLE);
        gyldig = false;
    }
    else {
        tvFeilDato.setVisibility(View.INVISIBLE);
    }
    if (knappVelgTid.getText().equals("Velg tid")) {
        tvFeilTid.setVisibility(View.VISIBLE);
        gyldig = false;
    }
    else {
        tvFeilTid.setVisibility(View.INVISIBLE);
    }
    if (lvElementer.getCount() > 0) {
        tvElementer.setVisibility(View.GONE);
        for (int i = 0; i < lvElementer.getCount(); i++) {
            view = lvElementer.getChildAt(i);
            et = (EditText) view.findViewById(R.id.etMengde);
            if (et.getText().toString().equals("")) {
                et.setError("Må fylle ut mengde!");
                gyldig = false;
            }
        }
    }
    else {
        tvElementer.setVisibility(View.VISIBLE);
        gyldig = false;
    }
    if (gyldig) {
        pd = new ProgressDialog( context: Registrering.this);
        pd.setMessage("Sender...");
        pd.setCancelable(false);
        pd.show();
        lagre();
    }
}
```

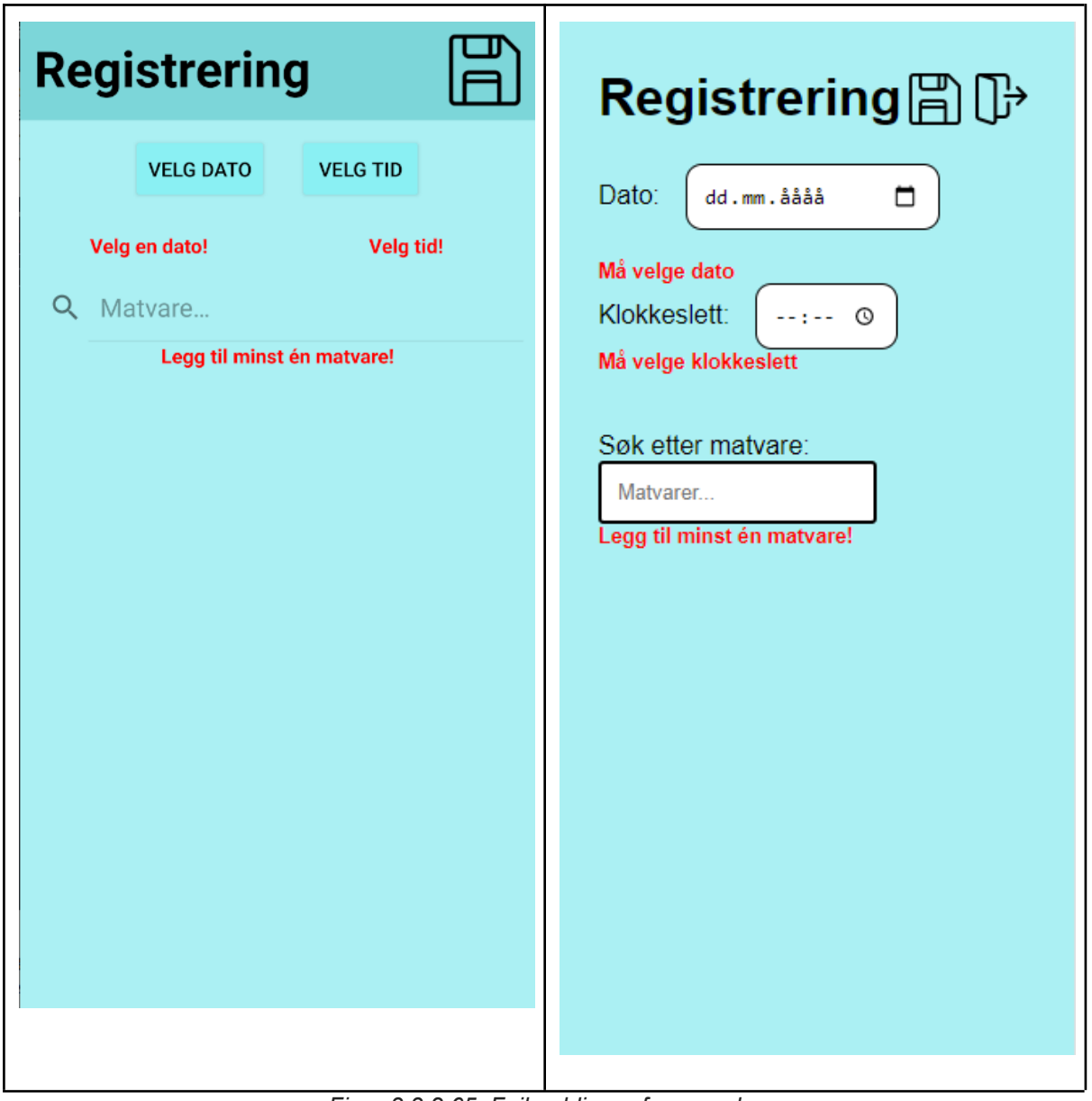
Figur 3.3.2.63: Funksjonen sjekker om feltene er gyldige for android

I figur **3.3.2.63** sjekker funksjonen først om teksten på knappen for dato er “Velg dato”. Er den det, vises en feilmelding om at brukeren må velge dato. Tilsvarende for tid-knappen, sjekkes det om teksten er “Velg tid”, og en feilmelding om at brukeren må velge tid vises dersom det stemmer.
Deretter sjekkes det om det er lagt til noen matvarer. Hvis antall matvarer er 0, vises en feilmelding om at brukeren må velge minst én matvare.

```
function valider() {
    felterGyldige = true;

    const feilDato = $("#feilDato");
    const feilTid = $("#feilTid");
    const datoVelger = $("#dato");
    const tidVelger = $("#tid");

    feilDato.css("display", "none");
    feilTid.css("display", "none");
    if (datoVelger.val() === "" || tidVelger.val() === "") {
        if (datoVelger.val() === "") {
            feilDato.css("display", "block");
        }
        if (tidVelger.val() === "") {
            feilTid.css("display", "block");
        }
        felterGyldige = false;
    }
    else {
        dato = datoVelger.val().split( separator: "-")[2]+datoVelger.val().split( separator: "-")[1]
            +datoVelger.val().split( separator: "-")[0];
        tid = tidVelger.val();
    }
    if ($("#tblListe tr").length > 0) {
        $("#feilListe").css("display", "none");
        felterGyldige = true;
        hentKostholdListe();

        for (let k of kostholdListe) {
            if (k.mengde.toString() === "") {
                felterGyldige = false;
                $("#feilMengde").css("display", "block");
                break;
            }
            else {
                felterGyldige = true;
                $("#feilMengde").css("display", "none");
            }
        }
    }
    else {
        felterGyldige = false;
        $("#feilListe").css("display", "block");
    }
}
```

Figur 3.3.2.64: Funksjonen sjekker om feltene er gyldige for iOS

I figur **3.3.2.64** sjekker funksjonen først om verdien av dato-inputfeltet er tom (“”). Er den det, vises en feilmelding om at brukeren må velge dato. Det samme sjekkes for tid-inputfeltet, og tilsvarende feilmelding vises. Igjen sjekkes det om antall matvarer er lik 0, og om det er det, får brukeren beskjed om å legge til minst én matvare. Feilmeldingene ser slik ut:



Figur 3.3.2.65: Feilmeldinger for mangler

Neste tilfelle er hvis brukeren har lagt til noen matvarer, men ikke har fylt ut mengden. Dette sjekkes i denne delen av koden vist i figurene **3.3.2.63** og **3.3.2.64**:

```
if (lvElementer.getCount() > 0) {
    tvElementer.setVisibility(View.GONE);
    for (int i = 0; i < lvElementer.getCount(); i++) {
        view = lvElementer.getChildAt(i);
        et = (EditText) view.findViewById(R.id.etMengde);
        if (et.getText().toString().equals("")) {
            et.setError("Må fylle ut mengde!");
            gyldig = false;
        }
    }
}
```

Figur 3.3.2.66: Funksjonen sjekker om mengde er utfylt for android

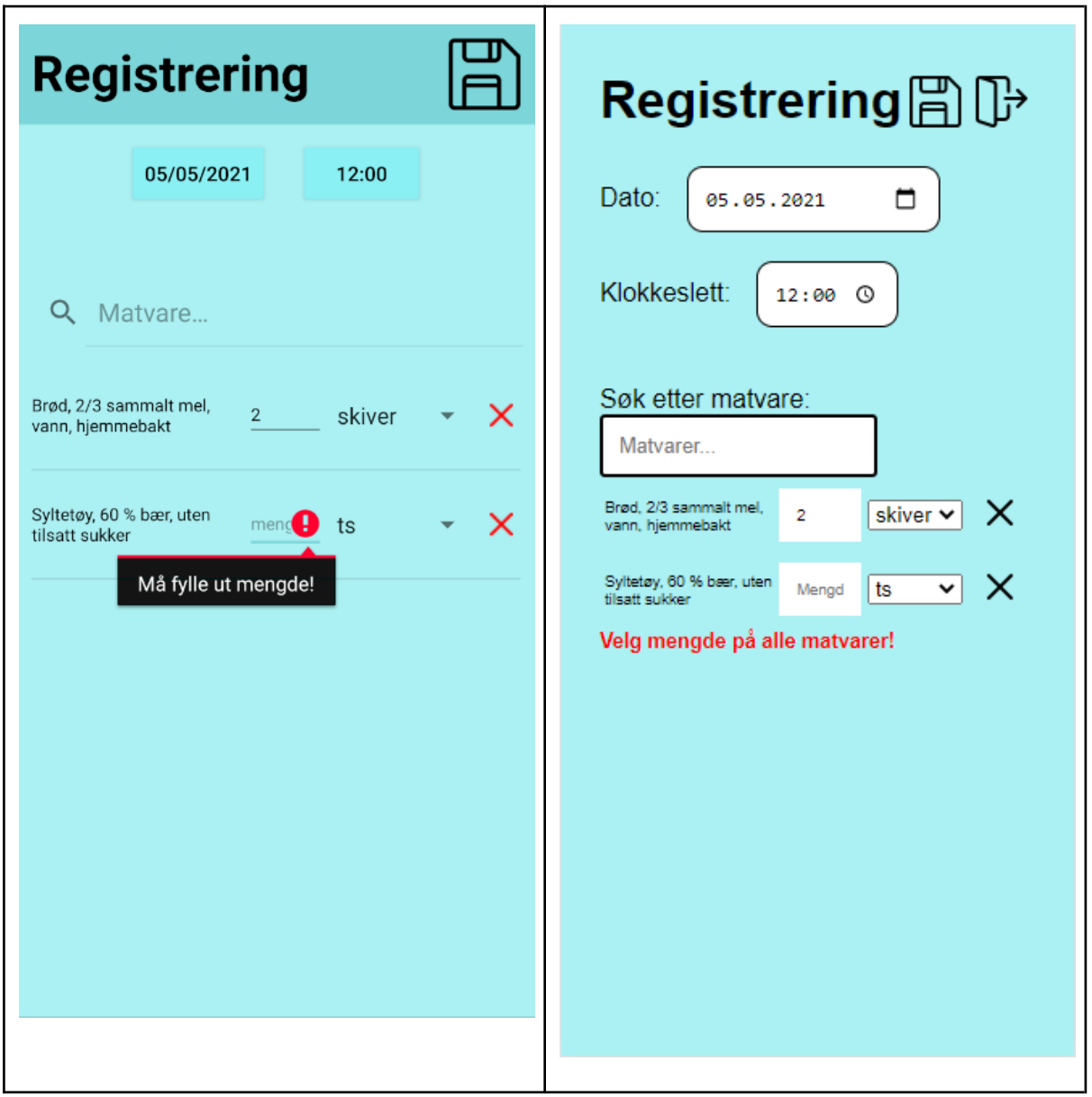
I figur **3.3.2.66** sjekker funksjonen om antall matvarer er større enn 0 for android. Hvis den er det, sjekker den for hver matvare om mengde er fylt ut. For hver matvare som mangler mengde, vises en feilmelding som ber brukeren fylle ut mengden.

```
if ($("#tblListe tr").length > 0) {
    $("#feilListe").css("display", "none");
    felterGyldige = true;
    hentKostholdListe();

    for (let k of kostholdListe) {
        if (k.mengde.toString() === "") {
            felterGyldige = false;
            $("#feilMengde").css("display", "block");
            break;
        }
        else {
            felterGyldige = true;
            $("#feilMengde").css("display", "none");
        }
    }
}
```

Figur 3.3.2.67: Funksjonen sjekker om mengde er utfylt for iOS

I figur **3.3.2.67** sjekker funksjonen om antall matvarer er større enn 0 for iOS. Hvis den er det, sjekker den for hver matvare om mengde er fylt ut. Hvis minst én matvare mangler mengde, vises en feilmelding som ber brukeren fylle ut mengde for alle matvarer.

Under vises feilmeldingene for mangel på mengde:



Figur 3.3.2.68: Feilmeldinger for manglende mengde

3.3.3 Backend

Som vist i systemmodellen i avsnitt **2.2.2.1**, består backenden i systemet mitt av en database og php-filer som et API på server. I dette kapittelet vil jeg gi en bedre innføring i hvordan databasen og php-filene er strukturert.

3.3.3.1 Database

Databasen består av tre tabeller, to hovedtabeller for data om brukerne og deres kosthold og en tabell for data om matvarene som kan registreres. Dette delkapittelet tar for seg disse tabellene og hvilke kolonner de består av.

Brukere

Tabellen *Brukere* består av informasjon om brukere. Hver bruker inneholder en unik brukerID (Patient_ID), et kryptert passord og et prosjektNr. Ettersom Patient_ID er unikt for hver bruker, ble dette naturlig nok primærnøkkel for denne tabellen.

	Column Name	Data Type	Allow Nulls
🔑	Patient_ID	varchar(10)	☐
	Passord	varchar(50)	☐
	ProsjektNr	int	☐

Figur 3.3.3.1: Tabellen Brukere

Kosthold

Tabellen *Kosthold* består av informasjon om brukernes kosthold. Hvert kosthold består av et løpenummer for hver bruker, brukerID'en, tidspunkt (EPOCH), dato (MLDTC), matvareID (MLTRTCD), matvarenavn (MLTRT), mengde (MLDOSE) og enhet (MLDOSU).

Som nevnt i avsnitt **2.2.2.4** valgte jeg å gå bort fra en kostholdsID for hvert kosthold. Istedenfor en ID for hvert kosthold, er det som er unikt for hvert kosthold en kombinasjon av Patient_ID, EPOCH og MLDTC. Disse tre utgjør til sammen en primærnøkkel.

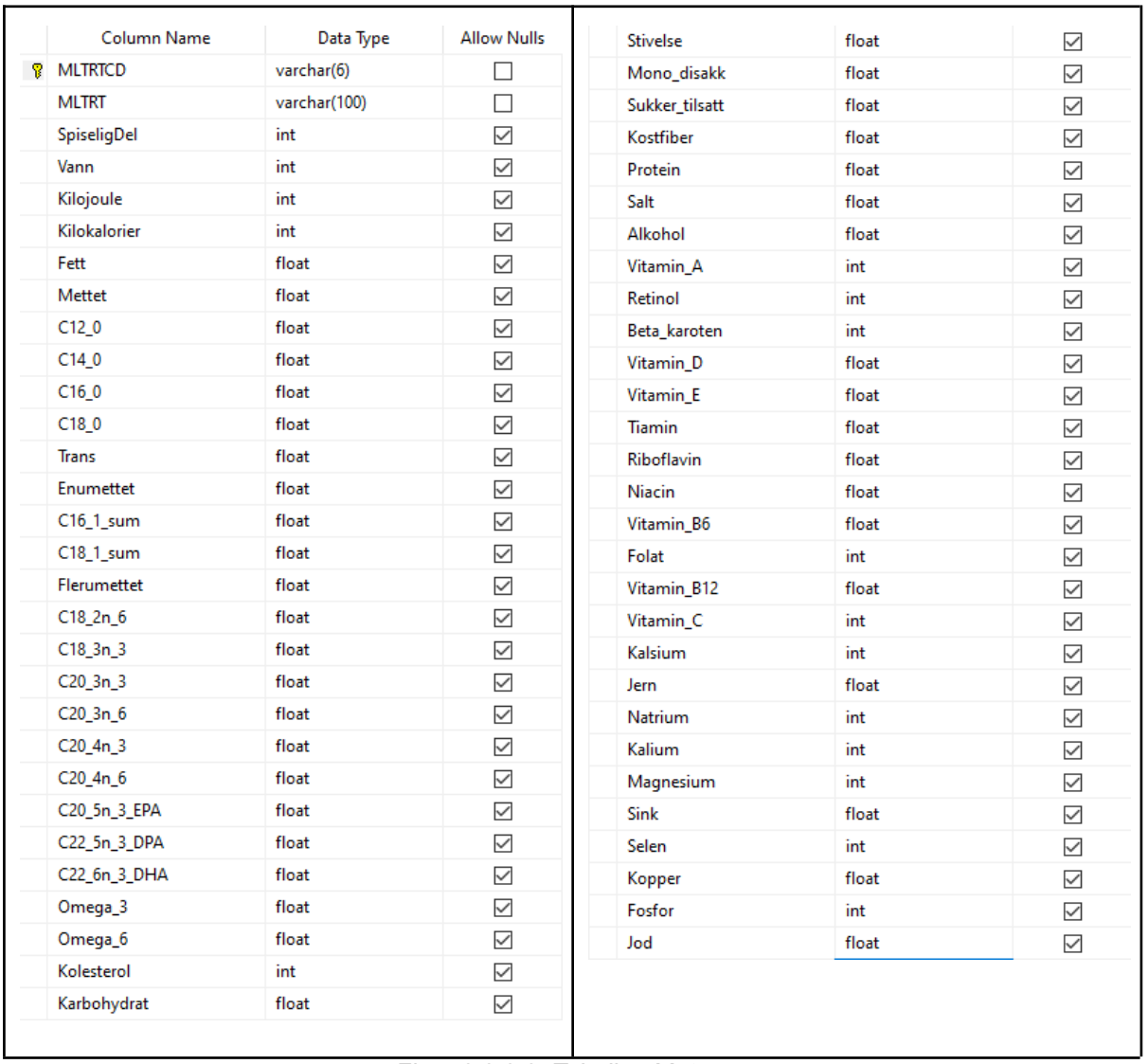
	Column Name	Data Type	Allow Nulls
	Val_no_seq	int	☐
🔑	Patient_ID	varchar(10)	☐
🔑	EPOCH	varchar(6)	☐
🔑	MLDTC	varchar(8)	☐
	MLTRTCD	varchar(6)	☐
	MLTRT	varchar(100)	☐
	MLDOSE	float	☐
	MLDOSU	varchar(6)	☐

Figur 3.3.3.2: Tabellen Kosthold

Matvarer

Tabellen *Matvarer* består av data om matvarer hentet fra *Matvaretabellen* [4]. Denne tabellen brukes for å knytte matvarene fra kostholdtabellen til relevant data for

matvaren. Jeg vil ikke gå gjennom kolonnene i denne tabellen da de er veldig mange, og da de fleste kolonnenavnene er meget forståelige. Disse kolonnene har heller ingen betydning for selve systemet, da denne tabellen kun vil bli brukt av forskerne i etterkant av registreringen. Legg merke til at jeg her har tillatt null-verdier for alt utenom *MLTRTCD* og *MLTRT* grunnet at flere av matvarene ikke inneholder data for alle kolonner.



Column Name	Data Type	Allow Nulls
MLTRTCD	varchar(6)	☐
MLTRT	varchar(100)	☐
SpiseligDel	int	☑
Vann	int	☑
Kilojoule	int	☑
Kilokalorier	int	☑
Fett	float	☑
Mettet	float	☑
C12_0	float	☑
C14_0	float	☑
C16_0	float	☑
C18_0	float	☑
Trans	float	☑
Enumettet	float	☑
C16_1_sum	float	☑
C18_1_sum	float	☑
Flerumettet	float	☑
C18_2n_6	float	☑
C18_3n_3	float	☑
C20_3n_3	float	☑
C20_3n_6	float	☑
C20_4n_3	float	☑
C20_4n_6	float	☑
C20_5n_3_EPA	float	☑
C22_5n_3_DPA	float	☑
C22_6n_3_DHA	float	☑
Omega_3	float	☑
Omega_6	float	☑
Kolesterol	int	☑
Karbohydrat	float	☑
Stivelse	float	☑
Mono_disakk	float	☑
Sukker_tilsatt	float	☑
Kostfiber	float	☑
Protein	float	☑
Salt	float	☑
Alkohol	float	☑
Vitamin_A	int	☑
Retinol	int	☑
Beta_karoten	int	☑
Vitamin_D	float	☑
Vitamin_E	float	☑
Tiamin	float	☑
Riboflavin	float	☑
Niacin	float	☑
Vitamin_B6	float	☑
Folat	int	☑
Vitamin_B12	float	☑
Vitamin_C	int	☑
Kalsium	int	☑
Jern	float	☑
Natrium	int	☑
Kalium	int	☑
Magnesium	int	☑
Sink	float	☑
Selen	int	☑
Kopper	float	☑
Fosfor	int	☑
Jod	float	☑

Figur 3.3.3.3: Tabellen Matvarer

3.3.3.2 API

API'et består av php-filer som utfører hver sin oppgave mot databasen. Disse filene blir kalt av frontend-delen av applikasjonen når tilgang til databasen er nødvendig. Dette delkapittelet gir en innføring i disse filene, når de benyttes og hvilke oppgaver de ulike filene har.

Innlogging

Når en bruker logger inn, kalles filen *brukerOut.php*. Denne filen henter brukere fra databasen, slik at applikasjonen kan sammenligne innloggingsinformasjonen med de registrerte brukerne. I figur **3.3.3.4** ser vi koden fra brukerOut.php. Den henter all informasjon fra brukere med en SELECT-operasjon og returnerer disse dersom det er tilkobling til databasen.

```php
$sql = "select * from Brukere";
$stmt = sqlsrv_query( $conn, $sql);

if( $stmt === false)
{
   echo "Feil ved henting av brukere.\n";
   die( print_r( sqlsrv_errors(), true));
}

$json = array();

while( $row = sqlsrv_fetch_array( $stmt, SQLSRV_FETCH_ASSOC))
{
    $json[] = $row;
}

echo json_encode($json);
sqlsrv_close($conn);
?>
```

Figur 3.3.3.4: Skjermdump fra brukerOut.php

Endre passord

Når en bruker skal endre passord, kalles filen *NyttPassord.php*. Denne filen tar inn brukernavnet og en kryptert versjon av det nye passordet.

I figur **3.3.3.5** ser vi koden fra NyttPassord.php. Her kjøres en UPDATE-operasjon mot databasen som oppdaterer det gamle passordet med det nye for den spesifikke brukeren.

```php
$innB=$_REQUEST['BrukerID'];
$innP=$_REQUEST['Passord'];

$brukerid=(String)$innB;
$passord=(String)$innP;

$sql = "update Brukere set Passord='$passord' where Patient_ID='$brukerid'";
$stmt = sqlsrv_query( $conn, $sql);

if( $stmt === false)
{
  echo "Feil ved endring av passord.\n";
  die( print_r( sqlsrv_errors(), true));
}

sqlsrv_close($conn);
?>
```

Figur 3.3.3.5: Skjermdump fra NyttPassord.php

Hente kosthold per tidspunkt

Når en bruker er logget inn, kalles umiddelbart filen *brukerKosthold.php*. Denne henter hvert måltid (alt kosthold som har blitt registrert til samme tidspunkt) for den påloggede brukeren. Det er denne filen som sender registreringer til oversiktslisten vist i figur **3.3.2.40**.

I figur **3.3.3.6** ser vi koden fra brukerKosthold.php. Her kjøres en SELECT-operasjon mot databasen som henter hvert unike tidspunkt. I koden betyr *distinct* unik, *EPOCH* er Meddoc sin variabel for klokkeslett, *MLDTC* er Meddoc sin variabel for dato og *Patient_ID* er Meddoc sin variabel for brukernavn. Dette hentes for den brukeren som er oppgitt.

```php
$innB=$_REQUEST['BrukerID'];

$brukerid=(String)$innB;

$sql = "select distinct EPOCH, MLDTC from Kosthold where Patient_ID='$brukerid'";
$stmt = sqlsrv_query( $conn, $sql);

if( $stmt === false)
{
   echo "Feil ved henting av brukerkosthold.\n";
   die( print_r( sqlsrv_errors(), true));
}

$json = array();

while( $row = sqlsrv_fetch_array( $stmt, SQLSRV_FETCH_ASSOC))
{
    $json[] = $row;
}

echo json_encode($json);
sqlsrv_close($conn);
?>
```

Figur 3.3.3.6: Skjermdump fra brukerKosthold.php

Hente kosthold

Når brukeren skal registrere et kosthold, kalles filen *brukerKostholdOut.php*. Denne gjør omtrent det samme som filen vist i figur **3.3.3.6**, men istedenfor å hente unike tidspunkter, henter denne alle registreringer. Dette er for å opprette en ID for det nye kostholdet. I applikasjonen blir antall tidligere registrerte kosthold summert, og for den nye ID'en blir dette antallet + 1.

I figur **3.3.3.7** ser vi koden fra brukerKostholdOut.php. Denne er som nevnt omtrent lik som *brukerKosthold.php*, så jeg velger å ikke forklare dette nærmere.

```php
$innB=$_REQUEST['BrukerID'];

$brukerid=(String)$innB;

$sql = "select * from Kosthold where Patient_ID='$brukerid'";
$stmt = sqlsrv_query( $conn, $sql);

if( $stmt === false)
{
   echo "Feil ved henting av brukerkosthold.\n";
   die( print_r( sqlsrv_errors(), true));
}

$json = array();

while( $row = sqlsrv_fetch_array( $stmt, SQLSRV_FETCH_ASSOC))
{
    $json[] = $row;
}

echo json_encode($json);
sqlsrv_close($conn);
?>
```

Figur 3.3.3.7: Skjermdump fra brukerKostholdOut.php

Registrere kosthold

Når brukeren trykker på knappen for å sende inn kostholdet sitt, kalles filen *kostholdIn.php*. Denne sender all data om kostholdet til databasen.

I figur **3.3.3.8** ser vi koden fra kostholdIn.php. Her kjøres en INSERT-operasjon mot databasen, som legger til en ny rad i tabellen Kosthold med all data fra registreringen til brukeren.

```php
$innKID=$_REQUEST['KostholdID'];
$innBID=$_REQUEST['BrukerID'];
$innKl=$_REQUEST['Klokkeslett'];
$innD=$_REQUEST['Dato'];
$innMatID=$_REQUEST['MatvareID'];
$innMat=$_REQUEST['Matvare'];
$innM=$_REQUEST['Mengde'];
$innEnhet=$_REQUEST['Enhet'];

$kostholdid=(String)$innKID;
$brukerid=(String)$innBID;
$klokkeslett=(String)$innKl;
$dato=(String)$innD;
$matvareid=(String)$innMatID;
$matvare=(String)$innMat;
$mengde=(float)$innM;
$enhet=(String)$innEnhet;

$sql = "insert into Kosthold values ('$kostholdid', '$brukerid', '$klokkeslett',
                                    '$dato', '$matvareid', '$matvare', '$mengde', '$enhet')";
$stmt = sqlsrv_query( $conn, $sql);

if( $stmt === false)
{
   echo "Feil ved innsending av kosthold.\n";
   die( print_r( sqlsrv_errors(), true));
}

sqlsrv_close($conn);
?>
```

Figur 3.3.3.8: Skjermdump fra kostholdIn.php

3.3.4 Sikkerhet

Når man utvikler et system, er det flere ting å ta hensyn til når det gjelder sikkerhet. Her vil jeg fortelle hvilke hensyn jeg har gjort.

3.3.4.1 Passord

Brukerne er lagret i en egen bruker-tabell i databasen. Her lagres brukernavn, passord og prosjektnr. Dersom passordene hadde ligget i ren tekst på databasen og uvedkommende hadde fått tilgang til disse, ville dette være katastrofalt. Passordene blir derfor kryptert før de lagres for å sikre brukerne. I min løsning har jeg brukt SHA256 med salting. SHA256 er en hash-funksjon, en funksjon som krypterer en gitt tekststreng slik at den får en ny verdi. SHA256 er en underkategori av hash-familien

SHA-2 og er én av de mest brukte hash-funksjonene. Et salt er en tilfeldig generert tekststreng som blir lagt til passord-tekststrengen før hashing for å gjøre krypteringen enda sikrere. Når brukerne skal logge inn, krypteres teksten som skrives inn i passordfeltet før den sjekkes mot det krypterte passordet som er lagret i databasen. Dersom disse er like, får brukeren tilgang og blir logget inn.

For android-applikasjonen ble dette implementert enkelt med innebygde metoder for kryptering (se figur **3.3.2.3**). For iOS var det derimot ikke like enkelt. Ettersom jeg allerede hadde laget en fungerende løsning for android, ble jeg nødt til å bruke en tilsvarende krypteringsmetode til iOS for å kunne sjekke de krypterte passordene i databasen. Jeg fant ut at jeg kunne få tilsvarende resultat ved å implementere modulen crypto.js [25] fra Node.js [24]. Node.js er som nevnt under **2.1.5.3** et system for å kjøre JavaScript-kode på server. Det kommer også med mange moduler, blant annet krypteringsmodulen crypto.js som jeg benyttet meg av.

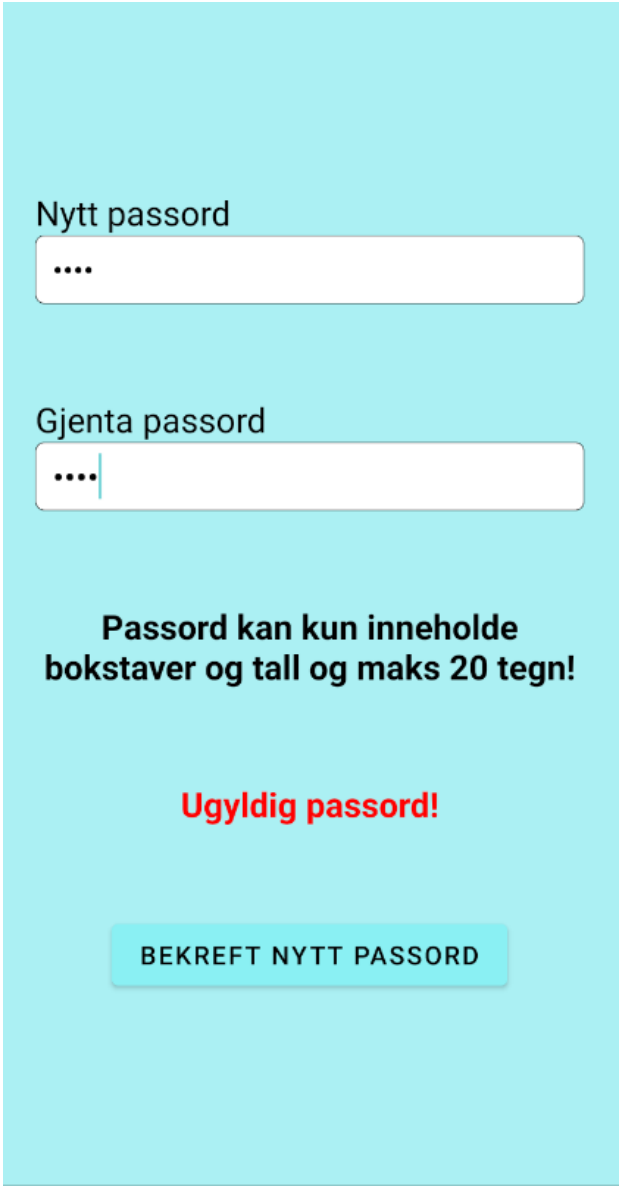
For å kunne benytte meg av denne modulen, var jeg nødt til å bruke metoden *require()*. Problemet var at denne metoden ikke kan brukes i nettlesere, hvilket var det jeg trengte modulen til. For å løse dette fant jeg verktøyet *Browserify [26]*. Browserify hjelper deg med å pakke sammen en javascript-fil (lage en bundle) med alle nødvendige avhengigheter som trengs for å få brukt *require*-metoden i nettlesere. Med dette på plass, fikk jeg tilsvarende resultat fra krypteringen som for android-applikasjonen.

3.3.4.2 SQL injection

SQL injection [34] er en angrepsmetode for å manipulere en sql-setning til å gjøre ting som den ikke var ment til å gjøre. Hvis et inputfelt i en applikasjon er knyttet mot en sql-spørring, kan en angriper utnytte dette for hente ut eller ødelegge data fra databasen. For å forebygge SQL injection, kan man blant annet bruke regulære uttrykk (RegEx) for å utelukke bruk av ulovlige tegn i inputfelt.

I min applikasjon er det kun to steder hvor et inputfelt er knyttet mot sql-spørringer. Dette er feltene for å velge nytt passord, og feltene for å angi mengde ved registrering. For nytt passord-feltene har jeg brukt RegEx til å kun tillate bokstaver og tall. På denne måten kan man ikke utnytte sql-spørringen med SQL injection. For

feltet for input av mengde, godkjennes kun tall og punktum (for desimaltall). Min applikasjon er dermed godt sikret mot SQL injection.



Figur 3.3.4.1: Skjermdump som viser regulært uttrykk tatt i bruk for nytt passord

3.3.4.3 Direkte tilgang til HTML-filer

For applikasjonen for iOS kan man i utgangspunktet gå direkte til en side som først skal vises etter innlogging. For å forebygge dette har jeg brukt sessionStorage til å sjekke om en bruker er logget inn. Når en bruker logger inn, lagres en nøkkel med en nøkkelverdi. For alle sider utenom innloggingssiden, sjekkes det om det finnes en verdi for denne nøkkelen. Dersom denne finnes får man tilgang til den aktuelle siden.

Hvis den ikke finnes, blir man sendt til innlogging. Slik unngår man at uvedkommende får brukt systemet.

```
$(function () {
    if (sessionStorage.getItem( key: "brukerID") === null) {
        location.replace( url: "../index.html");
    }
```

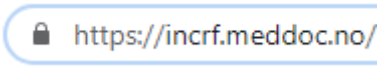
Figur 3.3.4.2: Eksempel hvor det sjekkes om en bruker er logget inn

3.3.4.4 Transport Layer Security (TLS)

TLS [29] er en sikkerhetsprotokoll for internett-kommunikasjon, og er en standard praksis for å bygge sikre webapplikasjoner. For TLS-protokollen oppnås tre hovedkomponenter:

- Kryptering: Skjuler overført data fra tredjeparter.
- Autentisering: Passer på at partene som kommuniserer er de de hevder å være.
- Integritet: Verifiserer at informasjonen ikke har blitt manipulert.

TLS-kryptering bidrar til å beskytte webapplikasjoner mot angrep. På webserveren til Meddoc brukes *HTTPS*, en implementering av TLS-kryptering i tillegg til vanlig HTTP-protokoll.



Figur 3.3.4.3: Viser HTTPS i URL

4. Testdokumentasjon

Testing utgjør en veldig viktig del av systemutvikling for å sørge for at alt virker som det skal, og for å finne og eventuelt fjerne feil. I dette kapittelet vil de ulike testmetodene som er blitt brukt forklares nærmere.

4.1 Enhetstesting

Enhetstesting [9] tester alle funksjoner og metoder i systemet isolert, uavhengig av de andre. Enhetstesting er ofte automatisert der et verktøy skriver og kjører testene, men det kan også gjøres manuelt.

Det er umulig å teste alt i et system, så man må ta prioriteringer [13]. Da det tar lang tid å sette opp enhetstesting, valgte jeg å bortprioritere dette og fokusere på de andre testmetodene som er beskrevet under.

4.2 Integrasjonstesting

Integrasjonstesting [10] er en såkalt white-box-testing [14]. White-box-testing er en testmetode som tester selve koden i et program. Fordelen med white-box-testing er at det kan utføres tidlig i utviklingsfasen. Integrasjonstesting tester at komponenter i et system som fungerer fint individuelt fortsatt fungerer når de blir integrert med hverandre. Dette gjøres normalt etter enhetstesting. Når en individuell komponent har blitt testet, legges en annen til og det testes om disse fungerer sammen.

Jeg valgte å gjøre integrasjonstestingen manuelt da jeg ikke har brukt automatiserte tester tidligere, og jeg fant det dermed mer hensiktsmessig å få all funksjonalitet på plass fremfor å sette meg inn i dette. Dette gjorde jeg fortløpende etter at metodene ble opprettet. For hver funksjon som ble lagt til, testet jeg at denne virket før flere komponenter ble implementert. Testingen bestod blant annet av å teste at alle knapper fungerte som de skulle og at kun godkjente input-verdier ble akseptert.

#	Beskrivelse	Forventet resultat	Resultat	Bestått/ Ikke bestått
1	Tomme felter	“Fyll ut begge feltene!”	“Fyll ut begge feltene!”	Bestått
2	Ulike felter	“Passordene stemmer ikke!”	“Passordene stemmer ikke!”	Bestått
3	Ugyldige tegn eller for langt passord	“Ugyldig passord!”	“Ugyldig passord!”	Bestått
4	Gyldige og like passord	Sendes tilbake til innlogging og kan logge på med nytt passord umiddelbart.	Sendes tilbake til innlogging og kan logge på med nytt passord umiddelbart.	Bestått

Figur 4.2.1: Utdrag fra testresultatene av integrasjonstesting

I figur **4.2.1** er testresultater for endring av passord. For testing av denne seksjonen, brukte jeg ulike inputverdier for feltene vist i figur **3.3.2.28**. Kolonnen *Beskrivelse* forteller hvilken tilstand inputen er i, og *Forventet resultat* forteller hva som er ment å skje.

4.3 Systemtesting

Systemtesting [11] er en black-box-testing [15] som tester hele systemet sammensatt. Black-box-testing er en testmetode som tester funksjonaliteten til et program fremfor strukturen og koden. Fordelen med denne testen er at man ikke nødvendigvis trenger kunnskap innen programmering. Dette vil si at andre enn systemutviklere også kan utføre denne testen. Systemtesting har fokus på at komponenter i systemet kommuniserer som forventet og at systemet oppfyller de funksjonelle kravene fra planleggingsfasen.

For min applikasjon ble denne testen utført av to personer ved Meddoc, uavhengig av hverandre. Ved å la flere teste applikasjonen er det større sjanse for å avdekke feil eller mangler. Jeg som systemutvikler vet hvordan koden fungerer, og har lett for å låse meg til en tankegang om hva bruker skal gjøre i applikasjonen. Ved å la andre som ikke kjenner koden teste applikasjonen, kan situasjoner som jeg ikke har tenkt på dukke opp. Måten testingen ble gjort på var å laste ned eller kjøre appen som en

helt ny bruker og teste alle muligheter. Dette vil si at det ble testet gyldig og ugyldig input for innlogging, endring av passord og registrering.

4.4 Akseptansetesting

For sluttproduktet ble det ved Meddoc gjennomført en akseptansetest [12]. Dette er en test hvor bruker går gjennom applikasjonen og sjekker om den tilfredsstiller forventet resultat. Dette er en viktig test som viser om applikasjonen faktisk er slik oppdragsgiver ønsker. Det er den siste testen i testfasen og skiller seg fra de andre testmetodene da det er oppdragsgiver som skal si om resultatene er riktige.

Under er brukers testplan med resultater.

#	Handling	Forventet resultat	Bestått/ Ikke bestått
1	Førstegangs innlogging	Når bruker starter applikasjonen for første gang, skal han/hun kunne velge ønsket mobil-operativsystem for å laste ned (Android) eller åpne (iOS).	Bestått
1-1	Bytte passord	1. Mulig å endre passord etter å ha skrevet inn forhåndsdefinert brukernavn og passord, og deretter å huke av boksen for å endre passord. 2. Applikasjonen tar brukeren tilbake til innloggingssiden etter å ha trykket på *Bekreft*.	Bestått
1-3	Huske brukernavn og passord	Når bruker huker av boksen for å huske brukernavn og passord, skal applikasjonen huske dette til neste gang brukeren skal logge inn.	Bestått

2	Oversiktsside	1. Skjermen viser oversiktssiden etter å ha logget inn med forhåndsdefinert brukernavn og passord. 2. Skjermen viser oversiktssiden etter å ha logget inn med forhåndsdefinert brukernavn og nylig endret passord.	Bestått
3	Registrere kosthold	Ved trykk på pluss-knappen (+), åpnes registreringssiden.	Bestått
3-1	Kostholdsregistrering	Mulig å registrere flere matvarer under samme registrering.	Bestått
3-2	Velge enheter til mat/drikkevarer	Mulig å velge ønsket enhet ved å velge enhetstypen fra nedtrekkslisten (g, ml, dl, ts, ss, stk og skiver).	Bestått
4	Lagre	Mulig å lagre data ved å trykke på lagre-ikonet.	Bestått
4-1	Historie over registreringstidspunkter	Etter å ha trykket på lagre-ikonet viser skjermen oversiktssiden og oversikten over registreringshistorikk oppdateres under “Tidligere registreringer”.	Bestått
4-2	Tilbake til oversikt	Etter å ha trykket på tilbake-knappen, viser skjermen oversiktssiden.	Bestått

Figur 4.4.1: Testresultater fra akseptansetesting

5. Avslutning

Gjennom prosjektperioden har jeg fått en enorm erfaring i å utvikle et system fra bunnen av. Tidligere har jeg bare utviklet enkelte deler om gangen. Prosjektet har vært veldig lærerikt og spennende. Det er gøy å kunne bruke det man har lært i løpet av studiet til å lage et sluttprodukt som skal brukes i forskningsarbeid.

Det å ha jobbet alene på dette prosjektet har ført til mye jobb, men siden jeg har måttet sette meg inn i alle delene av prosjektet, har jeg fått et godt innblikk i hvordan alt henger sammen. Jeg synes det har vært spennende å ha lagd alle aspekter ved applikasjonen, og jeg føler at jeg virkelig har utviklet meg som programmerer og utvikler.

Et punkt som kunne ha blitt forbedret er dersom brukeren glemmer passordet sitt. Slik det er nå må brukerne sende mail til Meddoc dersom de har glemt passordet, noe som er litt tungvint. En bedre løsning hadde vært å legge til en knapp for glemt passord, der brukeren kan få nytt passord tilsendt på mail.

Dersom det hadde blitt mer tid, ville jeg også ha satt meg inn i React Native og utviklet en kryssplattform-applikasjon for både android og iOS. Jeg tror produktet kunne fremstått som enda mer profesjonelt dersom alt fokus var på én applikasjon fremfor to. Jeg føler likevel at jeg klarte å utvikle to applikasjoner som begge tilfredsstiller oppdragsgivers behov.

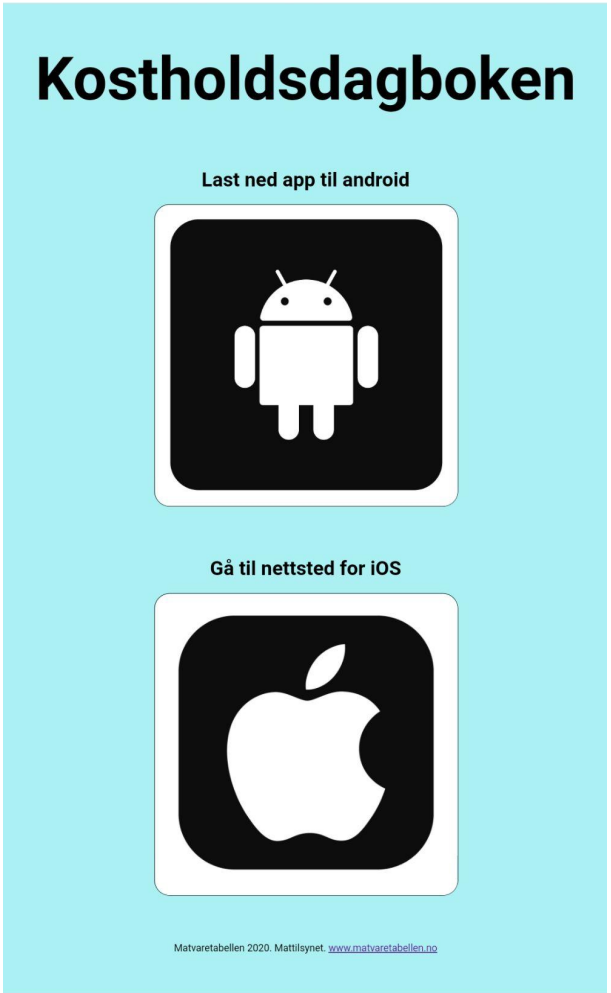
Ansatte ved Meddoc har under hele utviklingsfasen av prosjektet fått testet applikasjonen og kommet med gode tilbakemeldinger underveis. De virker fornøyde med sluttproduktet, og produktet er allerede implementert i systemene deres og brukes for øyeblikket i deres nåværende forskningsprosjekt.

6. Brukermanual

For å være sikker på at brukerne skal klare å bruke applikasjonen, ble det laget en brukermanual. Dette kapittelet tar for seg denne.

6.1 Valg av versjon

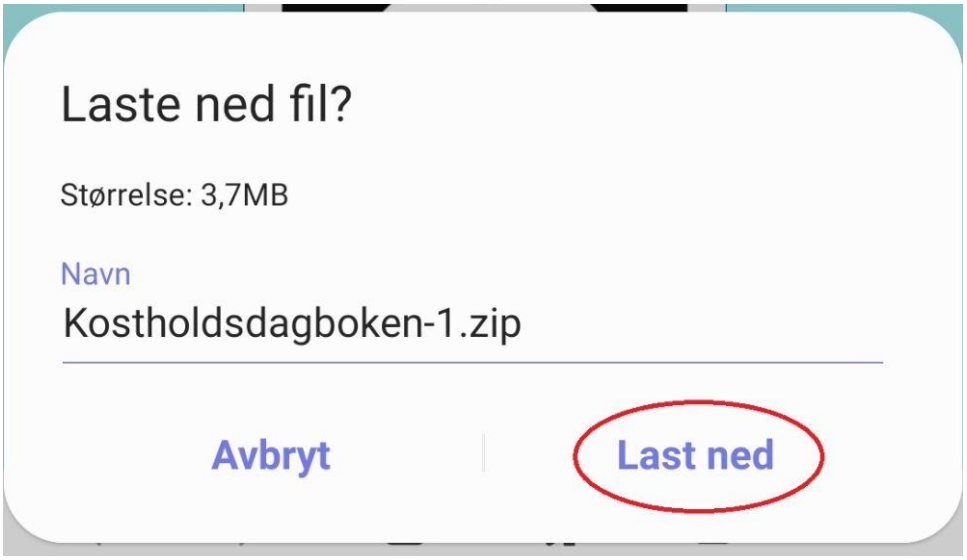
Du vil motta en lenke i e-posten din, og ved klikk på den så åpnes startsiden for programvaren i en nettleser. Herfra må du velge hvilket operativsystem mobiltelefonen din bruker, Android eller iPhone ved å trykke på aktuell knapp. Appen kan lastes ned på mobilen hvis du bruker Android-telefon, eller den kan åpnes i nettleseren hvis du bruker iPhone (iOS).



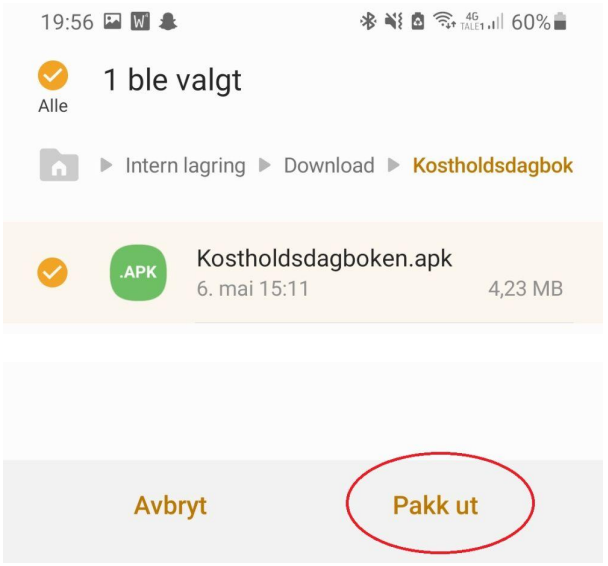
6.2 Android

6.2.1 Installering

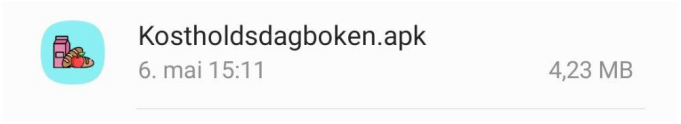
Ved å trykke på Android-knappen, vil du få en forespørsel om å laste ned en fil.



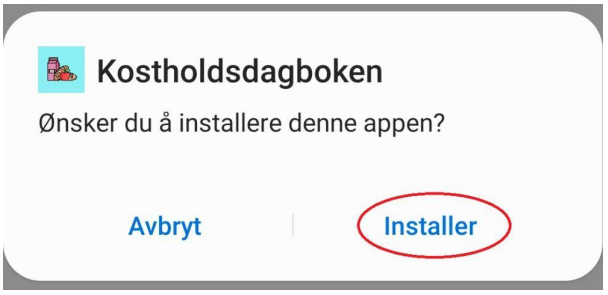
Trykk *Last ned*. Filen er en zip-fil som må pakkes ut. Lokaliser den nedlastede filen, og trykk pakk ut.



Du ender nå opp med en apk-fil (android application package). Trykk på denne.



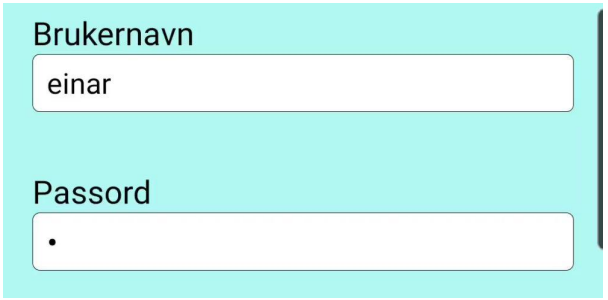
Velg *Installer*.



Applikasjonen blir nå installert på din mobil, og kan finnes blant dine andre apper.

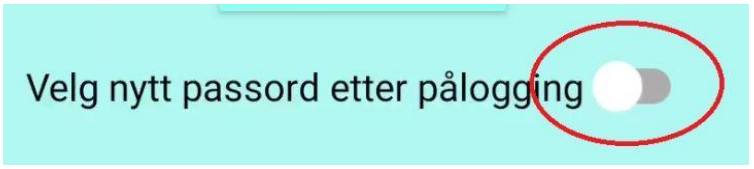
6.2.2 Innlogging

Når du kjører applikasjonen, kommer du til en innloggingsside. Skriv inn innloggingsopplysningene som ble oppgitt i din e-post.

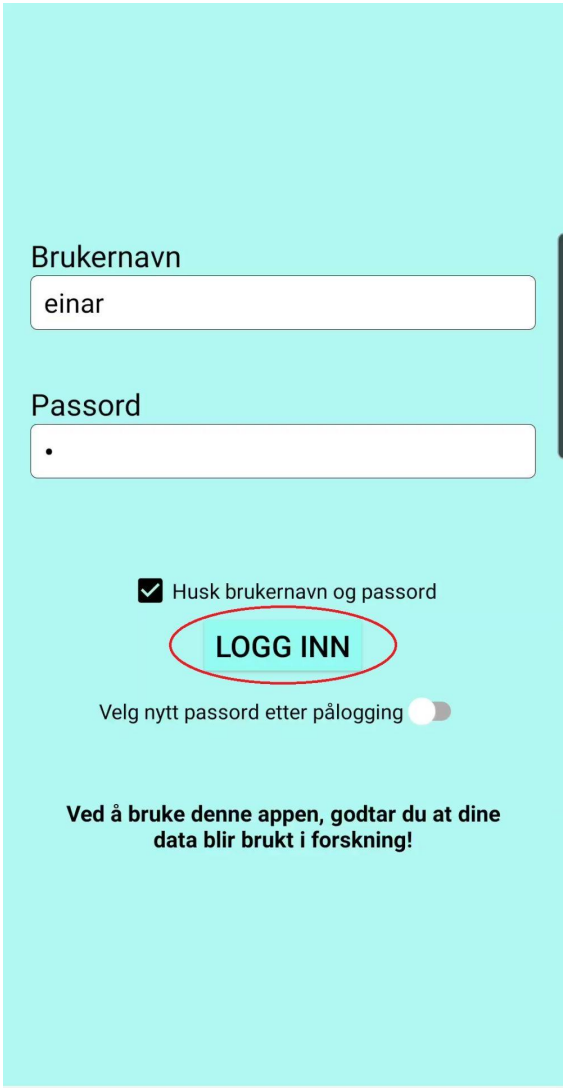


Dersom du ønsker at applikasjonen skal huske innloggingsinformasjonen for deg, kan du huke av for dette i boksen *Husk brukernavn og passord*.

Dersom du ønsker å endre passord etter innlogging, kan du huke av for dette ved *Velg nytt passord etter pålogging*.

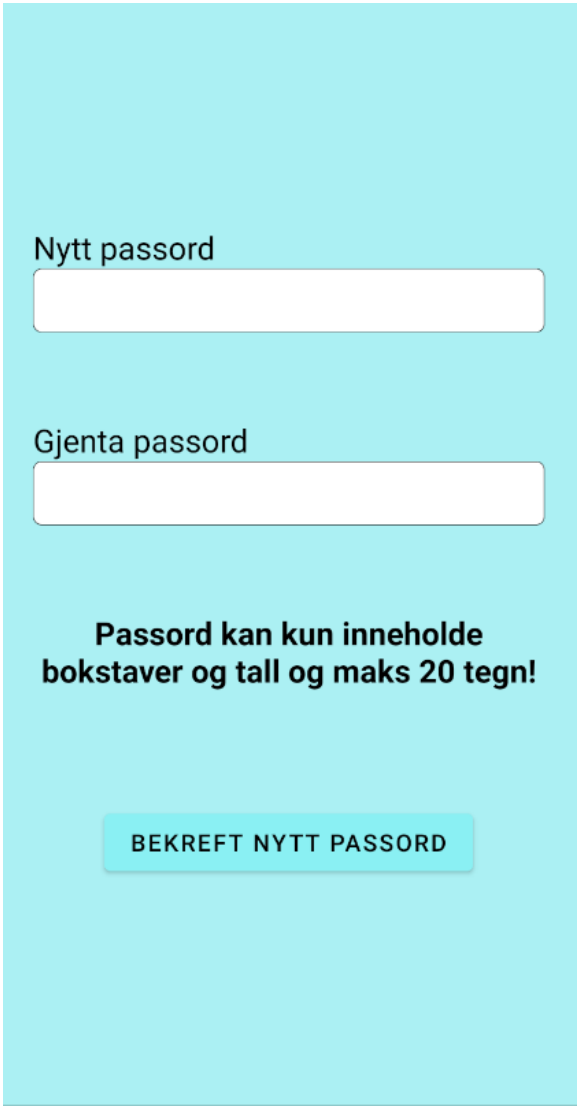


Når du er fornøyd, klikk *Logg inn* for å logge inn.



6.2.3 Bytte passord

Dersom du huket av for å endre passord under innloggingen, kommer du hit.

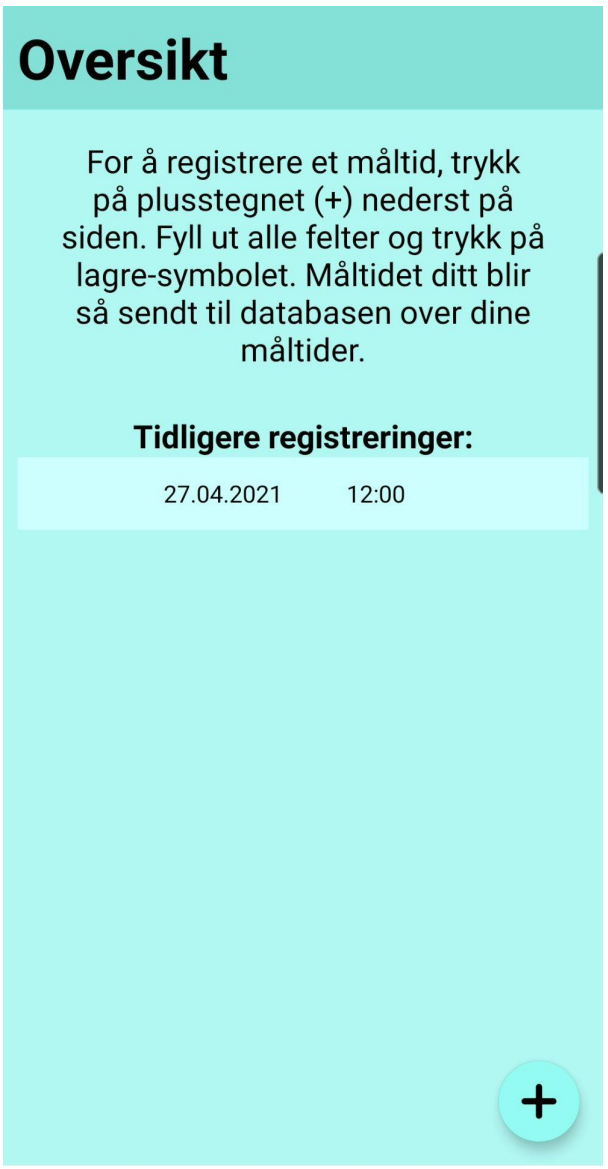


Skriv inn ønsket passord under *Nytt passord*, og gjenta dette under *Gjenta passord*. Når du er fornøyd, trykk *Bekreft nytt passord*.

Du blir nå tatt tilbake til innloggingen og kan logge på med det nye passordet umiddelbart.

6.2.4 Oversikt

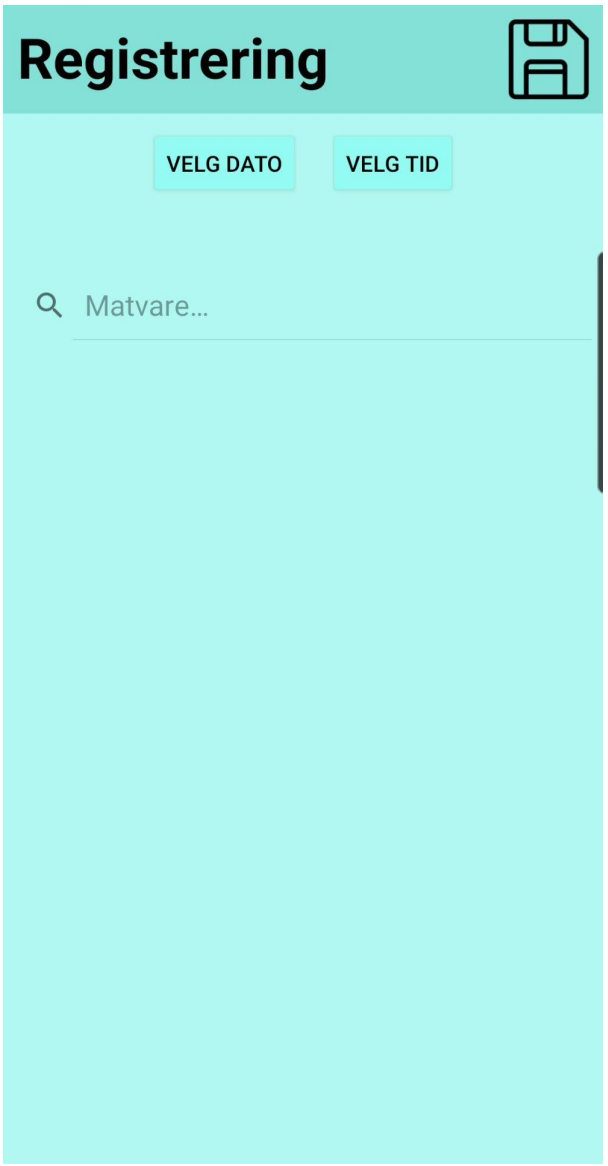
I oversiktssiden kan du se historikk for dine tidligere registreringer.



For å opprette en ny registrering kan du trykke på pluss-knappen (+) nederst til høyre på siden.

6.2.5 Registrering

Etter å ha trykket på pluss-knappen i oversiktssiden kommer du hit.

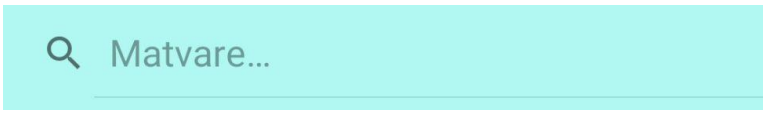


Velg dato og tid ved å trykke på knappene *Velg dato* og *Velg tid*.

Du får nå opp to dialogbokser hvor du kan velge ønsket dato og tidspunkt for måltidet.

Etter å ha valgt ønskelig tidspunkt, oppdateres teksten i knappene.

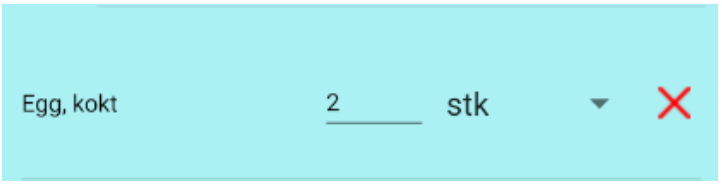
Søk etter ønsket matvare i søkefeltet.



Når du skriver inn noe, vises en liste med forslag under søkefeltet.

Trykk på ønsket matvare, og denne blir lagt til i en ny liste med dine valgte matvarer under søkefeltet.

Velg mengde og ønsket enhet for matvaren.



Du kan legge til flere matvarer om du ønsker ved å følge stegene over. Dersom du vil fjerne en matvare fra lista, trykk på det røde krysset.

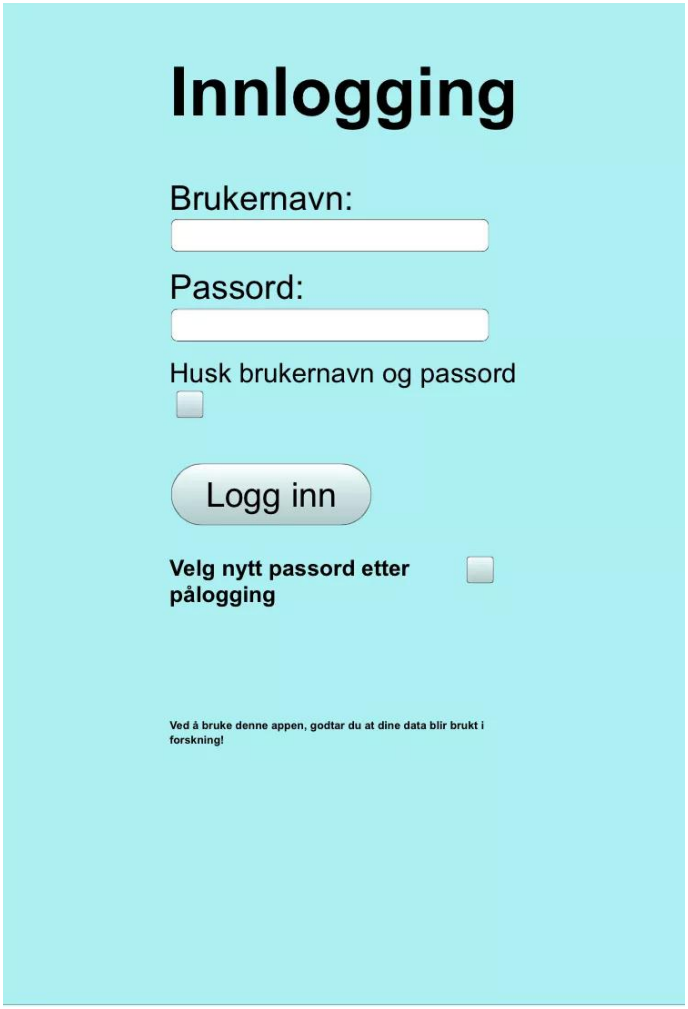
Når du er fornøyd, trykk på lagre-knappen øverst til høyre for å sende registreringen din.

Du blir nå tatt tilbake til oversiktssiden hvor du kan se at historikken din har blitt oppdatert.

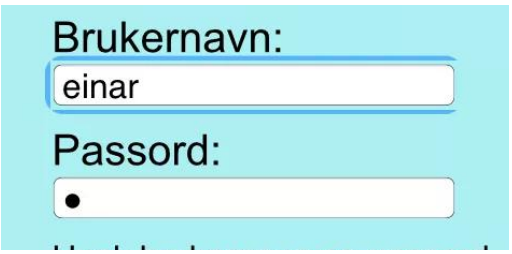
6.3 iOS

6.3.1 Innlogging

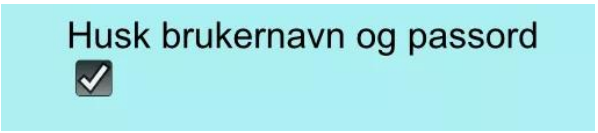
Etter å ha valgt iOS-versjonen kommer du rett til innlogging.



Skriv inn innloggingsopplysningene som ble oppgitt i din e-post.



Dersom du ønsker at applikasjonen skal huske innloggingsinformasjonen for deg, kan du huke av for dette i boksen *Husk brukernavn og passord*.

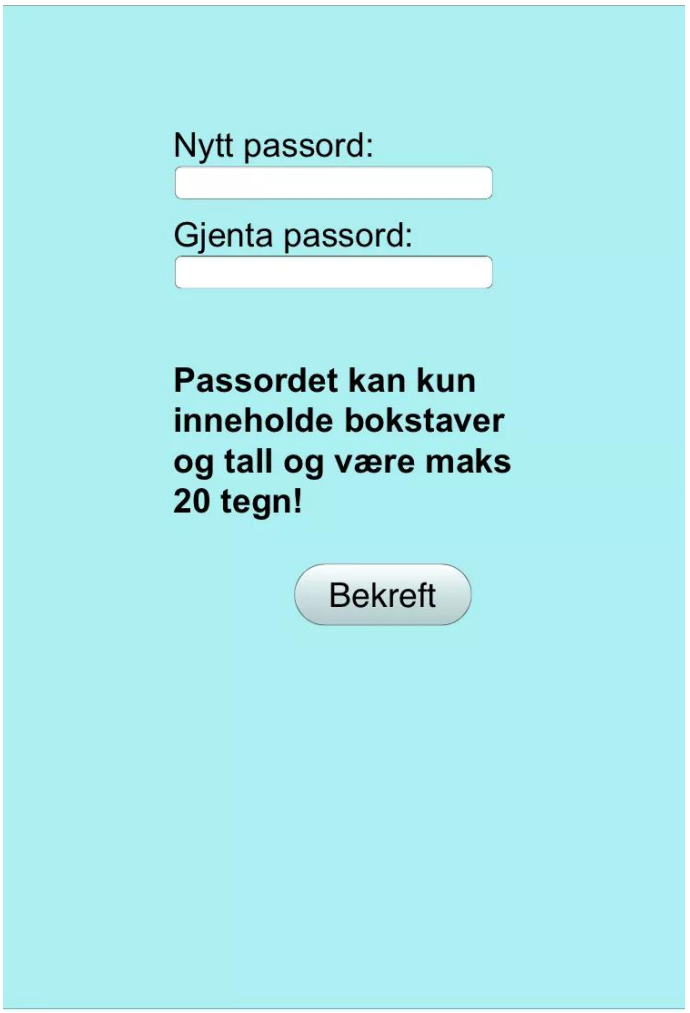


Dersom du ønsker å endre passord etter innlogging, kan du huke av for dette ved *Velg nytt passord etter pålogging*.

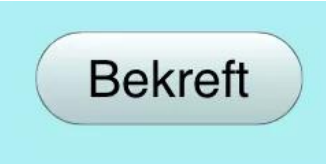
Når du er fornøyd, klikk *Logg inn* for å logge inn.

6.3.2 Bytte passord

Dersom du huket av for å endre passord under innloggingen, kommer du hit.



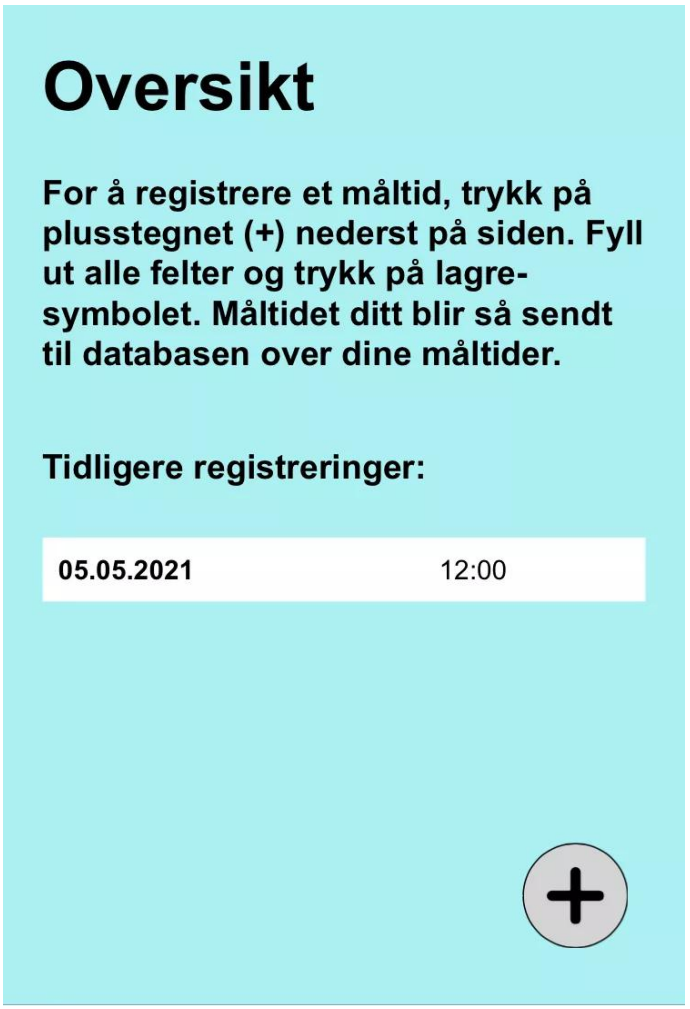
Skriv inn ønsket passord under *Nytt passord*, og gjenta dette under *Gjenta passord*. Når du er fornøyd, trykk *Bekreft*.



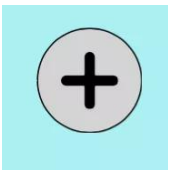
Du blir nå tatt tilbake til innloggingen og kan logge på med det nye passordet umiddelbart.

6.3.3 Oversikt

I oversiktssiden kan du se historikk for dine tidligere registreringer.



For å opprette en ny registrering kan du trykke på pluss-knappen (+) nederst til høyre på siden.

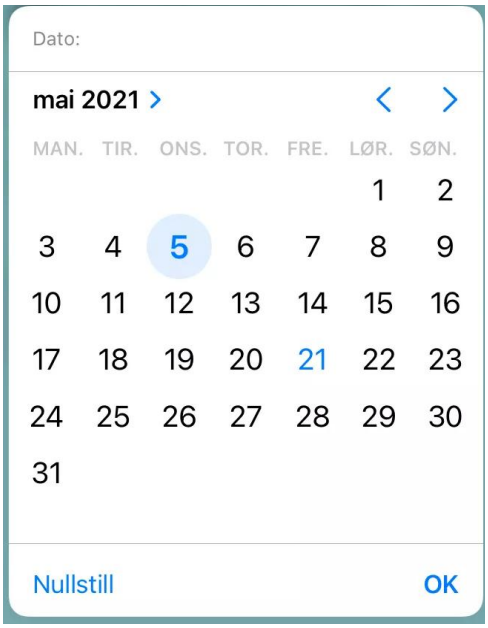


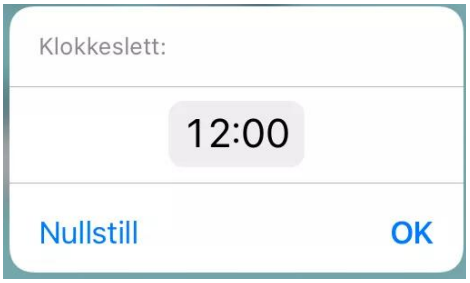
6.3.4 Registrering

Etter å ha trykket på pluss-knappen i oversiktssiden kommer du hit.

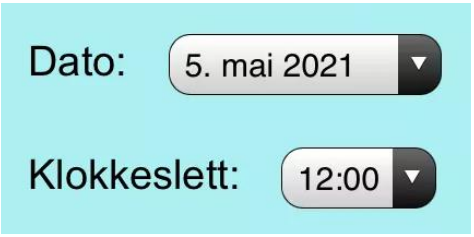
Velg dato og tid ved å trykke på feltene ved *Dato* og *Klokkeslett*.

Du får nå opp to dialogbokser hvor du kan velge ønsket dato og tidspunkt for måltidet.

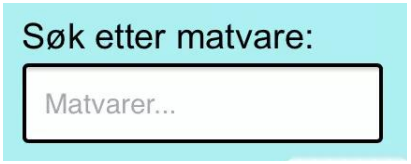




Etter å ha valgt ønskelig tidspunkt, oppdateres teksten i feltene.

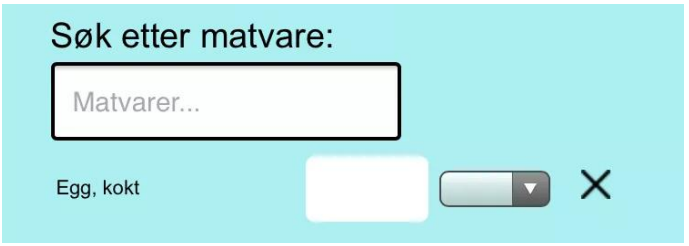


Søk etter ønsket matvare i søkefeltet.



Når du skriver inn noe, vises en liste med forslag under søkefeltet.

Trykk på ønsket matvare, og denne blir lagt til i en ny liste med dine valgte matvarer under søkefeltet.



Velg mengde og ønsket enhet for matvaren.

Du kan legge til flere matvarer om du ønsker ved å følge stegene over. Dersom du vil fjerne en matvare fra lista, trykk på det svarte krysset.

Når du er fornøyd, trykk på lagre-knappen øverst på siden for å sende registreringen din.

Du blir nå tatt tilbake til oversiktssiden hvor du kan se at historikken din har blitt oppdatert.

Dersom du skulle ønske å avbryte registreringen, kan du når som helst trykke på avbryt-knappen til høyre for lagre-knappen for å gå tilbake til oversikten.

7. Referanser

1. Alpha epsilon. (6.12.2017). *Personal Extreme Programming.* https://www.alpha-epsilon.de/programming/2017/12/06/personal-extreme-programming/
2. WebAIM. *Contrast Checker.* https://webaim.org/resources/contrastchecker/
3. Flaticon https://www.flaticon.com/
4. Matvaretabellen. (2020). *Mattilsynet*. www.matvaretabellen.no
5. Developer Android. (2021). *Shared Preferences.* https://developer.android.com/reference/android/content/SharedPreferences
6. Wikipedia. (2021). *Regular expression.* https://en.wikipedia.org/wiki/Regular_expression
7. W3 Schools. *HTML URL Encoding Reference.* https://www.w3schools.com/tags/ref_urlencode.ASP
8. W3 Schools. *How to create autocomplete.* https://www.w3schools.com/howto/howto_js_autocomplete.asp
9. Software Testing Help. (30.04.2021). *What is Unit Testing?* https://www.softwaretestinghelp.com/unit-testing/
10. Software Testing Help. (30.04.2021). *What is Integration Testing?* https://www.softwaretestinghelp.com/what-is-integration-testing/
11. Software Testing Help. (30.04.2021). *What is System Testing?* https://www.softwaretestinghelp.com/system-testing/
12. Software Testing Help. (30.04.2021). *What is User Acceptance Testing?* https://www.softwaretestinghelp.com/what-is-user-acceptance-testing-uat/
13. Guru 99. *7 Principles of Software Testing*. https://www.guru99.com/software-testing-seven-principles.html
14. Software Testing Fundamentals. *White Box Testing.* https://softwaretestingfundamentals.com/white-box-testing/
15. Software Testing Fundamentals. *Black Box Testing.* https://softwaretestingfundamentals.com/black-box-testing/
16. Babish, N. (22.09.2016) *Tips for Using Icons in Your App.* http://babich.biz/tips-for-using-icons-in-your-app/
17. PHP. *sqlsrv_connect.* https://www.php.net/manual/en/function.sqlsrv-connect.php

18. Microsoft. (03.09.2021) *System requirements for the Microsoft Drivers for PHP for SQL Server*. https://docs.microsoft.com/en-us/sql/connect/php/system-requirements-for-the-php-sql-driver?view=sql-server-ver15
19. SuperOffice. *Hva er GDPR, og hva betyr det for din bedrift?* https://www.superoffice.no/ressurser/artikler/hva-er-gdpr/
20. Uutilsynet. *Regelverk.* https://www.uutilsynet.no/regelverk/regelverk/266
21. Uutilsynet. *WCAG 2.0-standarden.* https://www.uutilsynet.no/wcag-standarden/wcag-20-standarden/86
22. Uutilsynet. *1.4.3 Kontrast.* https://www.uutilsynet.no/wcag-standarden/143-kontrast-minimum-niva-aa/95
23. Uutilsynet. *1.1.1 Ikke-tekstlig innhold*. https://www.uutilsynet.no/wcag-standarden/111-ikke-tekstlig-innhold-niva/87
24. NodeJS. *About Node.js*. https://nodejs.org/en/about/
25. Npm. *Crypto.js.* https://www.npmjs.com/package/crypto-js
26. Browserify. *Browserify*. https://browserify.org/
27. React Native. *React Native.* https://reactnative.dev/
28. Android Developers. *Fragments*. https://developer.android.com/guide/fragments
29. Cloudflare. *What is TLS (Transport Layer Security)?* https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/
30. Granevang, M. (31.07.2020). *Frontend* https://snl.no/frontend
31. Granevang, M. (31.07.2020). *Backend* https://snl.no/backend
32. Codecademy. *What is an IDE?* https://www.codecademy.com/articles/what-is-an-ide
33. Git. https://git-scm.com/
34. Imperva. *What is SQL Injection?* https://www.imperva.com/learn/application-security/sql-injection-sqli/

Forsidebilde fra Freepik:
https://www.freepik.com/free-photo/flat-lay-assortment-different-vegetables-with-copy-space_9906946.htm#page=1&query=Food&position=38