

# Campus Chat

ITF31619-1 25H Webapplikasjoner

Gruppe 2

**Blazej Zbigniew Jastrzebski**

**Karolina Gajos**

**Stefan Boytchev Strand**

<b>1. Introduksjon.....</b>	<b>4</b>
1.1 Beskrivelse av prosjektet.....	4
1.2 Motivasjon.....	4
1.3 Hvordan skal dette fungere?.....	5
1.4 Målgruppe.....	5
<b>2. Prosessen.....</b>	<b>6</b>
<b>3. Kravspesifikasjon.....</b>	<b>7</b>
3.1 Må-krav.....	8
3.1.1 Brukergrensesnitt og design.....	8
3.1.2 Roller og tilgang.....	8
3.1.3 Kommunikasjon.....	8
3.1.4 Profiler.....	8
3.2 Bør-krav.....	8
3.3 Kan-krav.....	9
3.4 Akseptansekrav.....	9
3.4.1 Autentisering og roller.....	9
3.4.2 Kommunikasjon (sanntid).....	10
3.4.3 Profil og innstillinger.....	10
3.4.4 Grupperom og tilgang.....	10
3.4.5 Varslinger.....	10
3.4.6 Responsivt design.....	10
3.5 Krav-mapping.....	11
3.5.1 Tabell: krav-akseptanse-mapping.....	11
3.5.2 Tabell: Krav-Skisser.....	12
<b>4. Systemdesign og tekniske detaljer.....</b>	<b>13</b>
4.1 Oppdeling av applikasjonen.....	13
4.1.1 Autentisering & Autorisasjon.....	14
4.1.2 Kommunikasjon.....	14
4.1.3 Grupper og prosjekter.....	14
4.1.4 Profiler.....	14
4.1.5 Varslinger.....	14
4.1.6 Filhåndtering.....	15
4.1.7 Backend og API.....	15
4.2 Design Patterns.....	15
4.2.1 Feature first organisering.....	15
4.2.2 Lagdeling og løse koblinger.....	15

4.2.3 Single Responsibility.....	15
4.2.4 Skalerbarhet og testbarhet.....	16
4.2.5 Trygghet og enkelhet.....	16
4.3 Valg av database.....	16
4.4 Vurdering av Convex.....	17
4.5 Roller i applikasjonen.....	18
4.7 Mappe og modulstruktur.....	18
4.8 Dataflow og ansvar.....	19
4.8.1 Leseflyt - (hente meldinger i et rom).....	19
4.8.2 Skriveflyt (sende melding).....	22
4.9 Sanntid.....	24
<b>5. Skisser.....</b>	<b>25</b>
5.1 Innlogging.....	25
Skisse-ID: M1 (Log inn Mob).....	26
Skisse-ID: T1 (Log inn tablet).....	27
Skisse-ID: D1 (Log in desktop).....	28
5.2 Profil og innstillinger.....	28
Skisse-ID: M2 (Innstillinger Mob).....	29
Skisse-ID: M2.1 (Profil mob).....	30
Skisse-ID: T2 (Profil/innst Tablet).....	31
Skisse-ID: D2 (Profil/innst Desk).....	32
5.3 Chat og samtaler.....	33
Skisse-ID: M3 (ChatOverview mob).....	34
Skisse-ID: M3P (PrivatChat Mob).....	35
Skisse-ID: T3 (Chats/privat tablet).....	36
Skisse-ID: D3 (Chats/privat desk).....	37
6. Diagram.....	38
6.1 Overordnet dataflow-diagram for applikasjonen.....	38
<b>7. Samarbeid.....</b>	<b>39</b>
<b>8.0 Prototype Alpha.....</b>	<b>39</b>
8.1 Database.....	39
Relasjoner i databasen.....	40
<b>9.0 Prototype Beta.....</b>	<b>42</b>
9.1 Arkitektur og Autentisering.....	42
9.2 Grupper.....	43
9.3 Profil.....	44

# 1. Introduksjon

## 1.1 Beskrivelse av prosjektet

Vi skal da utvikle **CampusChat**, en webbasert meldingsplattform laget spesielt for vår skole. Løsningen skal fungere som et raskt, enkelt men likevel pålitelig alternativ til allerede eksisterende løsninger. Blant annet som Teams, som oppleves ofte som tungvint, og Discord som ble nylig blokkert.

CampusChat vil tilby en globalchat for alle, egne grupper for fag og prosjekter, samt private meldinger for en-til-en samtaler. I tillegg kan brukere opprette profiler med navn, bilde og status, og selv bestemme hvor tilgjengelige de ønsker å være.

Vi ønsker å bygge en plattform som samler kommunikasjonen på ett sted. Altså uten unødvendig støy eller kompleksitet, men som likevel gir mulighet for funksjoner som filhåndtering, varslinger og sikker innlogging.

## 1.2 Motivasjon

Behovet for en egen kommunikasjonsløsning på skolen har vokst frem fordi de eksisterende alternativene ikke fungerer optimalt. Teams er tregt og oppleves uoversiktlig, og Discord har

blitt blokkert. Dette gjør det vanskelig å ha raske faglige diskusjoner, organisere prosjektarbeid eller bare ta en sosial prat.

CampusChat vil da være et verktøy som er laget for **oss**, av studenter som selv kjenner behovene. Vi får mulighet til å bygge en løsning som gir ekte verdi i hverdagen, samtidig som vi får erfaring med moderne webteknologier. Altså fra responsivt design til sanntidskommunikasjon og sikker autentisering.

Motivasjonen er også sterk fordi dette prosjektet kommer til å gi oss en verdifull mulighet til å bygge en fullstack-webapp fra bunnen av ved hjelp av Redwood SDK. Gjennom utviklingen vil vi få masse god praktisk erfaring med moderne webteknologier som React, TypeScript, databaser, autentisering og API-er. Vi vil også jobbe med sanntidsfunksjonalitet og sikkerhet, som er nok sentrale områder i dagens webutvikling. På denne måten får vi ikke bare god faglig utbytte, men også erfaring som er direkte relevant og nyttig når vi senere skal ut i arbeidslivet.

### 1.3 Hvordan skal dette fungere?

CampusChat skal fungere som en webapp tilgjengelig for mobil, nettbrett og desktop. Applikasjonen bygges mobile-first, men skal ha fullt responsivt design.

Hovedfunksjoner blir:

- Globalchat: En åpen kanal for alle brukere.
- Grupperom: For fag, prosjekter eller andre tema.
- Privatsamtaler: En-til-en samtaler.
- Profiler: Brukere kan sette navn, profilbilde, status, og velge personverninnstillinger.
- Roller: Studenter, lærere og eventuelle gruppeadministratører kan få ulike rettigheter.
- Sanntidsmeldinger: Når en melding sendes, dukker den opp med en gang hos mottaker(e).

## 1.4 Målgruppe

Målgrupper er ganske åpenbare for CampusChat. Først og fremst studenter og ansatte ved skolen. Systemet skal bli laget for faglige diskusjoner, prosjektarbeid, og ikke minst sosial prat. Systemet skal oppleves som raskt, enkelt og tilgjengelig.

## 2.Prosessen

De tre idéene som vi leverte til første obligatoriske har sin historie og begrunnelse. Først litt kort om de to som vi ikke valgte å gå for, så forklaringen på hvorfor vi ønsket å gå for CampusChat.

### **Job application tracker:**

Absolutt en sterk kandidat som har sin rot i at det fort kan bli uoversiktlig i perioder med intensiv jobbsøking. Det å samle alle data i en applikasjon som gjør kontakter, søknader, datoer, deadlines og så videre, var noe alle i gruppen var enig var savnet.

### **MedSched - Planlegging av vakter:**

Det er naturlig at en idé om IT-forbedringer innen helsevesenet er et av alternativene når det er to sykepleiere i gruppen. MedSched skulle optimalisere planlegging av turnus, ekstravakter, samt andre ressurser. Bemannings-styring fremst tiltenkt for sykehus og helsevesen, men kunne også blitt brukt i andre yrker med turnusordning.

## CampusChat

Idéen om en chat-webapp dukket ganske fort opp som alternativ under selve brainstormingen til hva vi kunne tenke oss å utvikle. Mye av grunnen til dette var nok det evige problemet med Discord og at dette er en uoffisiell kanal som studenter (og forelesere) benytter seg av, men som ikke dekker kravene om personvern og sikkerhet som stilles fra høgskolens side. For oss studenter, og mest sannsynlig for lærere også, oppleves det som rotete med de forskjellige kanalene som per i dag brukes. Det er mail, det er kunngjøringer på canvas (som også fås som mail i hiof-mailen) og det er Discord. Discord skal egentlig ikke brukes, men når denne applikasjonen tilbyr det den gjør så blir det veldig enkelt å ta den i bruk likevel.

Valget falt på CampusChat av sammensatte årsaker. Det er et veldig reelt problem i dag. Det finnes en meget stor kundebase som kan ta en slik app i bruk. Ikke minst var det en god utfordring rent utviklingsmessig for oss. Vi satser høyt og tenker at dette er et perfekt tilfelle å prøve å løse et eksisterende problem samtidig som vi får god trening i fullstack-utvikling.

Job application tracker føltes litt for mye som en “to-do” app, som kanskje ikke hadde gitt oss den utfordringen vi søker og det nivået vi strever etter. Det var nok heller ikke det problemet som plaget oss mest, dermed ikke like motiverende som CampusChat.

MedSched kunne definitivt blitt en utfordring, muligens for stor for tidsrammen vi opererer med i faget. Absolutt en reell plage, men for å sikre at vi som gruppe leverer best mulig tenkte vi at et problem som alle har mer eller mindre lik tilknytning til, vil være det som også gir mest motivasjon i det lange løpet.

## 3.Kravspesifikasjon

Kravspesifikasjonen skal da beskrive funksjoner og egenskaper som CampusChat skal ha. For å sikre god fremdrift og riktig prioritering har vi delt kravene inn i tre nivåer:

- Må-krav: Absolutte krav som må være på plass for at appen skal kunne fungere slik den er ment.
- Bør-krav: Viktige krav, men ikke kritiske. De skal implementeres dersom tid og ressurser tillater det.
- Kan-krav: Ekstra funksjonalitet som ikke er noe særlig nødvendig i grunnversjonen, men som kan øke brukeropplevelsen og være aktuelle for senere utvidelser.

### 3.1 Må-krav

Disse kravene er da som nevnt grunnleggende for at CampusChat skal fungere som en meldingsplattform:

#### 3.1.1 Brukergrensesnitt og design

- Responsivt design (mobil, nettbrett, desktop) - Mobile-first

#### 3.1.2 Roller og tilgang

- Støtte for ulike roller (student/bruker, foreleser, gruppeadmin)

#### 3.1.3 Kommunikasjon

- Globalchat for alle brukere
- Sanntidsmeldinger (meldinger dukker opp umiddelbart)
- Private meldinger
- Grupperom for fag, prosjekter eller andre tema

#### 3.1.4 Profiler

- Profilhåndtering (navn, bilde, status)
- Innstillinger (bestemme tilgjengelighet, synlighet osv)

## 3.2 Bør-krav

Disse kravene vil gjøre appen mer komplett og brukervennlig, men er ikke kritiske for første versjon.

- Kommunikasjon
  - Mulighet til å redigere og slette egne meldinger
  - Varslinger (notifiseringer i nettleser eller push)
- Filhåndtering
  - Mulighet for å sende og motta filer (dokumenter, bilder osv)
- Sosiale funksjoner
  - Vennefunksjon (legg til og organisere kontakter)
  - Blokkere andre brukere
- Sikkerhet
  - Kryptering av meldinger (under transport)
  - Multifaktor-autentisering for pålogging
  - Grunnleggende sikkerhet

## 3.3 Kan-krav

Disse funksjonene er ekstra og kan vurderes i videre utvikling:

- Brukertilpasning
  - Språkvalg (eller automatisk språk basert på nettleser)
  - Tilpasning av grupper (gruppens bilde og beskrivelse)
  - Dark mode
- Avansert kommunikasjon
  - Audio- og video samtaler
  - Reaksjoner på meldinger (emoji, likes osv)

## 3.4 Akseptansekrav

### 3.4.1 Autentisering og roller

- Brukeren skal kunne registrere og logge inn med e-post og passord.
- Feil innloggingsinformasjon skal alltid gi en tydelig feilmelding.
- Innlogget bruker får korrekt rolle (student/foreleser/admin(?)) og ser kun funksjoner de har tilgang til.

### 3.4.2 Kommunikasjon (sanntid)

- Når en melding sendes i globalchat, grupperom eller privat chat, skal den vises for mottaker(e) "med en gang" ved en stabil nettforbindelse.
- Meldinger skal vises i riktig kronologisk rekkefølge med tydelig avsender og tidsstempel.
- Redigering/sletting av egen melding skal oppdatere UI hos alle.
- Uautoriserte brukere skal ikke kunne lese private samtaler eller lukkede grupperom.

### 3.4.3 Profil og innstillinger

- Bruker skal kunne oppdatere navn, profilbilde, status. Endringer vises umiddelbart i chat-UI for andre.
- Bruker kan sette tilgjengelighet (online/opptatt/synlig); status speiles i brukerliste.
- Varslingsinnstillinger kan slås av/på.

### 3.4.4 Grupperom og tilgang

- Bruker kan opprette et grupperom.
- Private grupperom skal kun være synlige og tilgjengelige for medlemmer; invitasjon/forespørrelse må aksepteres.
- Meldingshistorikk i et rom lastes paginert (altså f.eks 50 meldinger per last).

### 3.4.5 Varslinger

- Ved ny melding i en aktiv samtale skal samtalen “hoppe” til siste melding uten reload
- Når appen er i bakgrunnen og varsle er aktivert skal nettleservarsel vises med avsender.

### 3.4.6 Responsivt design

- Appen skal fungere og se korrekt ut på mobil, tablet og desktop, uten at funksjonalitet går tapt.
- Layout skal tilpasses seg skjermstørrelse automatisk. Kan testes ved simulering

## 3.5 Krav-mapping

Tabellen viser hvordan de ulike kravene kan oversettes til konkrete akseptansekravene

### 3.5.1 Tabell: krav-akseptanse-mapping

Må-Krav-ID	Krav (må-krav)	Akseptansekrav	Akseptanse-ID
3.1.1	Responsivt design	Appen skal fungere og se korrekt ut på mobil, tablet og desktop, uten at funksjonalitet går tapt.	3.4.6
	Mobil, nettbrett, desktop Mobile-first	Layout skal tilpasses seg skjermstørrelse automatisk. Kan testes ved simulering	
3.1.2	Roller og tilgang	Bruker skal kunne registrere seg og logge inn med e-post og passord	3.4.1
	Støtte for ulike roller: student/foreleser	Feil innloggings-informasjon skal alltid gi en tydelig feilmelding  Innlogget bruker får korrekt rolle	

3.1.3	<p>Kommunikasjon (sanntid)</p> <p>Globalchat for alle brukere</p> <p>Sanntidsmeldinger</p> <p>Private meldinger</p> <p>Grupperom for fag, prosjekter/annet</p>	<p>Når en melding sendes i globalchat, grupperom eller privat chat, skal den vises for mottaker(e) "med en gang", ved stabil nettforbindelse</p> <p>Meldinger vises i riktig kronologisk rekkefølge med tydelig avsender og tidsstempel.</p> <p>Redigering/sletting av melding skal oppdatere UI hos alle.</p> <p>Uautoriserte brukere skal ikke kunne lese private samtaler eller lukkede grupperom</p>	3.4.2
3.1.4	Profiler og innstillinger	<p>Bruker skal kunne oppdatere navn, profilbilde, status. Endringer vises umiddelbart i chat-UI for andre.</p> <p>Bruker kan sette tilgjengelighet (online/opptatt/synlig): status speiles i brukerliste</p> <p>Varslingsinnstillinger kan slås av/på.</p>	3.4.3

### 3.5.2 Tabell: Krav-Skisser

Må-Krav-ID	Krav (må-krav)	Skisse-ID	Kommentar
3.1.1	Responsivt design  Mobil, nettbrett, desktop (mobile first)	<a href="#">M1 (Log inn mob)</a>  <a href="#">T1 (Log inn tablet)</a>  <a href="#">D1 (Log inn Desktop)</a>	
3.1.2	Roller og tilgang - Støtte for ulike roller(student/lærer).	<a href="#">M1 (Log inn mob)</a>  <a href="#">T1 (Log inn tablet)</a>  <a href="#">D1 (Log inn Desktop)</a>	Rolle velges ved registrering. Skisser viser ikke dette per nå, men linker her til log-in screen da vi har valg av rolle der sånn det ser ut nå.
3.1.3	Kommunikasjon -  Globalchat for alle brukere  Sanntidsmeldinger  Private meldinger  Grupperom for fag, prosjekter/annet	<a href="#">M3 (ChatOverview mob)</a>  <a href="#">M3P (PrivatChat Mob)</a>  <a href="#">T3 (Chats/privat Tablet)</a>  <a href="#">D3 (Chats/privat desk)</a>	
3.1.4	Profiler -  Profilhåndtering (navn, bilde, status)  Innstillinger (bestemme tilgjengelighet, synlighet, m.m)	<a href="#">M2.1 (Profil mob)</a>  <a href="#">M2 (Innstillinger Mob)</a>  <a href="#">T2 (profil/innst Tablet)</a>  <a href="#">D2 (Profil/innst Desk)</a>	

# 4. Systemdesign og tekniske detaljer

## 4.1 Oppdeling av applikasjonen

Mye av dette vises allerede i skissene våre, og mye er egentlig fortsatt litt uklart. Vi er åpne for endringer underveis fordi ting dukker naturligvis opp under arbeidet med applikasjonen. Det som i hvert fall er tankegangen er:

### 4.1.1 Autentisering & Autorisasjon

Denne modulen håndterer registering og innlogging av brukere, samt rollefordeling mellom studenter, forelesere og gruppeadmins. Her mener vi at vi har lyst å sikte mot JWT, men vi får se hva vi går for. Her ivaretas i hvert fall sikker pålogging, sesjonshåndtering og tilgangskontroll slik at brukerne får riktig funksjonalitet ut fra sin rolle.

### 4.1.2 Kommunikasjon

Kjernen i hele applikasjonen vår er jo selvfølgelig meldingssystemet. Dette inkluderer globalchat, grupperom og private meldinger. Systemet bygges på sanntid, slik at meldinger vises umiddelbart. Her håndteres også redigering og sletting av meldinger.

### 4.1.3 Grupper og prosjekter

Denne modulen gjør det mulig å opprette egne grupperom for fag, prosjekter eller andre tema. Gruppeadmins får mulighet til å invitere medlemmer og administrere dem.

#### 4.1.4 Profiler

Brukerne våre får da en egen profil der de kan oppgi navn, bilde, status, innstillinger. Det legges til rette for tilstedeværelsесindikatorer (online, opptatt, synlig) slik at andre ser hvem som er tilgjengelig.

#### 4.1.5 Varslinger

Varlingsmodulen vil da gi brukere beskjed om nye meldinger eller andre “aktiviteter”.

#### 4.1.6 Filhåndtering

Her vil filene håndteres. Altså opplasting og nedlasting av filer som dokumenter eller bilder. Modulen skal sørge for enkel deling i samtaler, samtidig som sikkerhet i lagringskapasitet ivaretas.

#### 4.1.7 Backend og API

Backendmodulen er så klart bygget med Redwood SDK på Cloudflare. Det vil gi oss sanntid, server-side rendering og sikker datalagring.

### 4.2 Design Patterns

#### 4.2.1 Feature first organisering

Vi organiserer koden etter funksjoner. Det vil si at alt som har med “chat” å gjøre ligger sammen og det samme med “auth”, “profile”, og så videre. Det betyr at skjermbilder, komponenter, API-kall og tjenester for en funksjon ligger i samme mappe. Fordelen er at det blir enklere å finne frem, og vi kan jobbe parallelt med ulike funksjoner uten å forstyrre hverandre.

#### 4.2.2 Lagdeling og løse koblinger

Vi deler koden i lag. Brukergrensesnittet (UI) snakker bare med hooks. Hooks snakker med API. API-laget snakker med services/repository. Til slutt er det services som faktisk henter og lagrer data. Denne oppdelingen gjør at vi kan bytte ut en del (f.eks databasen) uten å måtte endre alt. Det gjør også at det er lett å teste deler av systemet isolert.

#### 4.2.3 Single Responsibility

Hver fil skal ha et tydelig ansvar. En komponent viser bare data, en hook henter data, og en service snakker med databasen. Det gjør at koden blir lettere å forstå, vi unngår konflikter når flere jobber med samme kode og det blir enklere å rette opp feil.

#### 4.2.4 Skalerbarhet og testbarhet

Da vi bruker services/repository som mellomlag, kan vi lett lage tester der vi bytter ut databasen med falsk data (mocking). På denne måten kan vi teste logikk uten å være avhengig av ekte database. Hvis vi en dag ønsker å bytte database kan vi gjøre det uten å endre brukergrensesnittet.

#### 4.2.5 Trygghet og enkelhet

Vi velger enkle og pålitelige løsninger. SQLite er en lettvektsdatabase som krever lite oppsett og passer perfekt for et skoleprosjekt. I tillegg har vi enkle regler for tilgangskontroll (guards i core/auth) og håndtering av sesjoner i en egen mappe (session/). Dette gjør at vi raskt får en fungerende løsning som også kan vokse videre.

## 4.3 Valg av database

Vi har valgt å bruke SQLite. Det er flere grunner til det:

- Lett å sette opp
- Ingen serverdrift nødvendig
- Passer fint for et skoleprosjekt
- Rask utvikling lokalt
- Kan skaleres opp hvis vi kjører på f.eks Cloudflare D1.

Grov modell:

- **User**: id, e-post, navn, avatar, rolle, opprettetDato
- **Room**: id, navn, privat/offentlig, opprettetAv, opprettetDato
- **Membership**: hvilken bruker er medlem av hvilket rom
- **Message**: id, romId, avsenderId, tekst, tidspunkt
- **Presence**: ikke lagres permanent, men snapshots ved behov

Relasjoner håndheves i Drizzle. Muligens lager vi indekser på romId og createdAt for få raskere søk i meldingshistorikken.

## 4.4 Vurdering av Convex

Vi har vurdert Convex, som er en “alt-i-ett” tjeneste for database og sanntid.

Fordelene her er:

- Enklere oppsett
- Automatisk oppdatering av klienter
- Bra verktøystøtte

Ulemper:

- Vi blir låst til en “leverandør”
- Mindre kontroll på hvordan ting virker

Vi velger heller WebSocket + SQLite nå. Det gir oss kontroll og er enkelt nok. Men arkitekturen vår er laget slik at vi kan bytte til Convex senere hvis vi vil, uten å måtte endre alt.

## 4.5 Roller i applikasjonen

Rollene vi tenker å implementere, i hvert fall med det vi vet per i dag, er:

- Student
- Foreleser

Admin er noe vi har diskutert, men vi lar dette stå ubesvart akkurat nå. Vi får se hvordan applikasjonen utvikler seg om om det i hele tatt blir nytte for en slags admin-bruker.

## 4.7 Mappe og modulstruktur

Vi har valgt en struktur som gir oss oversikt:

```
src/
  app/          # ramme
    Document.tsx      # HTML-shellet vårt importerer tailwind
    routes.ts        # kodebasert ruteoppsett (ikke filnavn som I mobil)
    pages/          # tverrgående sider (Home/404)
    providers/       # globale provider f.eks Auth, Theme
    index.css        # tailwind
  features/
    auth/           # login/registrering/roller
      screens/ components/ hooks/ api/ services/
    chat/            # global/rom/DM
      screens/ components/ hooks/ api/ services/
    groups/          # romopprettelse, medlemskap
    profile/         # profil & innstillinger
  core/
    models/          # User, Room, Message, Role...
    lib/             # utils: tidsformat, validering, id, pagination
    auth/            # guards/rollelogikk
    realtime/        # reconnect, config av realtime meldinger?
  server/
    functions/       # server actions/loaders
    do/              # durable objects for chat/presence (muligens skrape?)
    db/
      schema.ts     # Drizzle (sqlite) - tabeller
      client.ts     # drizzle-client
      seed.ts       # seed-data (lokal/prod)
  session/          # enkel sesjon/cookie-håndtering muligens
  types/            # typer (vite/rw)
  worker.tsx        # kjernen av det hele
```

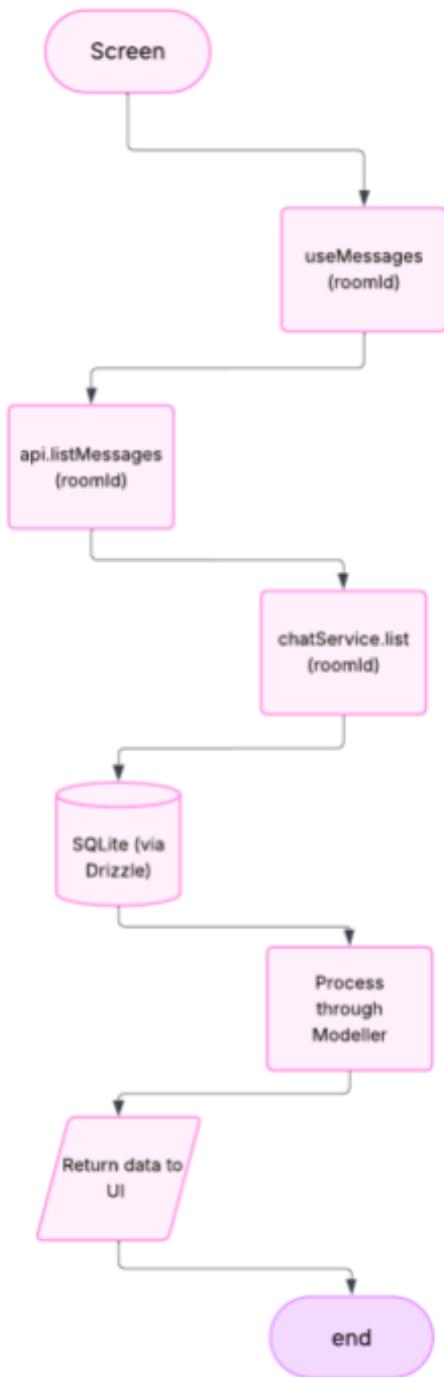
## 4.8 Dataflow og ansvar

### 4.8.1 Leseflyt - (hente meldinger i et rom)

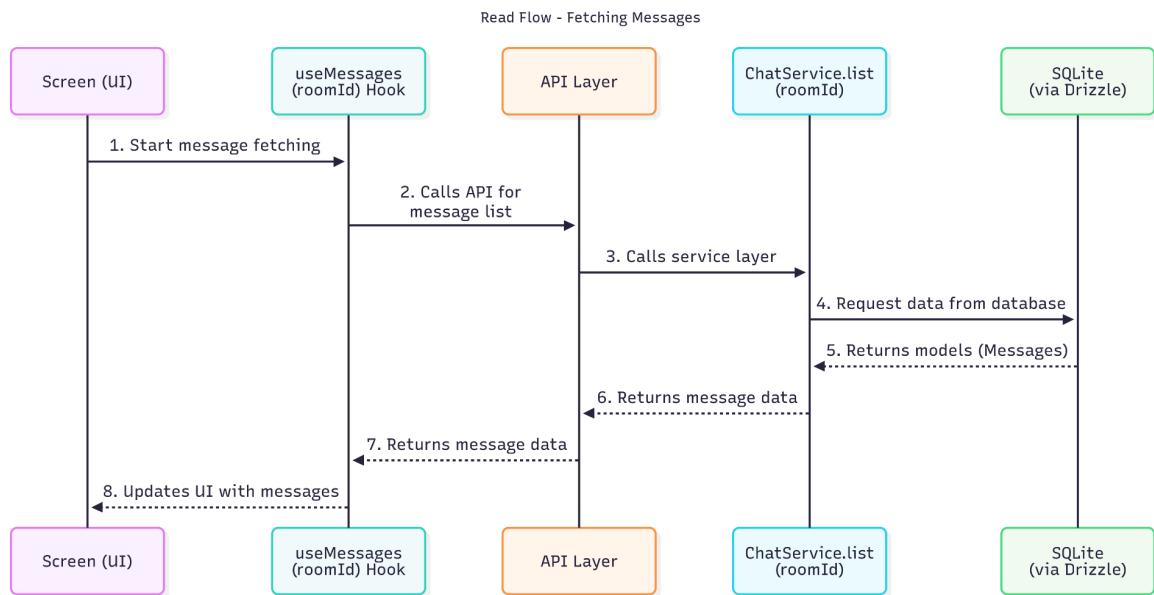
Screen → useMessages(roomId) → api.listMessages(roomId) → chatService.list(roomId) → SQLite (via Drizzle) → modeller → tilbake til UI.

Her beskriver vi hvordan data flyter fra databasen til skjermen. UI trenger ikke å vite hvordan databasen ser ut. Diagram og sekvensdiagram vises på neste side.

Diagram LeseFlytDataFlow:

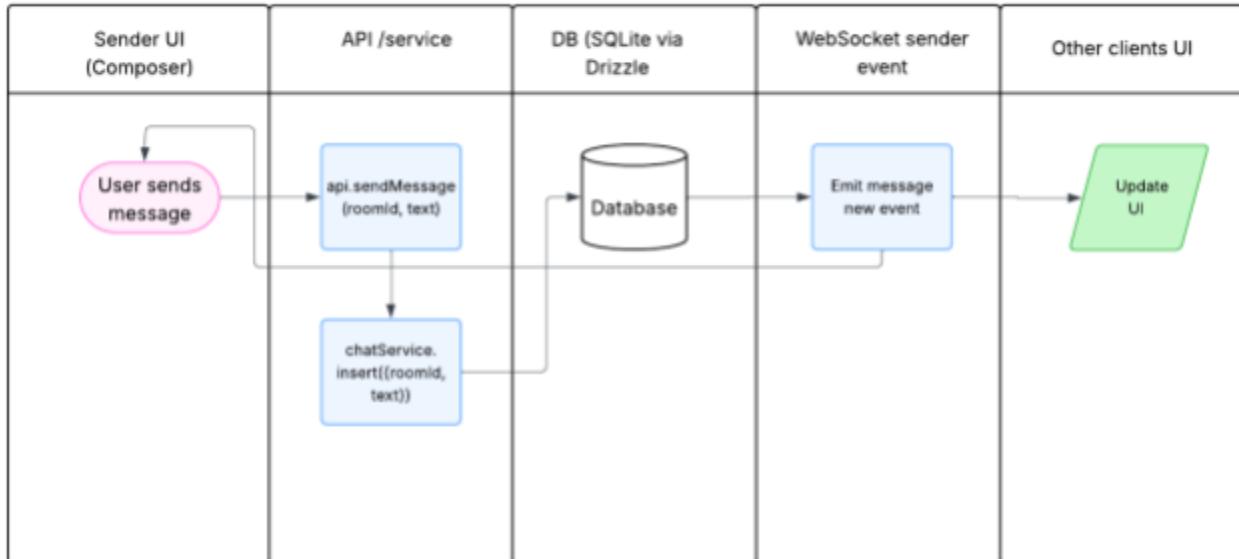


## Sekvensdiagram LeseFlyt.

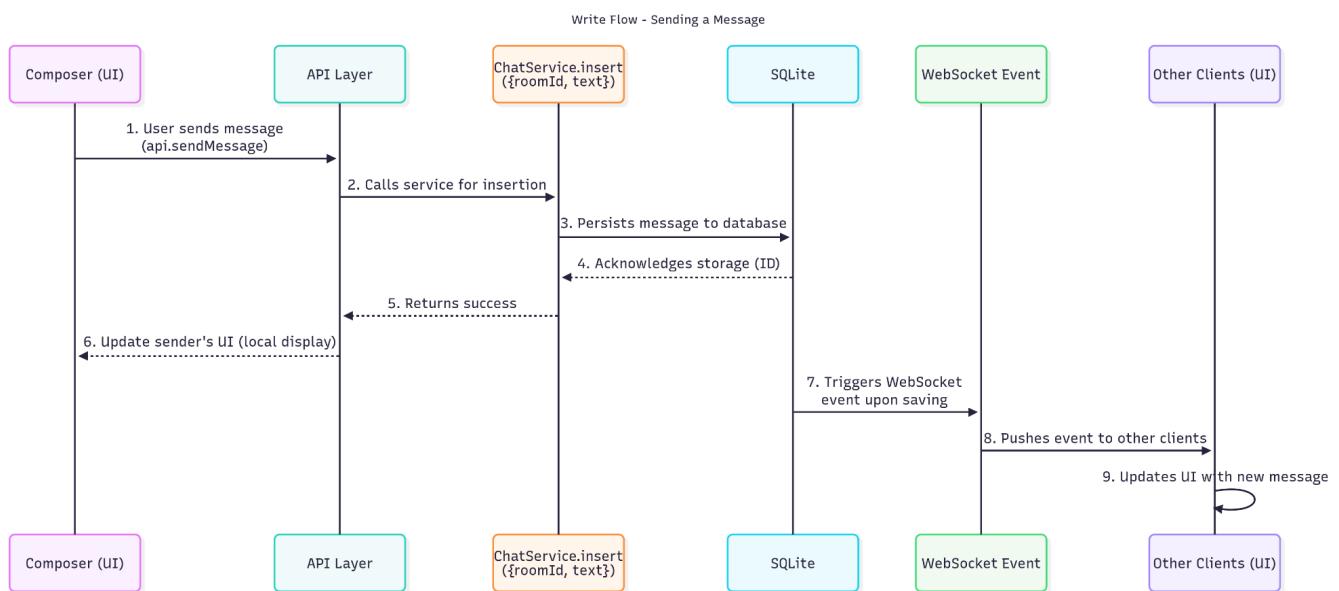


## 4.8.2 Skriveflyt (sende melding)

Composer → api.sendMessage(roomId, text) → chatService.insert({ roomId, text }) → SQLite  
 → WebSocket sender event → andre klienter oppdateres.



Sekvensdiagram skriveflyt.



## 4.9 Sanntid

For å støtte chat i sanntid bruker vi WebSocket.

Valgt nå: en enkel WebSocket-gateway i Worker (`/ws/:roomId`). Den holder oversikt over tilkoblinger i hvert rom og sender meldinger videre til alle.

Alternativ senere: Durable Objects. Da kan vi ha en “objekt-server” per rom som sørger for garantert rekkefølge, reconnect-håndtering og presence (hvem som er pålogget). Dette er mer avansert og kan legges til senere uten å endre hele arkitekturen.

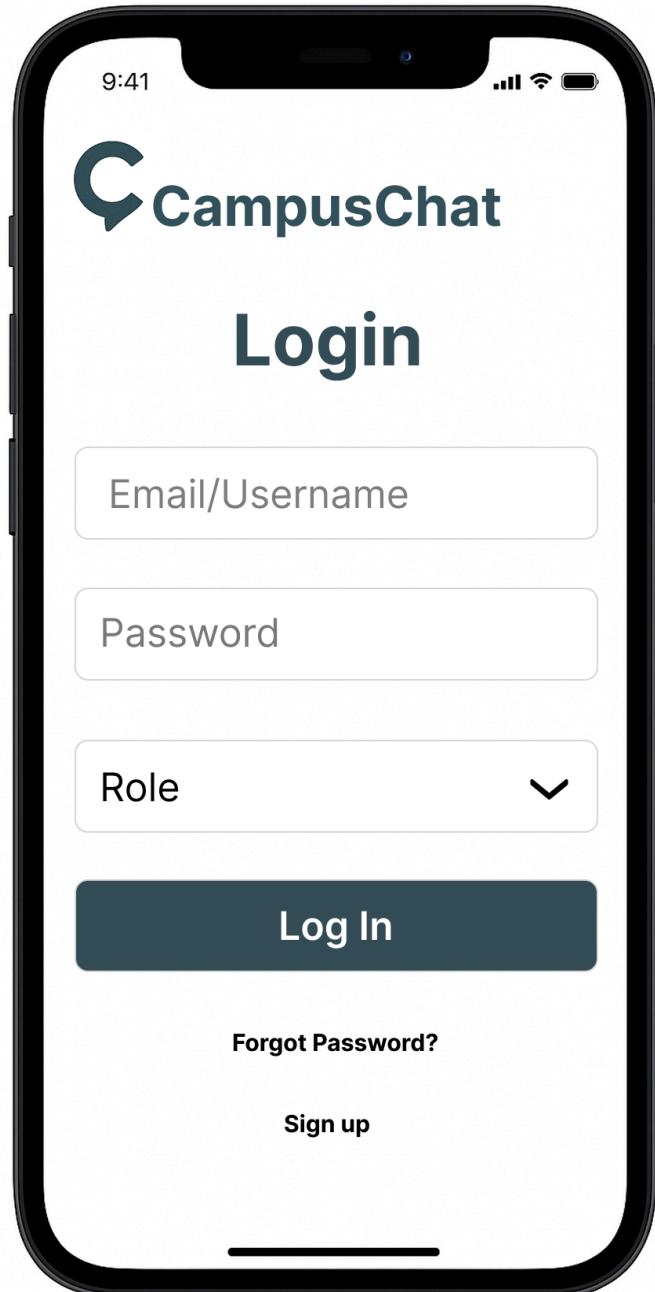
## 5.Skisser

### 5.1 Innlogging

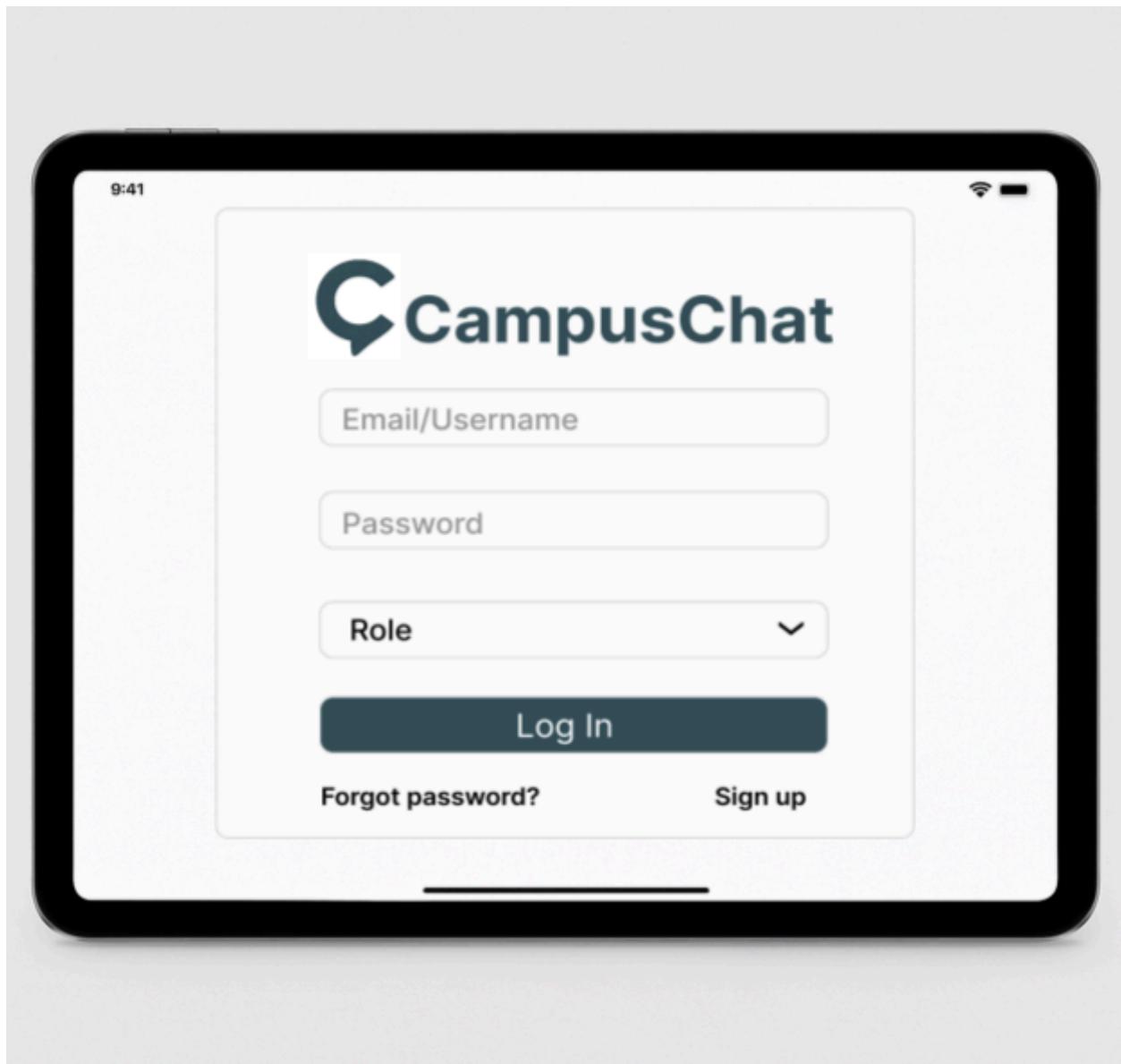
Innloggingsskjermen er første møtepunktet med CampusChat. Brukeren oppgir e-post/username og passord. Ved registrering blir valg av rolle tilgjengelig også. (Knappen for valg av rolle i innloggingsskjermen er ment til å kun vise hvordan det vil se ut for registrering.).

Skjermen er laget responsivt og tilpasser seg mobil, nettbrett og desktop. Som nevnt før, så skal vi følge mobile-first prinsippet.

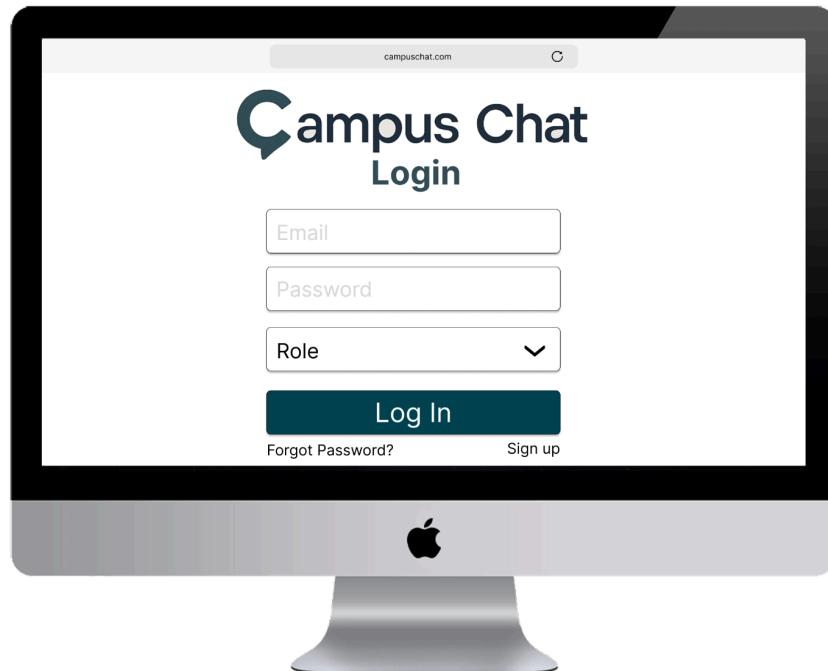
Skisse-ID: M1 (Log inn Mob)



Skisse-ID: T1 (Log inn tablet)



Skisse-ID: D1 (Log in desktop)

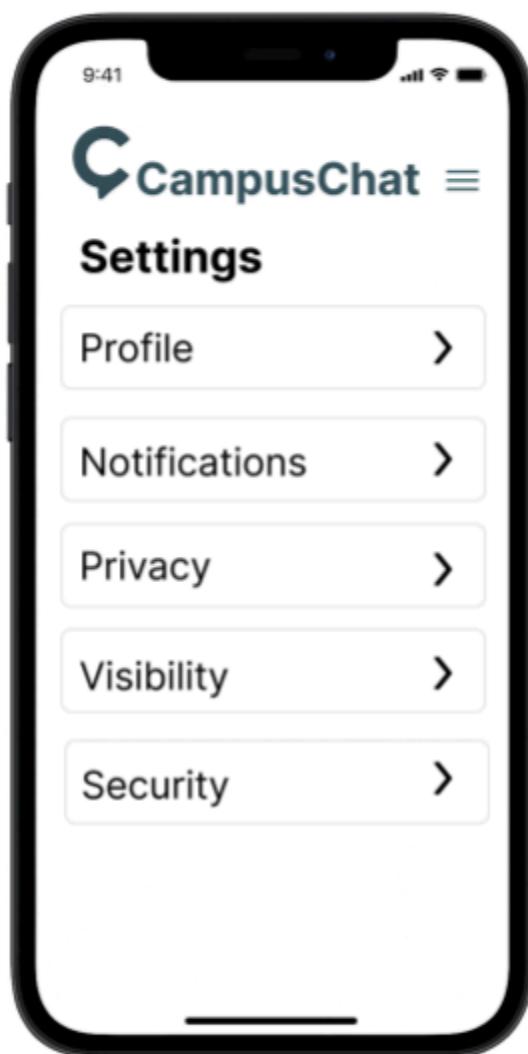


## 5.2 Profil og innstillinger

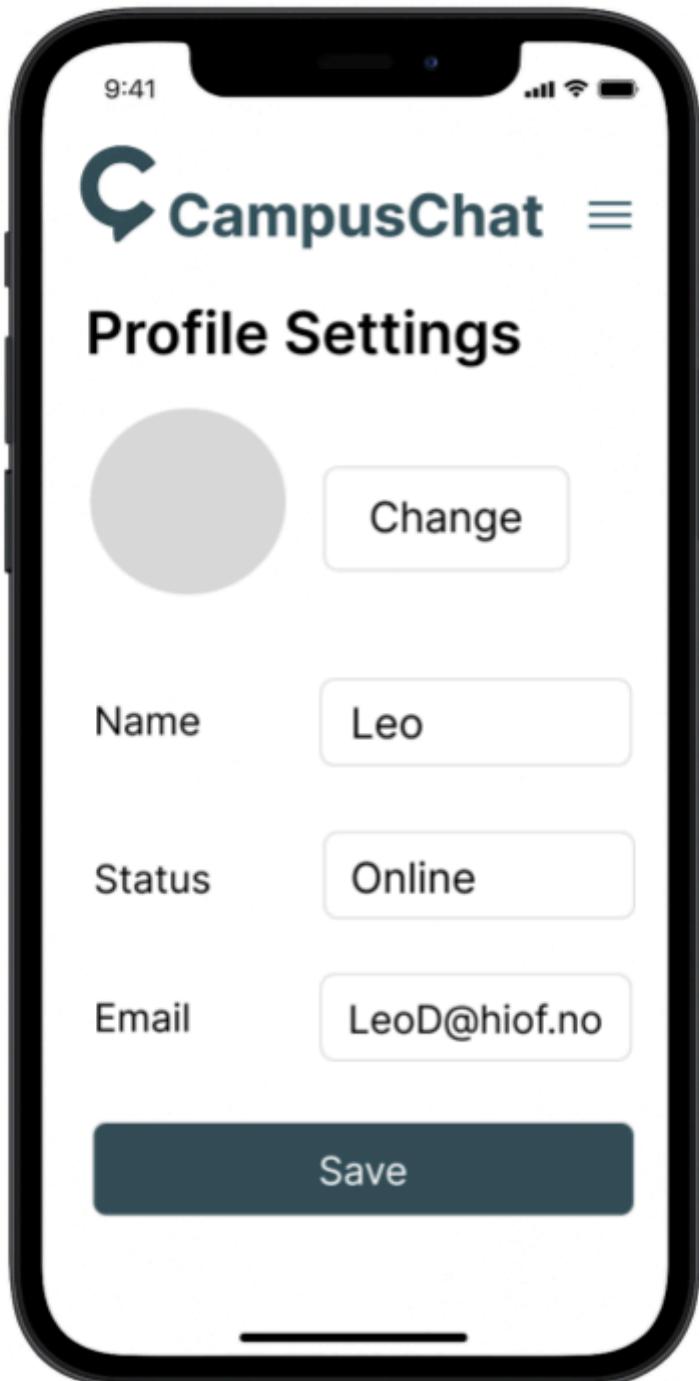
Her kan brukerne tilpasse sin egen profil med navn, status, e-post og profilbilde. Innstillinger gir tilgang til valg for varslinger, personvern, synlighet og sikkerhet.

Layouten er mobilvennlig med en enkel listavisning, mens nettbrett og desktop får en mer oversiktlig sidemeny. Dette vil gjøre navigasjonen enkel uansett enhet, samtidig som det viser tydelig at CampusChat-appen er laget med responsivt design i fokus.

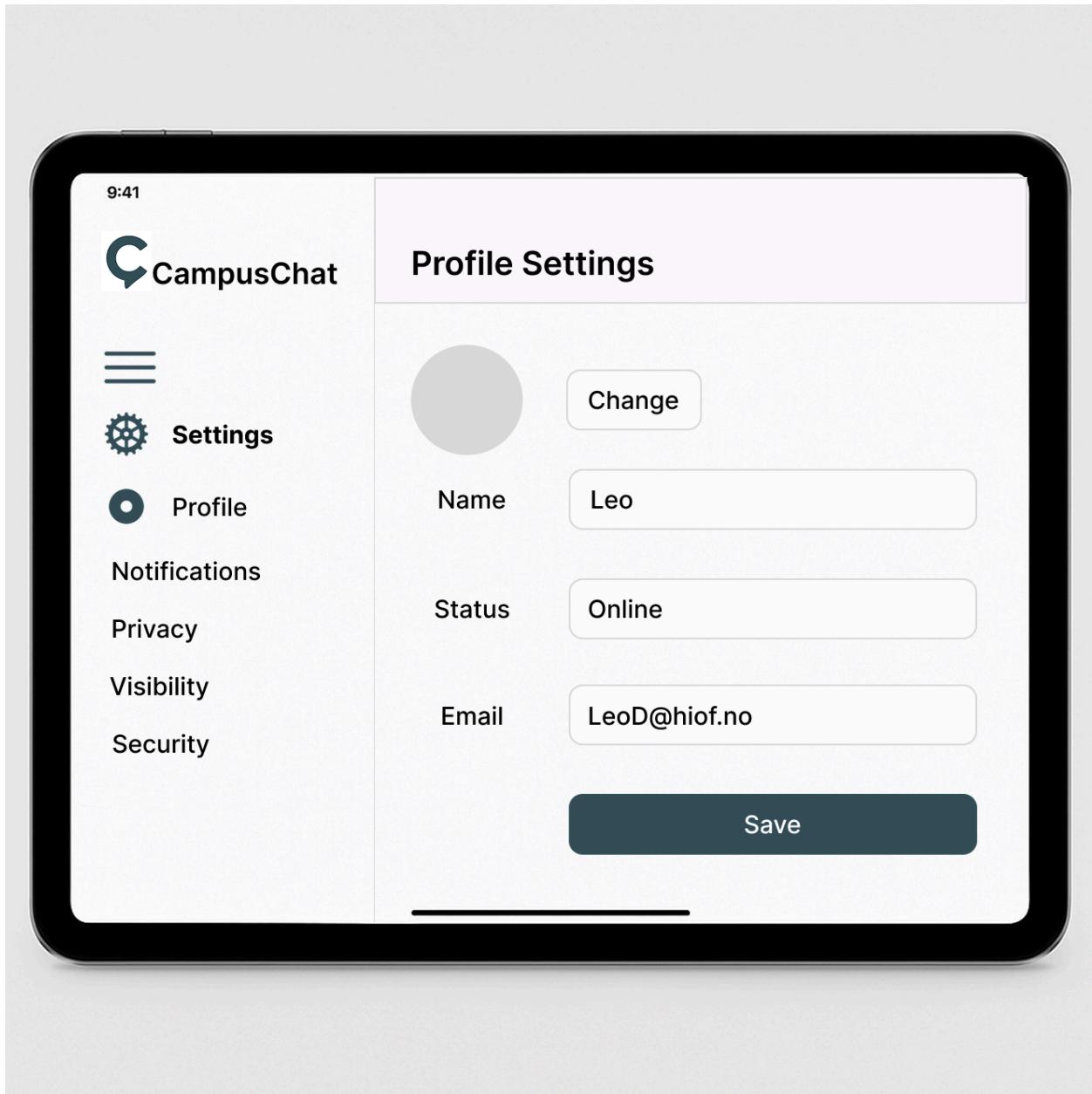
Skisse-ID: M2 (Innstillinger Mob)



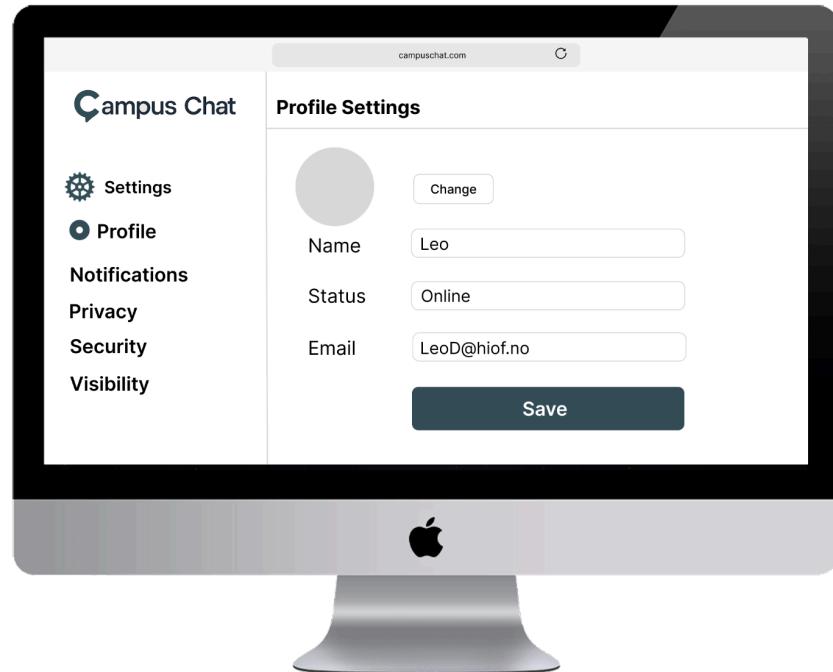
Skisse-ID: M2.1 (Profil mob)



Skisse-ID: T2 (Profil/innst Tablet)



## Skisse-ID: D2 (Profil/innst Desk)

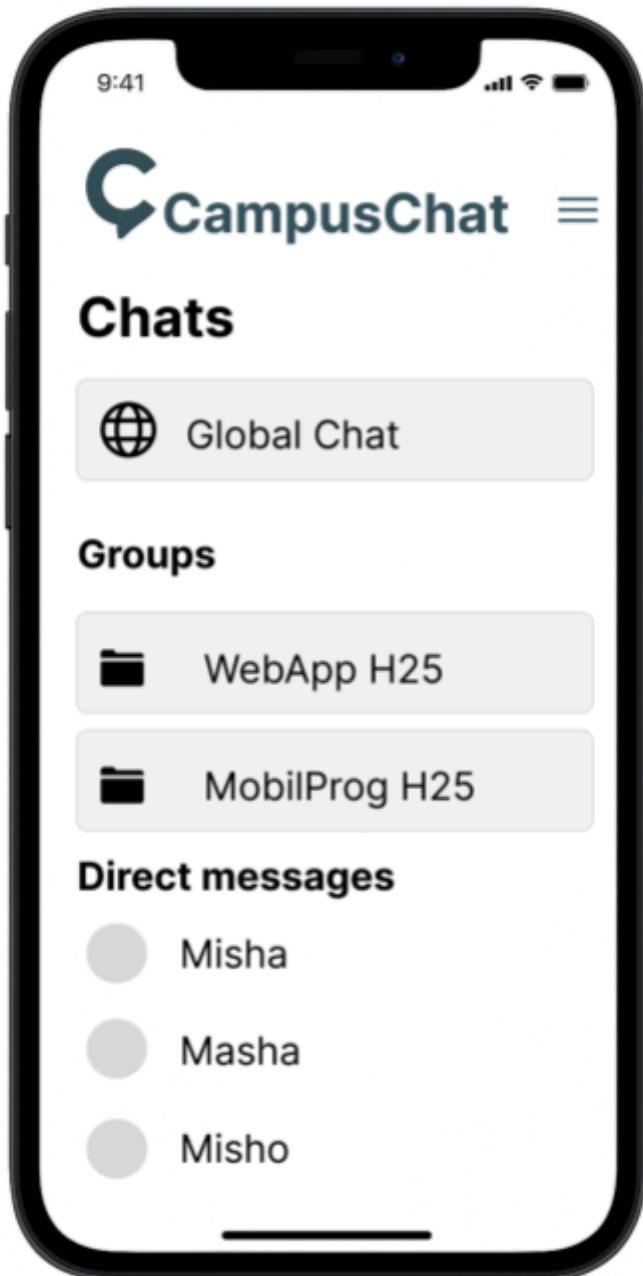


### 5.3 Chat og samtaler

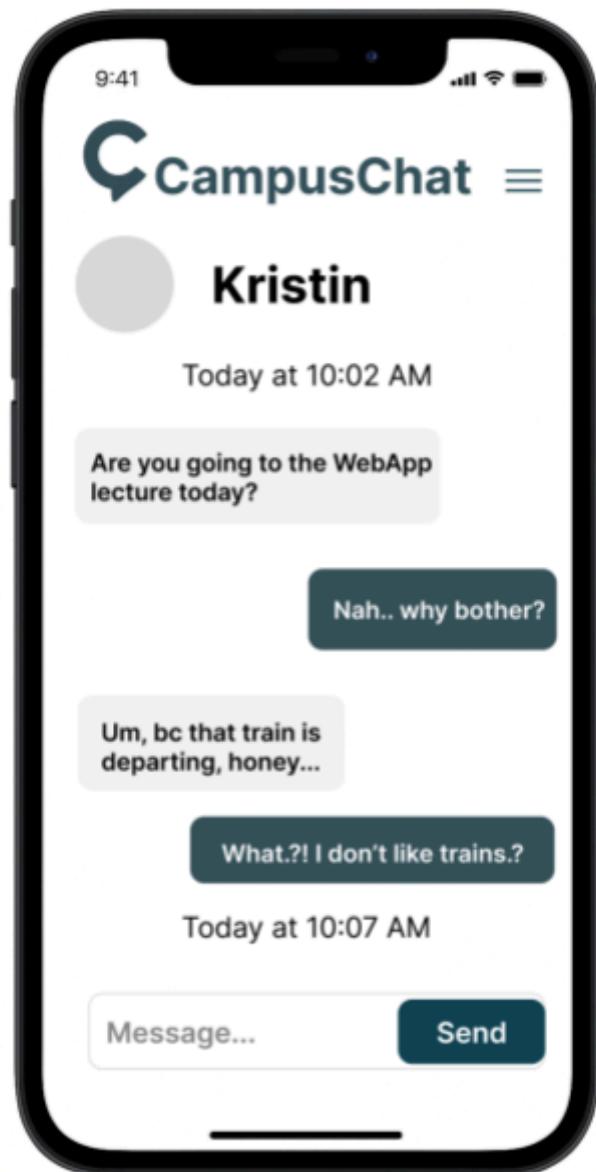
Dette er hoveddelen i CampusChat. Altså hvor all kommunikasjon foregår. Brukeren kan velge mellom global chat, egne grupper eller private meldinger. Selve samtalevisningen viser meldinger i sanntid, med avsender og tidsstempler.

På mobil vises enten kontaktliste eller samtale, mens nettbrett og desktop gir en mer oversiktlig todelt layout med liste til venstre og aktiv chat til høyre.

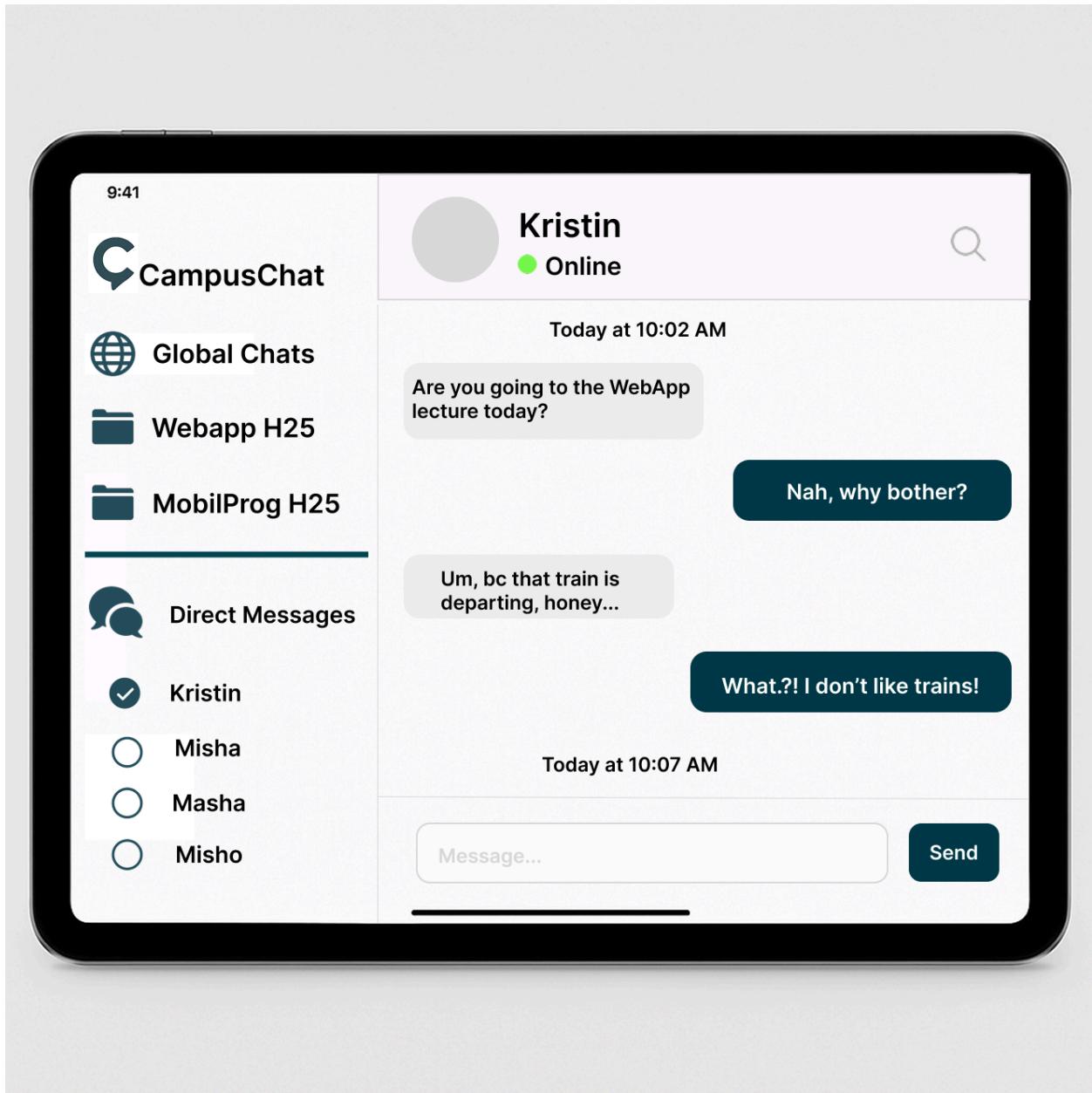
Skisse-ID: M3 (ChatOverview mob)



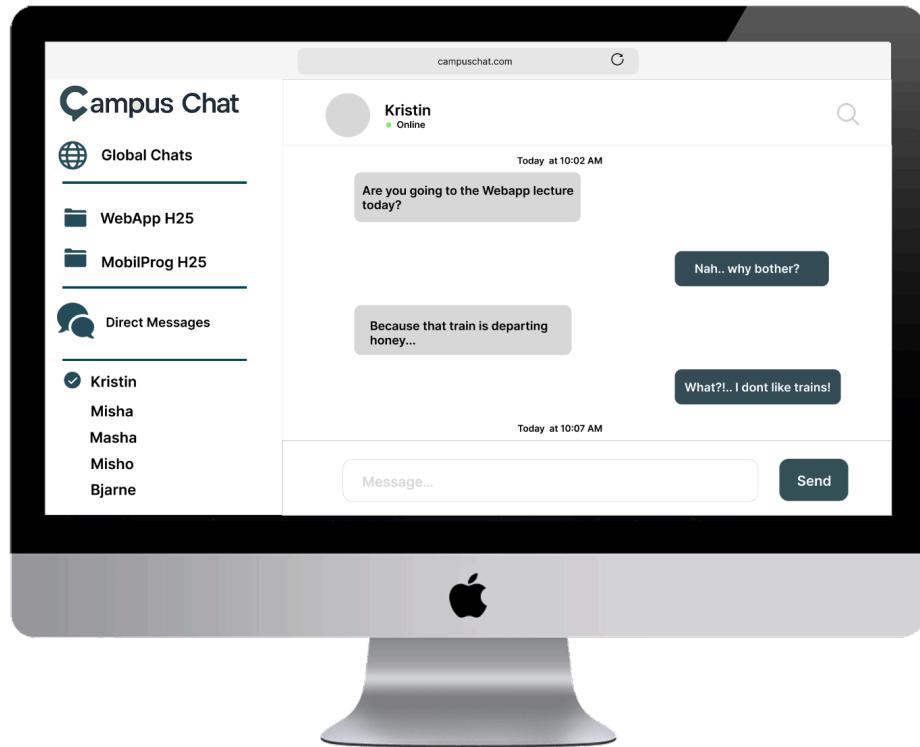
Skisse-ID: M3P (PrivatChat Mob)



Skisse-ID: T3 (Chats/privat tablet)

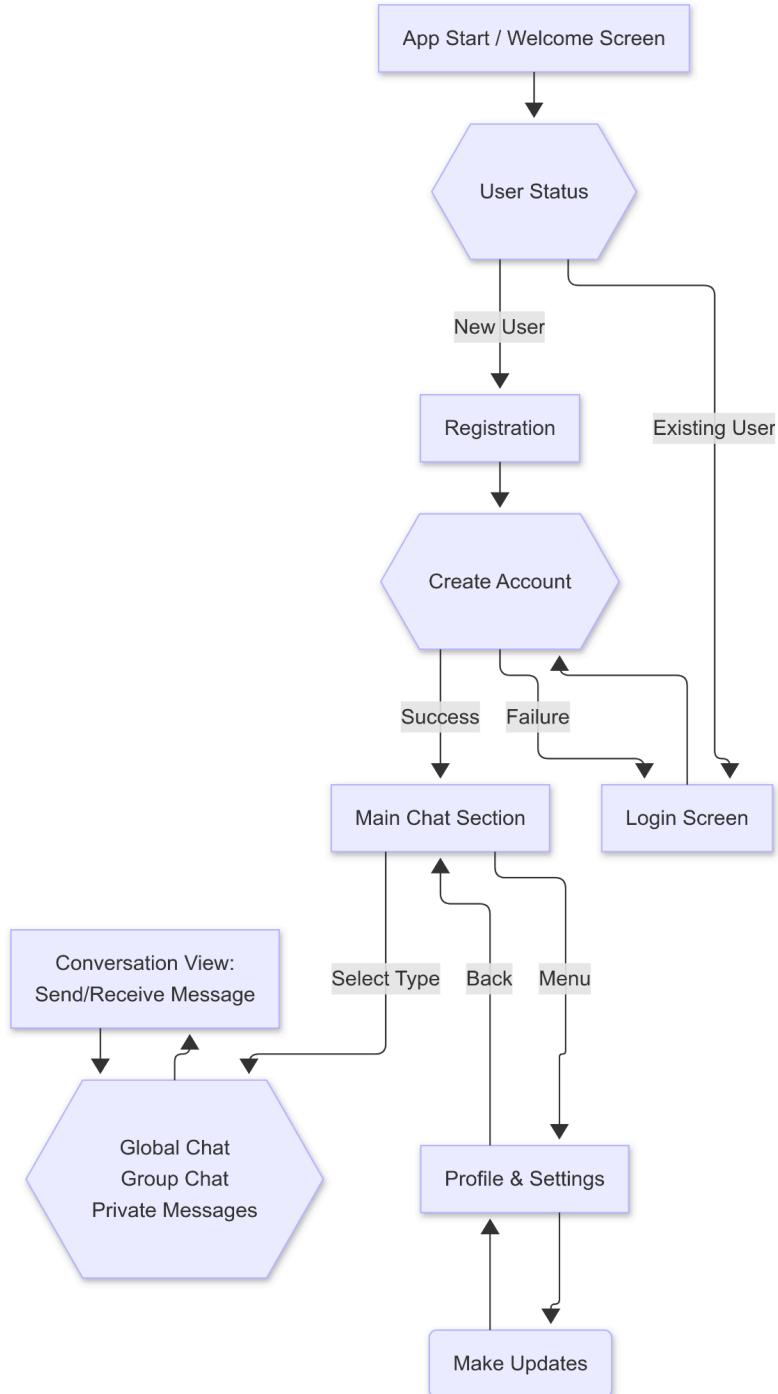


## Skisse-ID: D3 (Chats/privat desk)



# 6. Diagram

## 6.1 Overordnet dataflow-diagram for applikasjonen.



## 7. Samarbeid

Vi planlegger å møtes ukentlig for å gå gjennom status, diskutere fremdriften og bli enige om neste steg i prosjektet. I tillegg tar vi korte, uformelle møter ved behov dersom det dukker opp spørsmål eller trøbbel. På den måten vil vi sikre god og jevn fremdrift gjennom hele semesteret.

Arbeidet organiseres inspirert av smidig utvikling, der vi jobber iterativt og tester underveis i stedet for å planlegge alt i detalj fra starten av. Vi har valgt å bruke en enkel form for Kanban-board, der vi kan flytte oppgaver mellom kolonner. Dette vil gi oss en visuell oversikt og gjør det lettere å se hva hver av oss jobber med, uten at vi trenger å planlegge for langt fram i tid. Muligens at vi diskuterer oss gjennom hvem som gjør hva, dette må vi se nærmere på.

Vi har foreløpig, som nevnt, ikke delt noen konkrete oppgaver enda. Vi tar det litt som det kommer og tilpasser etter behov. Hovedideen er å fordele arbeidet litt naturlig etter kompetanse eller interesseområder, samtidig som vi samarbeider tett hele gjengen. Det er viktig for oss at alle har oversikt og alle vet “alt” og får god innsikt i de ulike delene av prosjektet. På denne måten blir vi fleksible og kan justere lett underveis, samtidig som vi hele tiden holder fokus på å levere et skikkelig produkt!

## 8.0 Prototype Alpha

### 8.1 Database

Vi har hatt en god del iterasjoner når det gjelder planleggingen av databasen. Dette var en av de viktigste delene for oss å få til, fordi alt annet avhenger av en god modell. Modellen vi har kommet med for første prototype er:

- Kravene sier globalchat, grupper, private meldinger, profiler, roller og varslinger. Datamodellen vår dekker dette uten noe form for ekstra kompleksitet. Globalchat er bare et “public” ROOM; private samtaler er egne DM\_THREAD med deltakere. Varslinger lagres i NOTIFICATION. Filer kobles til meldinger/rom via FILE\_OBJECT.
- Roller og tilgang: app-roller (USER.role) skiller student/lærer; rom-roller (ROOM\_MEMBERSHIP.role) gir eier/mod. Dette støtter akseptansekrav om riktig tilgang pr. Bruker.
- Sanntid og historikk: meldinger samles i MESSAGE, med valgfri redigeringslogg (MESSAGE\_EDIT) og lesekvitteringer (MESSAGE\_READ) for senere utvidelser uten å bryte skjema.

Mapping til krav:

- Globalchat, grupper, private meldinger -> ROOM, ROOM\_MEMBERSHIP, MESSAGE, DM\_THREAD, DM\_THREAD\_PARTICIPANT.
- Profiler/status -> PROFILE (display name, avatar, status).
- Roller -> USER.role + ROOM\_MEMBERSHIP.role.
- Varslinger -> NOTIFICATION
- Filhåndtering -> FILE\_OBJECT koblet til melding/rom.
- Sanntid -> WebSocket kan publisere nye rader fortløpende

Inni prosjektet fordelte vi de definerte databasemodellene mellom schemas:

#### **src/server/db/**

- **index** - sentrale konfigurasjonen for Drizzle ORM-forbindelsen (db) til Cloudflare D1-databasen i appen.
- **schema** - Inneholder export fra alle andre schema
- **userSchema** - brukerautentisering (users) og profilinformasjon (profiles)
- **roomSchema** - grupperom (rooms) og sporing av medlemskap i disse (roomMemberships)
- **dmSchema** - direktemeldinger (dmThreads) mellom brukere (dmThreadParticipants)
- **notificationSchema** - verslinger som sendes til brukere (notifications)
- **fileSchema** - opplastede filer og vedlegg i systemet (fileObjects)
- **messageSchema** - meldinger (messages) og all tilknyttet funksjonelitet (messageEdits, messageReads)

## **Relasjoner i databasen**

## USER

- **1:1** En bruker har én profil (USER.id → PROFILE.user\_id)
- **1:N** En bruker kan være medlem av mange rom (USER.id → ROOM\_MEMBERSHIP.user\_id)
- **1:N** En bruker kan delta i mange DM-tråder (USER.id → DM\_THREAD\_PARTICIPANT.user\_id)
- **1:N** En bruker kan skrive mange meldinger (USER.id → MESSAGE.author\_id)
- **1:N** En bruker kan motta mange varsler (USER.id → NOTIFICATION.user\_id)
- **1:N** En bruker kan laste opp mange filer (USER.id → FILE\_OBJECT.uploader\_id)

## PROFILE

- **1:1** En profil tilhører nøyaktig én bruker (PROFILE.user\_id → USER.id)

## ROOM

- **1:N** Et rom kan ha mange medlemmer (ROOM.id → ROOM\_MEMBERSHIP.room\_id)
- **1:N** Et rom kan ha mange meldinger (ROOM.id → MESSAGE.room\_id)
- **1:N** Et rom kan ha mange filer (ROOM.id → FILE\_OBJECT.room\_id)

## ROOM\_MEMBERSHIP

- **N:1** Hvert medlemskap peker på én bruker (ROOM\_MEMBERSHIP.user\_id → USER.id)
- **N:1** Hvert medlemskap peker på ett rom (ROOM\_MEMBERSHIP.room\_id → ROOM.id)

## DM\_THREAD

- **1:N** En DM-tråd har mange deltakere (DM\_THREAD.id → DM\_THREAD\_PARTICIPANT.thread\_id)
- **1:N** En DM-tråd har mange meldinger (DM\_THREAD.id → MESSAGE.thread\_id)

## DM\_THREAD\_PARTICIPANT

- **N:1** Hver deltaker tilhører én DM-tråd (DM\_THREAD\_PARTICIPANT.thread\_id → DM\_THREAD.id)
- **N:1** Hver deltaker peker på én bruker (DM\_THREAD\_PARTICIPANT.user\_id → USER.id)

## MESSAGE

- **N:1** Hver melding har én forfatter (MESSAGE.author\_id → USER.id)
- **N:1** En melding kan tilhøre ett rom (MESSAGE.room\_id → ROOM.id)
- **N:1** En melding kan tilhøre én DM-tråd (MESSAGE.thread\_id → DM\_THREAD.id)
- **1:N** En melding kan ha mange redigeringer (MESSAGE.id → MESSAGE\_EDIT.message\_id)
- **1:N** En melding kan ha mange "lesemeldinger" (MESSAGE.id → MESSAGE\_READ.message\_id)

## **MESSAGE\_EDIT**

- **N:1** Hver redigering tilhører én melding (MESSAGE\_EDIT.message\_id → MESSAGE.id)

## **MESSAGE\_READ**

- **N:1** Hver lesekvittering tilhører én melding (MESSAGE\_READ.message\_id → MESSAGE.id)
- **N:1** Hver lesekvittering tilhører én bruker (MESSAGE\_READ.user\_id → USER.id)

## **NOTIFICATION**

- **N:1** Hvert varsel tilhører én bruker (NOTIFICATION.user\_id → USER.id)

## **FILE\_OBJECT**

- **N:1** Hver fil har én opplaster (FILE\_OBJECT.uploader\_id → USER.id)
- **N:1** En fil kan være knyttet til ett rom (FILE\_OBJECT.room\_id → ROOM.id)
- **N:1** En fil kan være knyttet til én melding (FILE\_OBJECT.message\_id → MESSAGE.id)

# 9.0 Prototype Beta

## 9.1 Arkitektur og Autentisering

Fra prototype Alpha til Beta har vi gjort flere forbedringer og justeringer for å gjøre løsningen mer robust og produksjonsklar. Arkitekturen har gjennomgått mange iterasjoner, der hovedstrukturen fra Alpha i stor grad ble beholdt, men finpusset og optimalisert for bedre skalerbarhet og enklere videreutvikling. Fokus har vært på å rydde opp i strukturen, gjøre koden mer modulær og legge til rette for ekspansjon i fremtidige versjoner.

En annen viktig endring er autentiseringen, som nå er fullført og implementert med JWT-basert innlogging via HTTP-only cookies, i stedet for lagring i LocalStorage som i Alpha. Dette gir en betydelig forbedring i sikkerhet og samsvarer bedre med moderne beste praksis. Resultatet er en mer gjennomtenkt, sikker og fleksibel løsning som er klar for videre utvikling.

## 9.2 Grupper

Fra prototype Alpha til Beta har det endret seg mye for Gropus feature. I prototype alfa var det database og schemas som ble implementert. I Beta, i et forsøk på å minimum dekke alle må-krav og akseptansekrav, har vi fokusert på å skape lagdelt arkitektur, inkludert funksjonelle frontend-komponenter. Det ble opprettet:

- dto som spesifiserer inndata for API-kall, som f.eks. RoomCreateDto (inkludert zod-skjema for validering av romnavn og synlighet).
- types.ts som inneholder bruk av dataobjekter gjennom systemet (f.eks. RoomResponse, RoomFromRepo).
- roomRepository som håndterer all direkte kommunikasjon med databasen. Inkluderer per nå metoder som createRoom, findRoomsByUserId. Videre skal utvides med flere kritiske metoder som deleterom, add member, delet member, osv
- roomService som inneholder all forretningslogikk og utfører oppgaver som transformasjon av data, og orkestrering av databaseoperasjoner via Repository-laget.
  - \*\*- Inneholder getInternalIdFromExternalId - mock funksjon som foreløpig tar den unike eksterne bruker-ID-en (fra innlogging) og returnerer den unike eksterne ID-en direkte som den interne ID-en – videre skal utvides med kobling til ferdigimplement profil/user features
- roomController som mottar data fra Route, kaller Service-lagets funksjoner, og håndterer HTTP-responser (f.eks. returnerer 201 Created ved vellykket romopprettelse)
- roomRoute: Definerer API-endepunktene (/api/v1/groups/create, /api/v1/groups,) og knytter dem til de tilsvarende Controller-funksjonene.
- RoomSidebar som er en 'use client'- komponent. Håndterer den lokale tilstanden for listen over rom (rooms[]) og tilstanden for modalen for romopprettelse. Implementerer useFetch for asynkrone API-kall (GET for å laste rom, POST for å lage rom). Håndterer tilstandslogikk for opprettelse av rom (modal, validering ved hjelp av dto/zod). Sender onSelectRoom funksjonen opp til DashboardPage.tsx for global tilstandshåndtering.

- RoomCard component som tar imot onSelect funksjonen (fra RoomSidebar) og kaller denne når brukeren klikker, for å oppdatere den globale selectedRoomId-tilstanden. Kontrollerer betinget rendering av RoomDetailsPanel basert på isSelected-propoen.
- Den enkelte romlisten-elementet. Håndterer visuell tilbakemelding (valgt/ikke valgt) og logikken for å vise/skjule detaljpanelet (accordion/toggle).
- RoomDetailsPanel component som viser romspesifikk informasjon (som roomId og mock-medlemmer). Panelet inneholder TODO-kommentarer for videre utvikling. I et komplett system vil RoomDetailsPanel sende et onDelete-kall tilbake til RoomSidebar, som da utfører det asynkrone API-kallet og oppdaterer romlisten.

Alle frontend-komponentene er plassert under DashboardPage.tsx som sikrer at selectedRoomId kan deles mellom Groups og Messages (Global Chat) for å bytte mellom chat-visningene.

### 9.3 Profil

Fra prototype Alpha til Beta har feature profile gått fra å være et statisk skjermbilde til en mer dynamisk og fungerende modul. Profilesiden henter nå reell data fra databasen via ProfileController, profileRepository og ProfileSchema. NB! Notat om profileSchema i db - akkurat nå brukes denne tabellen og refereres til i profilsiden, men i videre utvikling skal dette refaktoreres til å bruke userSchema sin profiles. Derfor vil det for denne releases eksistere noe redundans i tabeller for profil.

Når profilsiden lastes blir brukerinformasjon som navn, status og varslingsinnstillinger hentet via /api/v1/profile. Endringer sendes tilbake med patch. E-post feltet er disabled - her har vi ikke landet på hvordan vi skal gjøre det, om endring skal tillates eller ikke.

CampusChatAllroundButton og CampusChatAllroundInputField fikk noe ekstra props, altså mer fleksibilitet slik at gjenbruksmuligheter ska bli flere og enklere.

En enkel toggle-knapp for å slå av og på varslinger er også implementert.

Mesteparten av logikken for å laste opp profilbilde (avatar) er på plass, men vi har dessverre ikke denne fungerende for tilfellet. Vi trenger CloiudFlare R2 storage for å få dette på plass.

Local storage ble prøvd med hjelp av fs - noe som støttes i node, men Cloudflare wia wrangler støtter ikke dette. Dessverre ikke mulig å få denne funksjonaliteten i gang til release Beta, men dette skal til end release være fungerende.