



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

PROSJEKT NR.  
17

TILGJENGELIGHET  
Offentlig

Telefon: 22 45 32 00

# BACHELORPROSJEKT

HOVEDPROSJEKTETS TITTEL  99X Smartphone App	DATO  22/05-18
	ANTALL SIDER / BILAG  128 / 22
PROSJEKTDeltakere  Andreas Danielsen (s236310) Leif Niklas Lundberg (s305341) Sondre Haldar-Iversen (s305344) Aleksander Kløve Strengelsrud (s305350)	INTERN VEILEDER  André Brodtkorb

OPPDRAUGSGIVER  99X.no AS	KONTAKTPERSON  David Kjell Tlf: 413 82 965 Email: david.kjell99x.no
---------------------------------	---

SAMMENDRAG  Bachelorprosjekt utviklet i samarbeid med 99X.no AS for OsloMet - storbyuniversitetet.  Prosjektet omhandler en smartphone-applikasjon som skal benyttes av sluttbrukere hos kundene til oppdragsgiver i forbindelse med supporthenvendelser. Det er også utviklet en tilhørende server-løsning for smartphone-applikasjonen, med mekanismer for autentisering av brukere.  Smartphone-applikasjonen er utviklet i rammeverket Ionic. Server-løsningen er utviklet i ASP.NET MVC og Web API.
---

3 STIKKORD Cross-plattform-utvikling
ASP.NET
Sikkerhet

# Forord

Denne rapporten er utarbeidet i forbindelse med bacheloroppgave ved OsloMet – storbyuniversitetet, avdeling for informasjonsteknologi, våren 2018.

Rapporten gir et overblikk over målene med oppgaven, en presentasjon av det leverte produktet, informasjon om teknologier og miljøer, en beskrivelse av vår arbeidsmetodikk, og teknisk produktdokumentasjon. Dokumentasjon som underbygger vår arbeidsprosess og utvikling av applikasjonen ligger som vedlegg. Denne rapporten understøttes av kodeeksempler og skjermbilder.

Takk til vår oppdragsgiver 99X som har gitt oss denne spennende oppgaven, og takk til kontaktpersonen David Kjell som har tilrettelagt og vært til stor hjelp for oss under prosjektet. Takk til vår veileder Andrè Brodtkorb, som har hjulpet oss med rapporten og motivert oss til å stå på. Andrè fikk gruppen fort til å forstå at tiden går fort og at vi måtte jobbe godt fra første stund. Til slutt en takk til Line Löfgreen som har hjulpet oss med design og kommet med gode tilbakemeldinger, samt Kai Kristian Schøne og Bjarte Sætveit for hjelp med nettverksproblemer.



# 99X Smartphone App

Mobile and web application

**99X.no AS**

# Innholdsfortegnelse

<b>1 Innledning</b> .....	<b>6</b>
1.1 Prosjektgruppen .....	6
1.2 Valg av oppgave.....	8
1.3 Oppdragsgiver .....	8
1.4 Fokus for oppgave .....	9
<b>2 Produktpresentasjon</b> .....	<b>10</b>
2.1 Smartphone app.....	11
2.2 Dashboard .....	20
<b>3 Prosesdokumentasjon</b> .....	<b>28</b>
3.1 Planleggingsfase .....	28
3.2 Utviklingsfase .....	45
3.3 Evaluering av utviklingsprosessen.....	56
<b>4 Teknologi og miljø</b> .....	<b>63</b>
4.1 Front-end-teknologier .....	64
4.2 Back-end-teknologier .....	66
4.3 Utviklingsverktøy.....	68
4.4 Støttesystemer .....	69
4.5 Utviklingsmiljø.....	71
4.6 Test- og produksjonsmiljø .....	75
<b>5 Teknisk produktdokumentasjon</b> .....	<b>77</b>
5.1 Kravspesifikasjon vs. levert produkt .....	77
5.2 Struktur og kodeeksempler .....	80
5.3 Mekanismer for sikkerhet og autentisering.....	83
5.4 Smartphone app.....	90

5.5 Web API.....	103
5.6 Dashboard .....	116
<b>6 Oppsummering .....</b>	<b>125</b>
<b>7 Referanser.....</b>	<b>127</b>
<b>8 Vedlegg.....</b>	<b>129</b>
8.1 Brukerhistorier .....	129
8.2 Møteforberedelse og referat .....	131
8.3 Prototyper .....	136
8.4 Akseptansetest .....	140
8.5 Risikoer .....	147

# 1 Innledning

Arbeidet med bacheloroppgaven er utført av en gruppe på fire studenter for firmaet 99X, som jobber med drift og support av IT-systemer. En viktig del av 99X sine tjenester til kundene er kontakt i forbindelse med IT-supporthenvendelser. I dag håndteres dette manuelt via et sakssystem og e-postkorrespondanse, og det er krevende for 99X å kunne gi god kundeservice til alle sine sluttbrukere. Prosjektet omfatter utviklingen av en mobilapplikasjon, som skal brukes av 99X sine kunder og en dashboard-løsning, som skal brukes av 99X sine ansatte.

Prosjektet har til hensikt å forenkle den daglige kontakten mellom kunden og 99X sin servicedesk. Ønsket kjernefunksjonalitet er blant annet å kunne sende inn og se status på saker, utsendelse av nyheter og kritiske varsler via et dashboard, samt å forenkle og tydeliggjøre kontaktkanalene til servicedesken. Applikasjonen skal, i korte trekk, gjøre det lettere for sluttbrukere å kontakte og følge med på sine henvendelser til 99X.

## 1.1 Prosjektgruppen

Prosjektgruppen består av fire studenter som har studert informasjonsteknologi ved OsloMet, Andreas Danielsen, Sondre Haldar-Iversen, Leif Niklas Lundberg og Aleksander Kløve Strengelsrud. Gruppen har en kjernekompetanse innenfor informatikk, men med noen forskjellige spesialiseringer.



**Andreas Danielsen** fikk interesse for programmering når han var med på å lage en nettside på videregående skole. Andreas har blant annet erfaring med java, C#, PHP, HTML, JavaScript, og SQL. Ved siden av studiene har Andreas laget websider og jobbet som IKT-Support.



**Sondre Haldar-Iversen** fikk interesse for programmering på videregående skole; da med enkle spill og nettsider. Sondre har blant annet erfaring med Java, C#, PHP, HTML, JavaScript, og SQL. Han foretrekker “backend”-programmering fremfor “frontend”, altså funksjonalitet fremfor GUI.



**Leif Niklas Lundberg (Niklas)** lærte først å programmere i C++ på videregående skole og har siden det jobbet med en rekke prosjekter i forskjellige språk og systemer, blant annet programmering av spill. Han jobber som konsulent hos oppdragsgiver ved siden av studiene og hjelper ofte medstudenter med programmeringsrelaterte spørsmål.



**Aleksander Kløve Strengelsrud** har under studiene spesialisert seg på nettverk, sikkerhet, linux, prosjektstyring og systemutvikling. Han har lang erfaring med IT-support og drift av systemer gjennom IT-deltidsjobber og fagbrev i IKT. Aleksander har også erfaring med Java, C#, PHP, HTML, JavaScript og SQL. Ved siden av studiene jobber han som IT-konsulent for oppdragsgiver.

## 1.2 Valg av oppgave

Den første oppgaven for gruppen var å finne en oppdragsgiver. Vi spurte flere bedrifter som gruppemedlemmene enten hadde jobbet for eller kjente kontaktpersoner fra. Bedriftene vi kontaktet var Sopra Steria, Olympus, Capgemini, Cowi og 99X. Den bedriften vi fikk best respons fra var 99X. Både Aleksander og Niklas var ansatte der da vi spurte dem.

David inviterte gruppen til et møte hvor vi diskuterte mulige bacheloroppgaver som 99X kunne være interessert i. Gruppen ble presentert en rekke mulige prosjekter, som var rangert på forhånd i forhold til hvor sterkt 99X ønsket det aktuelle prosjektet gjennomført. Av de presenterte forslagene var smartphone applikasjon, etter vår vurdering, den mest interessante av prosjektene. Vi presenterte de ulike forslagene for Tor Krattebøl hos OsloMet, og vi ble enige i at smartphone-applikasjonen kunne være et godt bachelorprosjekt. I januar 2018 begynte vi arbeidet på selve bachelorprosjektet.

## 1.3 Oppdragsgiver

99X er en norsk IT-driftspartner som tilbyr IT-tjenester og IT-plattformer as-a-service. De ble etablert i 2007, og har siden den gang hjulpet virksomheter med diverse IT-tjenester tilpasset deres behov. 99X har 130 ansatte og holder til i sentrum av Oslo.

99X drifter og har ansvaret for en rekke applikasjoner, nettverk og infrastruktur som deres kunder benytter seg av. Servicedesken fungerer som førstelinje for kunder av 99X og alle problemer og hendelser registreres i et saksbehandlingssystem (Livetime). Per i dag er det få kunder som selv benytter seg av saksbehandlingssystemet for å se og registrere saker - noe som fører til at informasjon om pågående saker er lite tilgjengelig for kunden.



## 1.4 Fokus for oppgave

I vårt prosjekt har vi valgt å fokusere på et sett med temaer. Disse temaene kan kort oppsummeres som følgende:

- Prosjektstyring
- Sikkerhet
- Teknologier
- Brukervennlighet

Vårt viktigste fokusområde har vært å jobbe med og mot en god prosjektstyring. Med god prosjektstyring mener vi at vi under hele prosjektet fokuserte på å kunne levere et ferdig produkt. Vi prioriterte mindre og godt testet funksjonalitet fremfor å implementere en større mengde funksjonalitet som ikke fungerer optimalt og som potensielt kunne utgjøre en sikkerhetsrisiko. I rapporten skriver vi om prosjektstyringen i kapittel 3.

Et annet viktig fokusområde har vært sikkerhet. Sikkerhet var noe vi tidlig hadde stort fokus på under utviklingen, ettersom 99X er en kommersiell aktør med spesielle krav til sikkerhet. Nøkkelpunkter i forbindelse med sikkerhet innebærer spesielt håndtering av autentisering og autorisering, ved hjelp av intern kodegjennomgang og gjennomtenkte sikkerhetstiltak. I tillegg forsøkte vi å ha et bevisst forhold til kjente sikkerhetshull som nevnes i OWASP[1] sin topp 10-liste. Sikkerhet diskuteres i kapittel 3.3 (seksjonen Autentisering av sluttbrukere) og 5.3.

Gruppen har også fokusert på å sette seg inn i nye, og for oss ukjente, teknologier. I tillegg har vi hatt som mål å fordype oss i allerede kjente rammeverk og verktøy. Ved å tilegne oss kunnskap utover hva som har blitt undervist i løpet av studiet ønsker vi å vise vår evne til å tilegne oss ny kunnskap. Samtidig var dette nødvendig for å levere et tilfredsstillende produkt, som kunne bidra til å oppfylle kravspesifikasjonen fra 99X. Teknologier gjennomgås og evalueres i kapittel 4.

Vi ønsker som et siste poeng å trekke frem fokus på brukervennlighet i smartphone-appen. Det er essensielt at applikasjonen er enkel å bruke og forstå ettersom den skal benyttes av svært mange brukere, med varierende bakgrunn og stillinger. Brukervennlighet diskuteres i innledningen for neste seksjon (kapittel 2), kapittel 3.3 (seksjonen Cross-plattform-utvikling via Ionic), og 4.1.

## 2 Produktpresentasjon

I dette kapittelet går vi igjennom mobilapplikasjonen og dashboardet fra brukerens perspektiv. Meningen her er å forklare og illustrere i grove trekk hvordan mobilapplikasjonen og dashboardet fungerer. Se kapittel 5 for teknisk dokumentasjon av produktet.

Vi valgte å bruke engelsk språk i mobilapplikasjonen og på dashboardet fra starten ettersom det er et språk de aller fleste kundene til 99X forstår. Det er også vanlig å utvikle på engelsk, så det var naturlig for gruppen å designe grensesnittet på engelsk først. Planen var å implementere en norsk versjon også, men tiden strakk ikke til ettersom det var viktigere å få mobilapplikasjonen og dashboardet til å fungere optimalt.

For å gjøre applikasjonen brukervennlig har vi fra starten av forsøkt å holde det grafiske grensesnittet så ukomplisert som mulig. Vi har også hatt som mål å kun bruke gjenkjennbare løsninger for funksjonaliteten, slik at den oppfører seg som forventet ved input som gjør at brukere raskt blir familiære med applikasjonen. Med dette mener vi at hvis et element ser ut som en knapp skal bruker kunne forvente seg at den oppfører seg som en knapp. Som en hjelp for å oppnå godt design har vi også brukt Android Core App Quality Guidelines[2] som grunnlag og støtte.

Ettersom mobiltelefoner kommer i en rekke formater, med mange forskjellige oppløsninger, har vi også vært bevisste på kun å bruke elementer som tilpasser sin størrelse basert på skjermens bredde og høyde. En positiv bieffekt av dette var at applikasjonen også kunne benyttes på andre typer enheter som kjører Android eller iOS, som for eksempel nettbrett.

## 2.1 Smartphone app

Denne seksjonen viser bilder av brukergrensesnittet til applikasjonen, med en forklaring av tilhørende funksjonalitet.

### Innlogging

Det første en bruker møter er et påloggingsvindu med ett felt for email og ett for passord. Begge disse feltene er påkrevd, og hvis bruker ikke har fylt inn passordet eller hvis email er på feil format vil det bli vist en forklarende feilmelding under feltet. For å forenkle andre gangs pålogging lagres email-adressen slik at bruker kun må taste dette inn ved første pålogging.

Dersom brukeren har glemt passordet vil «RESET PASSWORD» sette i gang prosessen for tilbakestilling av passord.

For å forenkle inntastingen av passordet, som ofte kan være vanskelig på telefoner grunnet små on-screen-tastaturer, er det mulig å vise passordet under inntasting ved å trykke på øye-ikonet.

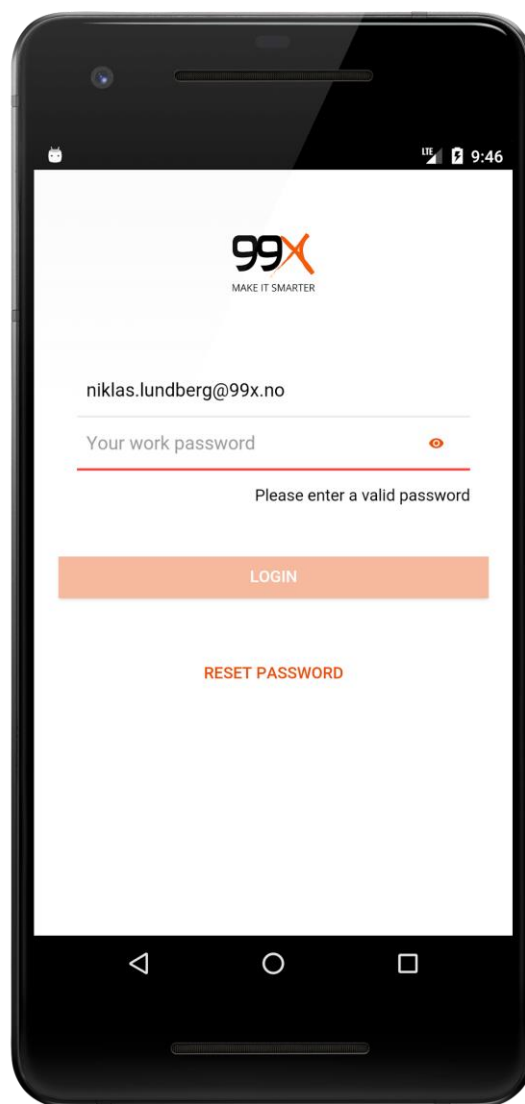


Fig. 2.1.1 - Applikasjonens innloggingsside

## Veiledning

Ved første innlogging vises en veiledning. Her får brukeren en kort gjennomgang av funksjonene i applikasjonen. Veiledningen består av fire bilder, en for hver hovedside i appen unntatt innstillingssiden.

Øverst på siden finnes en indikator for hvilken side i veiledningen brukeren befinner seg på. Man sveiper til venstre for å komme til neste bilde. På det siste bildet finnes en knapp som tar brukeren videre til hovedsiden i applikasjonen.

Denne veiledningen kan hvis ønskelig startes på nytt via innstillinger på et senere tidspunkt.

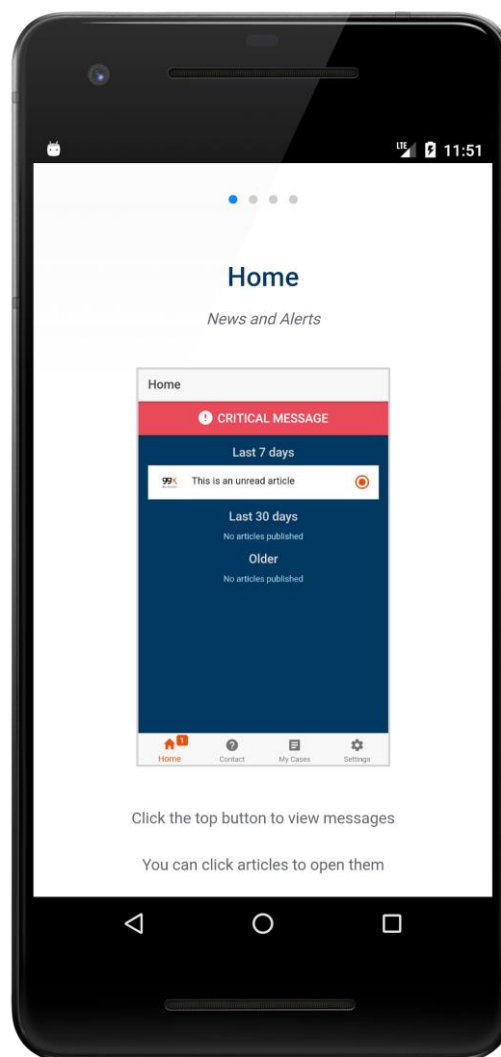


Fig. 2.1.2 - Veiledning vist ved første start

## Reset password

Når bruker klikker på reset password knappen videresendes bruker til siden der man kan tilbakestille passordet for den aktuelle bedriften man jobber i.

Her blir brukernavnet sendt med fra appen, og bruker vil få en veiledning for tilbakestilling av passordet.

Reset passord er et separat system som ikke er utviklet i forbindelse med dette prosjektet. Vår applikasjon har kun ansvar for å sende brukere til korrekt system for tilbakestilling av passord.

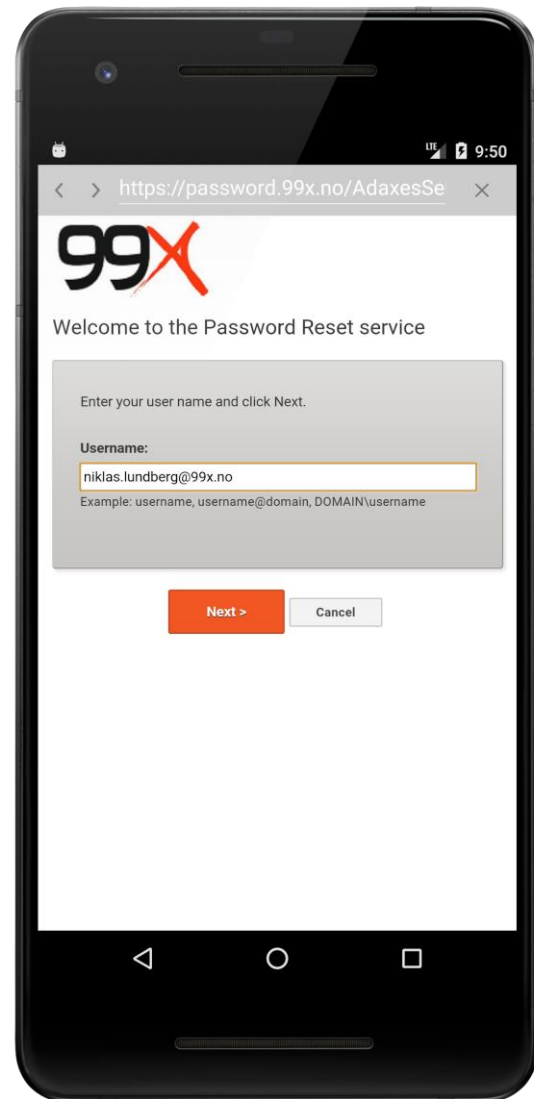


Fig. 2.1.3 - Side for tilbakestilling av passord

## Home

Her vises en oversikt over nyheter sendt ut fra 99X. Listen er sortert basert på publiseringsdatoen til artikkelen. Hvis en artikkel er ulest blir det indikert med et oransje symbol, som vist ved siden av artikkel nummer to fra toppen i fig 2.1.4.

For å lese en artikkel kan brukeren trykke på den aktuelle artikkelen i listen. Det vil da bli åpnet et nytt bilde som viser innholdet i artikkelen. Artikkelen merkes da som lest.

Under Home vil man også få opp kritiske hendelser i et banner på toppen. I dette eksemplet er det ikke noen pågående kritiske hendelser. Hvis det sendes ut en push-varsel om en kritisk hendelse vil teksten i toppen bli fremhevet.

Når bruker klikker på banneret for kritiske hendelser vil det vises en liste over alle tidligere og nåværende kritiske meldinger. Lengst ned på siden er det også en navigasjonsliste, som brukes til å bytte til andre faner/sider i applikasjonen. Denne er til stede overalt fremover i applikasjonen. Uleste artikler er også representert som et nummer i denne listen.

Hvis en bruker ønsker å hente artikler manuelt kan dette gjøres med et nedtrekk, noe som er vanlig i mange applikasjoner. Oppdatering skjer også automatisk ved hver navigering til siden.

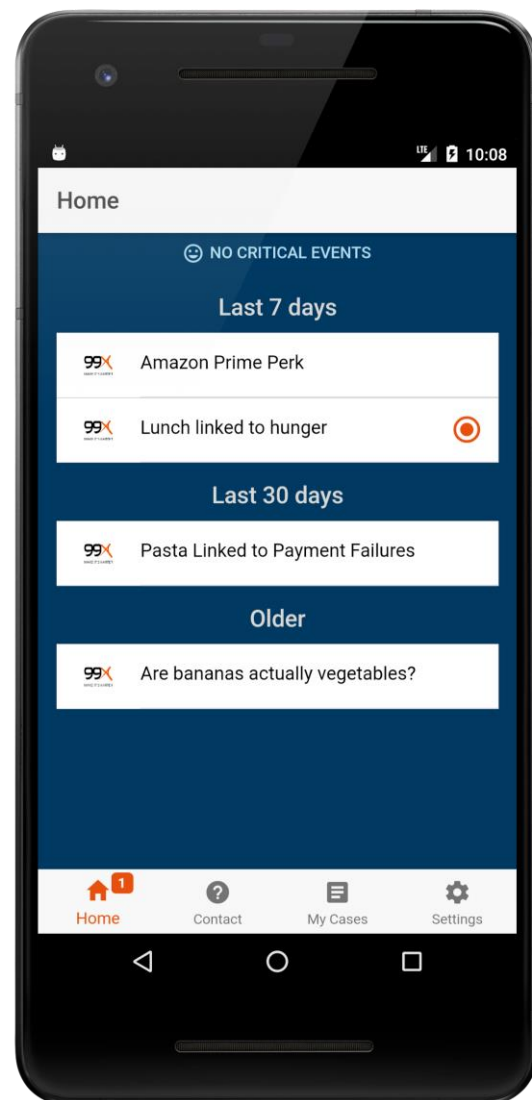


Fig. 2.1.4 - Visning av kritiske hendelser og nyhetsartikler (første bildet etter pålogging i applikasjonen)

## Contact

På contact-fanen listes noen forskjellige måter å kontakte bedriften på, samt generell firmainformasjon, veibeskrivelse og et kart.

Ved å trykk på "Send us an email" blir en dialogboks vist som informerer bruker om hvor e-mailen sendes og ber om bekreftelse. Denne dialogen er illustrert i fig. 2.1.6. Når brukeren godkjenner dialogen blir personen omdirigert til den primære mailapplikasjonen som er installert på brukerens enhet. Korrekt mottakeremail vil bli fylt inn automatisk.

Hvis brukeren ønsker å kontakte 99X over telefon finnes det også mulighet for dette. Når brukeren klikker på "Give us a call" blir det vist en dialog på samme måte som ved email. Ved godkjenning starter applikasjonen enhetens mobilapp, og fyller i nummeret til 99X.

Det er også mulig å starte en chat med en konsulent hos 99X via "Start a chat". Dette blir vist i neste seksjon (Chat).

Hvis brukeren trykker på kartet blir Google Maps-nettstedet (eller applikasjonen hvis den er installert) åpnet, sentrert på koordinatene til 99X sitt hovedkontor.

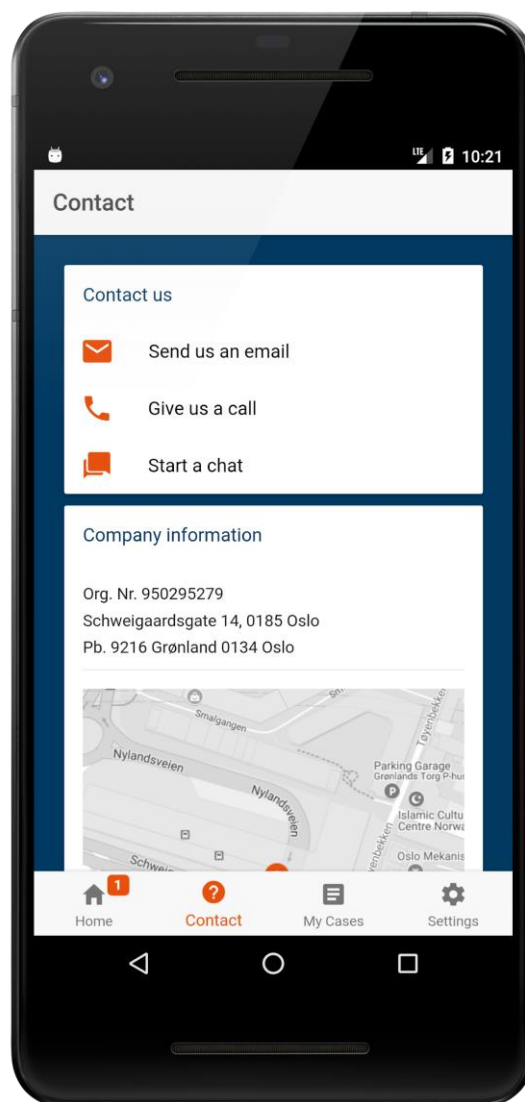


Fig. 2.1.5 - Fane for kontakthinformasjon og kommunikasjonsfunksjonalitet

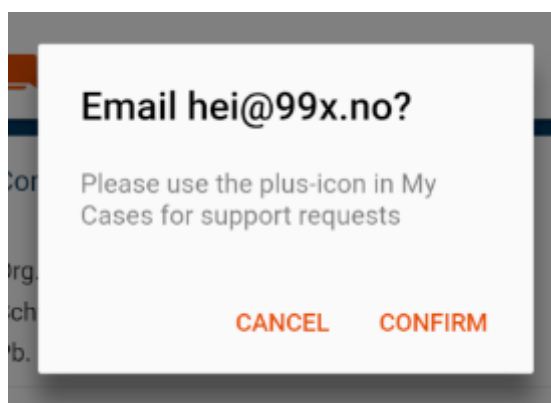


Fig. 2.1.6 - Dialog som vises ved trykk på "Send us an email"

## Chat

Hvis brukeren trykker på “Start a chat” i Contact-fanen (forrige seksjon) blir bruker omdirigert til en nettbasert chat-klient. Dette er en modifisert utgave av Puzzel Chat, som er den samme løsningen som brukes på oppdragsgiverens eget nettsted.

I det første bildet som vises bes brukeren om å fylle inn kontaktinfo. Etter dette tas brukeren videre til bildet som vist til høyre.

Fig. 2.1.7 viser en faktisk chatdialog mellom en av gruppe medlemmene og en servicedesk konsulent på 99X.

Når chatten er avsluttet kan brukeren lukke denne gjennom et trykk på krysset i høyre hjørne.



Fig. 2.1.7 - Samtale med konsulent via chat-funksjonen i applikasjonen



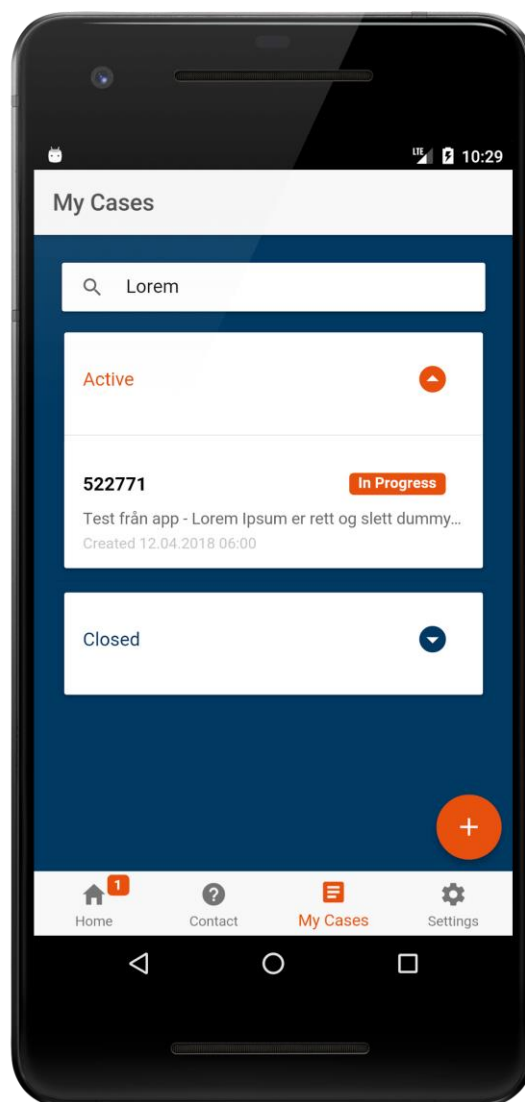
## My Cases

Det første brukeren møter i denne fanen er et søkefelt der man kan filtrere aktive og lukkede saker. For å gjøre applikasjonen responsiv gjøres filtreringen i realtid på saksnummer og sakens tittel (subject).

Saker blir listet opp med saksnummer, tittel og dato for opprettelse, slik at man enkelt og raskt kan få oversikt over aktive og lukkede saker. Hvis saken er aktiv er også en indikator for status synlig, som viser om saken for eksempel er urørt, venter på svar eller er under behandling.

Fanen bruker to lister for å separere aktive og lukkede saker. Begge ligger i nedtrekksmenyer slik at brukeren enkelt kan lukke det personen ikke ønsker å se. Hvis bruker ønsker å se detaljer om en sak åpnes en ny side ved et klikk på saken.

Bruker kan også trykke på knappen med pluss-symbolet hvis personen ønsker å sende inn en ny sak til 99X. Emailapplikasjonen som er standard på telefonen vil da bli åpnet, med korrekt mottaker og tittel fylt inn, slik at saken blir sendt inn på en korrekt måte til saksbehandlingssystemet.



*Fig. 2.1.8 - "My Cases"-fanen, med søkefunksjon og overblikk over saker fra saksbehandlingssystemet, samt knapp for opprettelse av ny sak*

## Ticket

Denne siden representerer en sak som brukeren har klikket på i listen over saker fra My Cases-fanen.

Her vises informasjon om hva saken gjelder, når den er opprettet og så videre. Informasjonen er hentet direkte fra databasen til saksbehandlingssystemet.

Lenger ned på siden finnes også en nedtrekksliste, der korrespondanse mellom brukeren og 99X listes opp. Det er også mulig å svare på disse notatene (ikke representert i bildet).

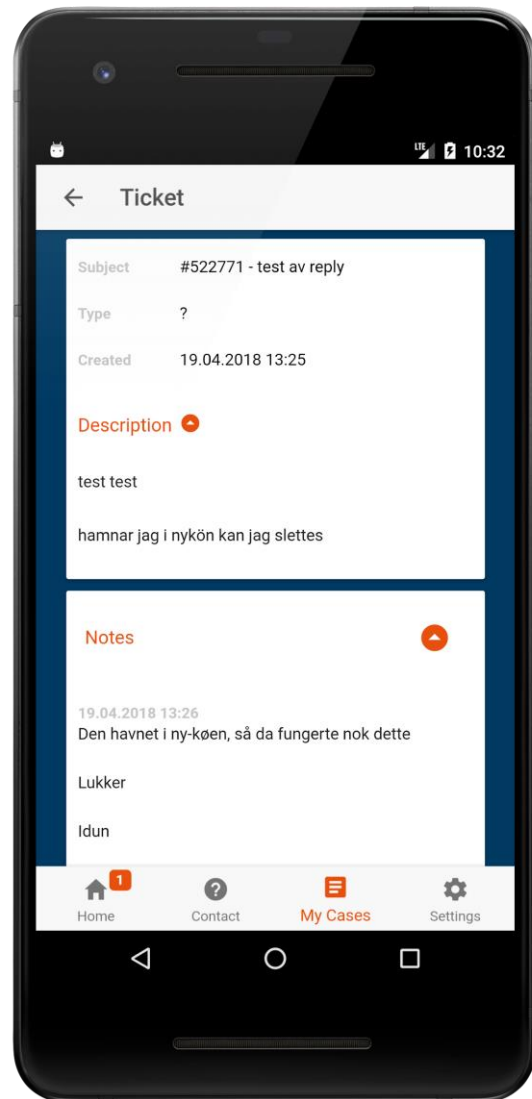


Fig. 2.1.9 - Detaljert oversikt for en sak med liste over notater (korrespondanse)

## Settings

Som illustrert øverst på fig 2.1.10 er det lagt til mulighet for å slå av og på lyd og vibrasjon. Det er også lagt til valg for språk og tekststørrelse. Denne funksjonaliteten er deaktivert, da vi ikke rakk å implementere dette. Ettersom vi har utviklet et system for å lagre og hente de personlige innstillingene i applikasjonen, besluttet vi og la denne funksjonaliteten ligge, slik at det er lett å implementere dette i en fremtidig versjon.

I settings-fanen kan brukere også logge ut, slette lagret data og tilbakestille applikasjonen. Tilbakestilling og sletting av data har vi brukt mye i testingen, men funksjonene kan også være nyttige for sluttbrukere hvis noen skulle få uforutsette problemer med applikasjonen.

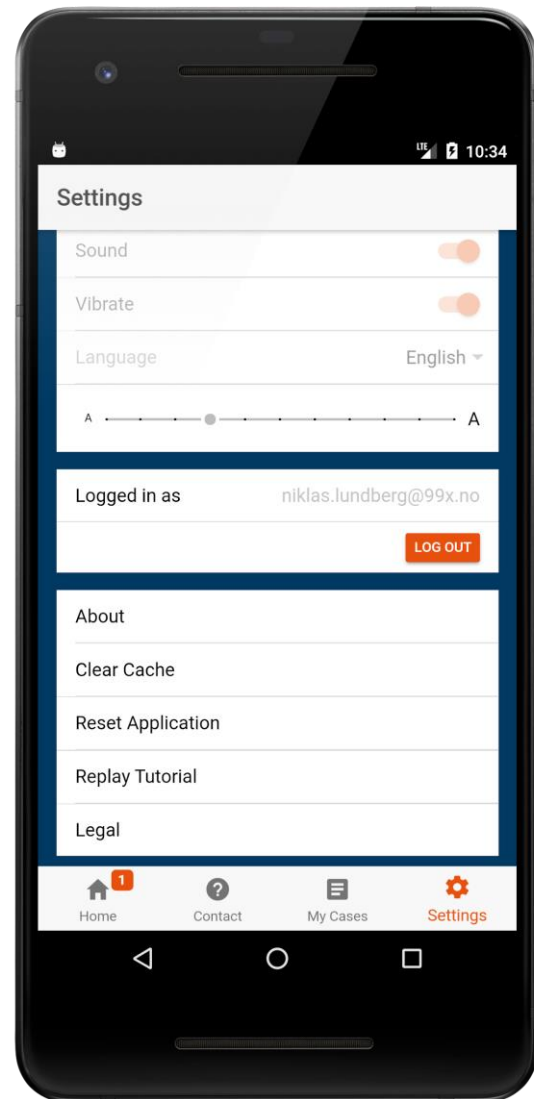


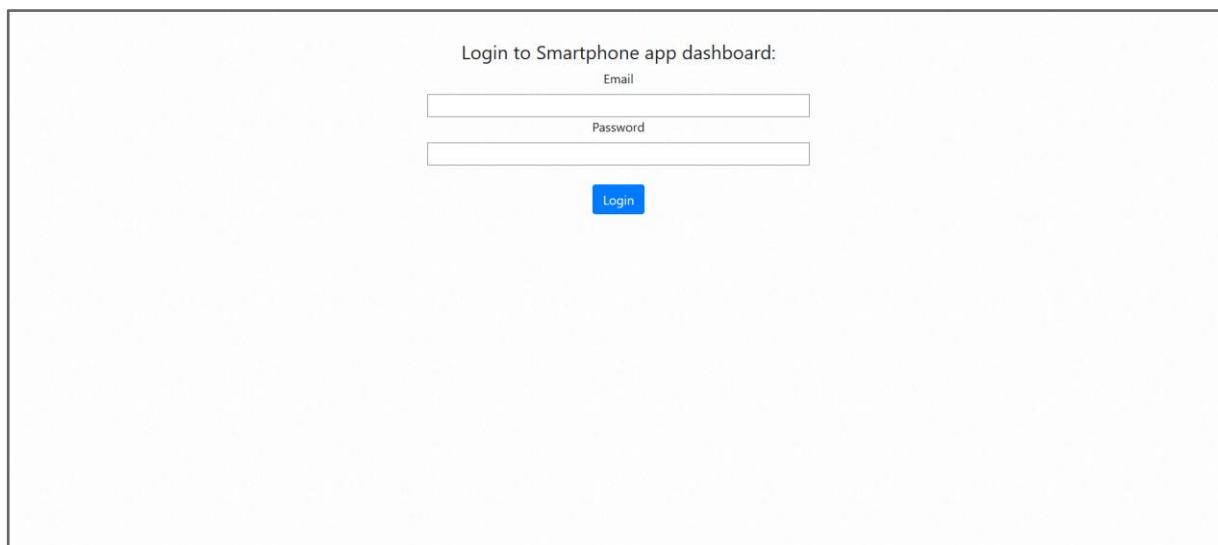
Fig. 2.1.10 - Instillingsfanen i applikasjonen

About åpner en dialog som forteller kort om utviklerne av applikasjonen. Her ligger også en kopi av lisensen til rammeverket Ionic.

Hvis en bruker ønsker å gå gjennom veiledningen på nytt kan denne også startes fra settings. Dette skjer også ved en tilbakestilling av applikasjonen.

## 2.2 Dashboard

### Login



Login to Smartphone app dashboard:

Email

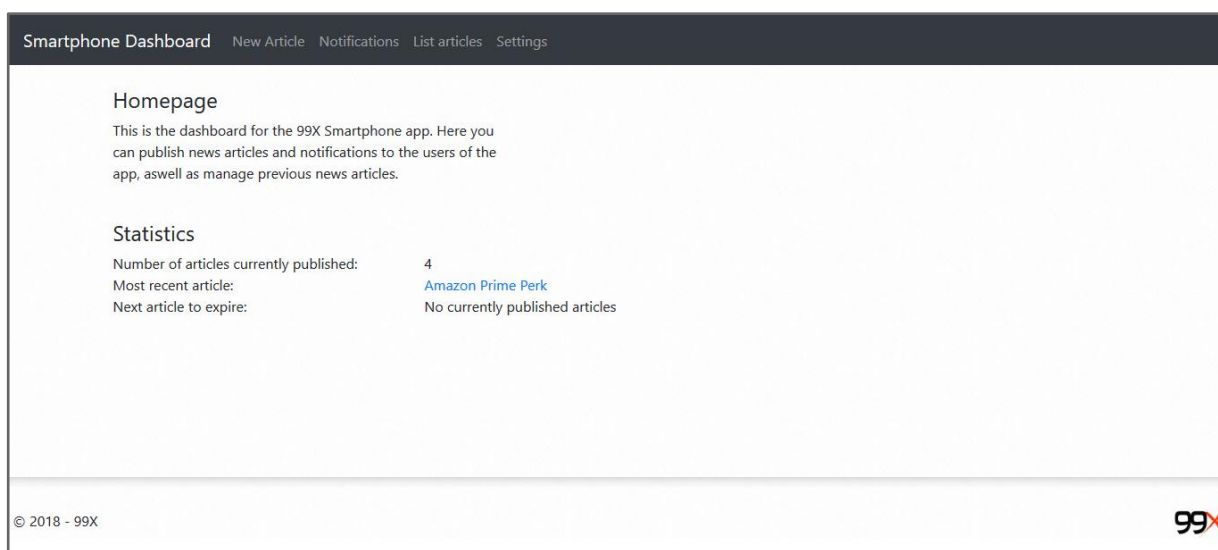
Password

Login

*Fig. 2.2.1 - Skjermbilde av login-siden på dashboardet*

Den første siden man kommer til i dashboardet er innloggingssiden. Her kreves en email og et passord til en administratorbruker. Man kommer ikke forbi denne siden uten å ha rettigheter til å kunne logge seg inn i systemet. Om brukeren taster inn feil format på email vil det bli gitt en feilmelding. I tillegg får man en feilmelding ved om man taster inn feil passord.

## Homepage



*Fig. 2.2.2 - Forsiden av dashboardet*

Når brukeren har logget inn kommer personen til forsiden av dashboardet. Forsiden er ment til å gi en oversikt over artiklene som er publisert. Siden henter nøkkeltall fra listen over artikler, som “Antall artikler som er publisert” og “Nyeste artikkel”.

På elementer som “Nyeste artikkel” vil tittelen til artikkelen vises som en link. Klikker brukeren på linken vil personen komme til redigeringsiden (se seksjonen Edit an article). Brukeren kan også navigere til forsiden av dashboardet ved å trykke på “Smartphone Dashboard” i toppmenyen.

## New article

Smartphone Dashboard New Article Notifications List articles Settings

Publish a new article:

Title

Article text

Select channel  
Could not retrieve elen

Publish date  
YYYY-MM-DD hh:mm

Expiry date  
YYYY-MM-DD hh:mm

Include image in article?  
Bla gjennom ... Ingen fil valgt. Upload

Or select an image from URL

Publish

© 2018 - 99X 99X

Fig. 2.2.3 - Side for opprettelse av ny artikkel

På artikkelsiden (New article) kan artikler lages som skal vises i mobilapplikasjonen. Den har flere felt der noen er valgfrie. Det første feltet er tittelen på artikkelen, som man er nødt til å ha med. Artikkeltekstfeltet (Article text) kan heller ikke være tomt når man publiserer en artikkel. Hvis et av de påkrevde feltene er tomme vil de markeres med en rød farge:

Publish a new article:

Title

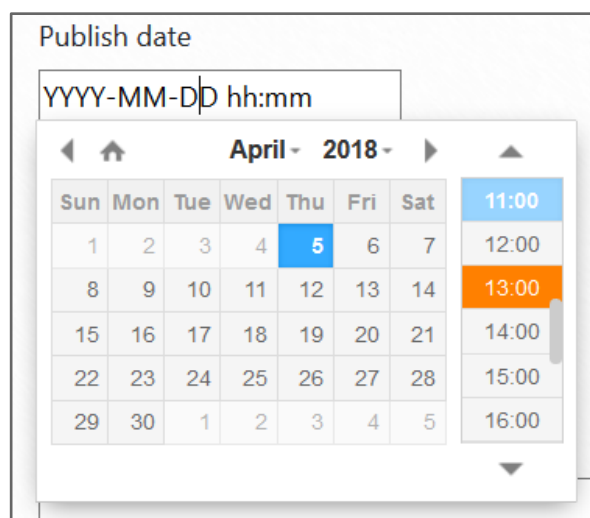
Article text

Fig. 2.2.4 - Et obligatorisk felt lyser rødt hvis det er tomt

“Select channel”-feltet er en obligatorisk nedtrekksliste hvor “kanaler” vises. Kanaler inneholder en liste med de forskjellige organisasjonene som er kunder av 99X. Det kan for eksempel være en spesifikk organisasjon, eller “all” for alle som bruker mobilapplikasjonen.

De tre første feltene er minimum av innhold man trenger for å lage en artikkel. Man kan i tillegg bestemme når artikkelen skal publiseres (publish date) og når artikkelen skal fjernes (expiry date). Når den fjernes vil artikkelen fortsatt ligge i databasen, men brukerne vil ikke lenger se artikkelen. Hvis man ikke spesifiserer når artikkelen skal publiseres vil den bli publisert med en gang artikkelen blir opprettet.

Når man trykker på dato-feltene vil en dialogboks vises. Der kan man sette dato og tid, ned til hele timer.



*Fig. 2.2.5 - Dialogboks for å sette publiseringstid på en artikkel*

Det finnes også mulighet for å inkludere et bilde i artikkelen. Man kan enten velge et bilde man har på datamaskinen, eller man kan linke til et bilde som ligger på nettet.

## Notifications

Smartphone Dashboard New Article Notifications List articles Settings

Create a new notification:

Message text

Select channel

99X

Expiry date ⓘ

YYYY-MM-DD hh:mm

Send

List of sent notifications:

ID	Message	Channel	Expiry date
1	Welcome to all new users of the 99x Smartphone app!	all	---

© 2018 - 99X

99X

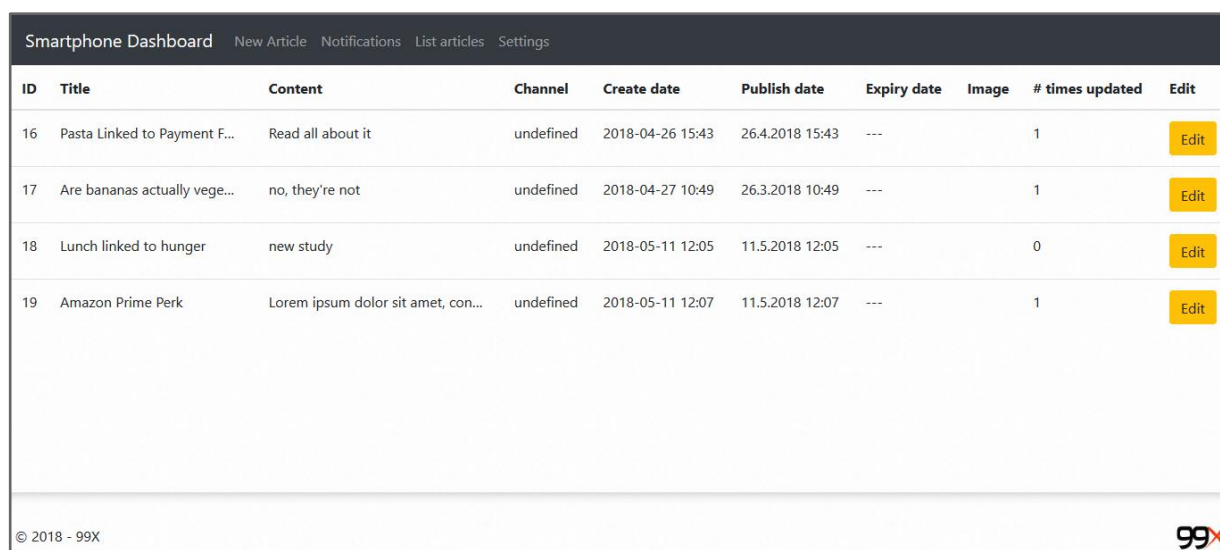
Fig. 2.2.6 - Notifikasjonsside for utsendelse av og oversikt over meldinger (push-notifikasjoner)

Notifikasjonssiden ligner på artikkelsiden, men er noe forenklet og sender notifikasjoner i stedet for artikler til brukerne. Forskjellen på en notifikasjon og en artikkel er blant annet at en notifikasjon vil “vekke” mobilen selv hvis den ikke er i bruk. En notifikasjon vil komme øverst på artikkelsiden i mobilapplikasjonen, og vil ha et advarselstegn (se 2.1 - Home).

Som på artikkelsiden er tittelen og meldingsteksten påkrevd, mens dato for når meldingen skal fjernes er valgfri.



## List articles



ID	Title	Content	Channel	Create date	Publish date	Expiry date	Image	# times updated	Edit
16	Pasta Linked to Payment F...	Read all about it	undefined	2018-04-26 15:43	26.4.2018 15:43	---		1	Edit
17	Are bananas actually vege...	no, they're not	undefined	2018-04-27 10:49	26.3.2018 10:49	---		1	Edit
18	Lunch linked to hunger	new study	undefined	2018-05-11 12:05	11.5.2018 12:05	---		0	Edit
19	Amazon Prime Perk	Lorem ipsum dolor sit amet, con...	undefined	2018-05-11 12:07	11.5.2018 12:07	---		1	Edit

© 2018 - 99X 99X

*Fig. 2.2.7 - Listing av opprettede artikler med snarveier for editering*

På siden for listing av artikler (List Articles) vil alle artiklene som er registrert være listet, også de som har gått ut. Listen består av kolonnene ID, Title, Content, Create date, Publish date, Expiry date, Image, Number of times updated, og en knapp til å redigere (Edit).

ID er et unikt tall som økes med en for hver nye artikkel som opprettes. Bilde-kolonnen (Image) vil vise et miniatyrbilde av det bildet som eventuelt vises i artikkelen. Man kan se bildet i full størrelse hvis man går til redigeringssiden (se seksjonen Edit an article). “# times updated” vil økes med en for hver gang en artikkel oppdateres. Trykker man på redigerknappen (Edit) vil man bli sendt til en side hvor man kan se og redigere en artikkel.

## Edit an article

Smartphone Dashboard New Article Notifications List articles Settings

Edit an article:

Title  
Lunch linked to hunger

Article text  
new study

Select channel  
▼

Publish date  
2018-05-11 12:05

Expiry date

Include image in article? (optional)  
Bla gjennom ... Ingen fil valgt. Upload

Or select an image from URL

Update

Delete

© 2018 - 99X 99X

Fig. 2.2.8 - Redigeringsiden for en artikkel

På redigeringsiden til en artikkel (Edit an article) kan man se en artikkel, redigere den og slette den.

Hvis en artikkel blir endret på denne måten vil en teller for antallet ganger en artikkel er blitt endret økes. Når enheter som kjører smartphone-applikasjonen ser etter nye artikler vil også de artiklene som allerede er hentet bli oppdatert hvis telleren ikke samsvarer. På den måten er det altså mulig for 99X at korrigere og endre artikler de allerede har sendt ut.

## Settings

Smartphone Dashboard New Article Notifications List articles Settings

### Settings

Logged in as **admin**

Logout

Create a new channel:

Channel name

Email domains ⓘ

ADFS Endpoint URL

Create

Channel list:

Name	Email domains	ADFS Endpoint URL	Remove
99x	99x.no	http://eksempel.no/adfs/endpoint	Remove channel

© 2018 - 99X 99X

Fig. 2.2.9 - Innstillingsside med funksjonalitet for administrering av kanaler

På innstillingssiden (Settings) kan man logge seg ut av dashboardet og administrere kanaler (se 2.2 - New article). Øverst på siden vil man se brukeren man er logget inn med. For feltene relatert til opprettelse av en ny kanal er “Channel name”, “Email domains”, og “ADFS Endpoint URL” påkrevd. Når pekeren holdes over info-ikonet (i) vil en forklarende tekst vises:

Create a new channel

Channel name

Email domains ⓘ

If a channel has several email domains, separate them with a comma without space (,)

Fig. 2.2.10 - Info-vindu vises når musepeker holdes over info-ikonet

Under feltet for registrering av nye kanaler vil de registrerte kanalene listes. Der er det mulig å slette kanaler, hvis de for eksempel ikke lenger er i bruk.

## 3 Prosessdokumentasjon

Det var viktig for oss å ha en strukturert og gjennomgående plan for prosjektet. Vi utarbeidet en målbeskrivelse på hva som måtte gjøres for å sikre en god flyt i prosjektet. Planene og målbeskrivelsen hjalp gruppen med å holde oversikt over hele prosjektet, samt at det ga oss en pekepinn på hvilke utfordringer som kunne dukke opp underveis. Arbeidet med dette bidro til at arbeidsoppgavene for gruppen ble mer håndterbare og lettere å fokusere på. Det reduserte også risikoen for feilestimering.

For å kunne tilfredsstill de kravene 99X hadde satt, i tillegg til standarden på prosjektet, var det avgjørende at all planlegging var godt dokumentert. Planleggingen resulterte i at vi fikk en overordnet forståelse av hva som måtte gjøres og når en spesifisert oppgave var planlagt ferdigstilt. Det var også viktig for gruppen og 99X, at dokumentasjonen var grundig og oversiktlig ved overlevering av prosjektet og for eventuelle fremtidige utviklere. Vi benyttet tankegangen og de grunnleggende prinsippene bak Kanban-metodikken for å strukturere arbeidsoppgavene våre. Bruken av Kanban og hvordan vi benyttet arbeidsmetoden vil bli diskutert senere i dette kapitlet.

### 3.1 Planleggingsfase

Vi begynte planleggingen av prosjektet allerede i november 2017. Det ble da bestemt at vi skulle lage en smartphone-applikasjon som kunne motta relevante nyheter fra 99X og forenkle den daglige kontakten med deres servicedesk. Bedriften ønsket at applikasjonen skulle være tilgjengelig både via AppStore og Google Play. For å utvikle applikasjonen til både iOS og Android begynte vi tidlig å se på ulike løsninger som kunne muliggjøre å utvikle applikasjonen til de to operativsystemene simultant. Etter å ha undersøkt flere mulige rammeverk som støtter dette kom vi frem til at vi skulle utvikle applikasjonen ved å bruke rammeverket Ionic. Dette diskuteres i detalj under seksjonen Løsningsalternativer.

## Resultatmål

Målet med applikasjonen er å forenkle og effektivisere en rekke arbeidsoppgaver som i dag utføres manuelt av 99X sin servicedesk. Oppdragsgiver ønsker å flytte en del oppgaver som i dag utføres av servicedesk over til kunden. Dette gir mulighet til å effektivisere servicedesken sin arbeidsprosess. Samtidig ønsker man å gjøre det enklere for brukerne å følge med på sine pågående saker.

Et annet mål for prosjektet er å redusere kostnader til oppdragsgiver. I dag bruker bedriften store midler på å sende ut SMS til brukere som rammes av kritiske systemfeil. Ved bruk av push-varslere kan disse meldingene sendes ut kostnadsfritt via mobilapplikasjonen.

I dag tilegner kunder seg informasjon om pågående saker gjennom å enten ringe inn eller følge med på mail for oppdateringer. I tillegg til de overnevnte utfordringene kommer det også en rekke henvendelser som gjelder reset av passord, bestilling av utstyr og tilganger, spørsmål ved kritiske hendelser samt andre generelle support spørsmål.

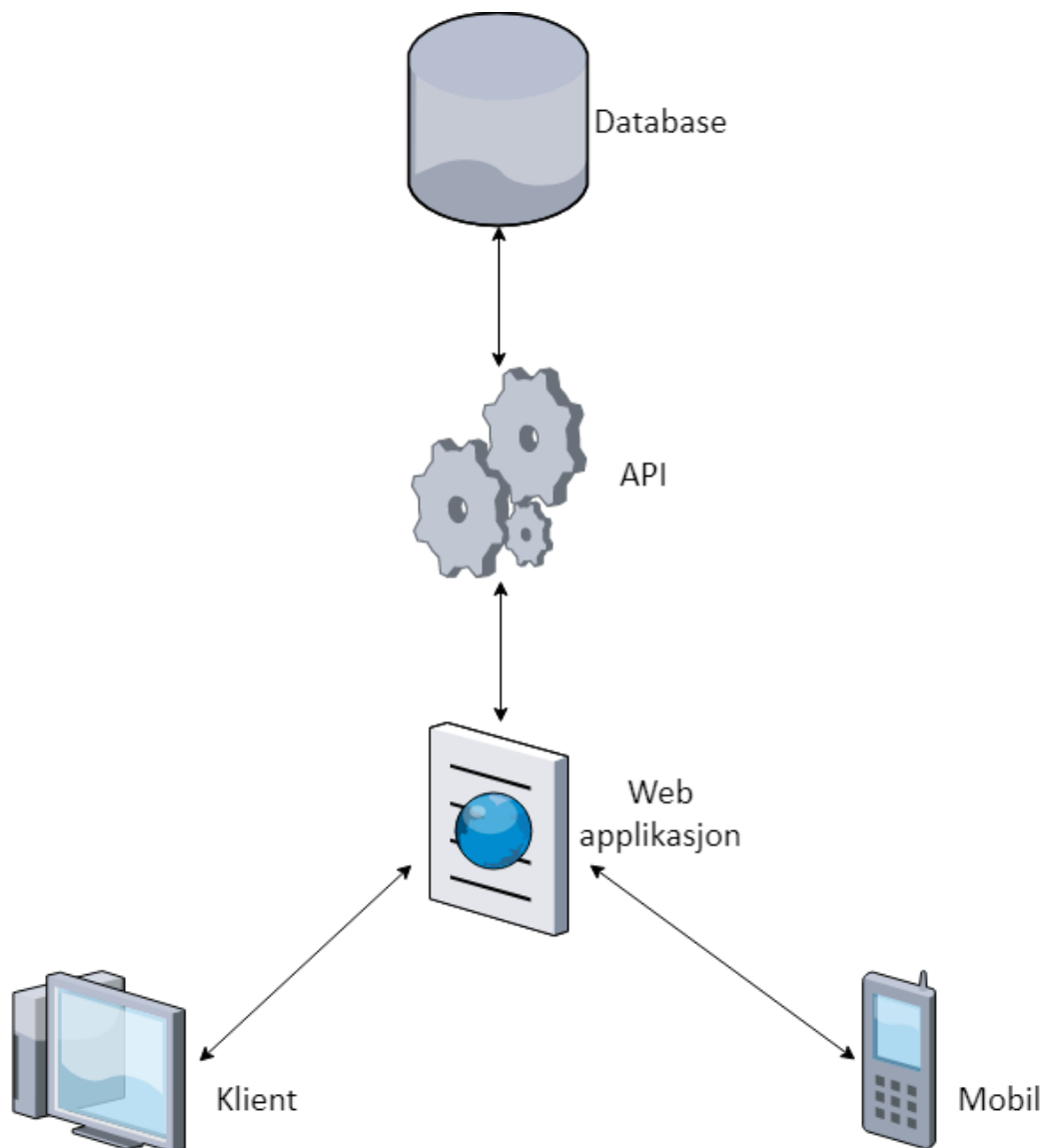
Servicedesken bruker mye tid og ressurser på oppgaver kundene hadde kunnet utføre på egenhånd hvis de rette verktøyene hadde vært tilgjengelige. En applikasjon vil kunne bidra til å samle denne funksjonaliteten på ett sted, slik at man kan henvise kundene direkte til applikasjonen. 99X ønsker at applikasjonen skal bli en integrert del av kundens bruksmønster og være en hovedinformasjonskanal for dem.

## Ansvarsområder

Ettersom prosjektet består av en rekke frittstående arbeidsoppgaver og systemer var det naturlig for oss å spesifisere dedikerte ansvarsområder for gruppemedlemmene. Sondre fikk ansvaret for utvikling av dashboardet og Niklas fikk hovedansvaret for utviklingen av smartphone-applikasjonen. Andreas og Aleksander fikk hovedansvar for oppgaver relatert til prosjektstyring, dokumentasjon og modellering av diagrammer.

Mot slutten av prosjektet var det nødvendig at alle medlemmene skrev på rapporten, i tillegg til å gjøre ferdig produktene, konfigurere serveroppsett, og se på publisering til butikker.

## Systemarkitektur



*Fig. 3.1.1 - Grunnleggende oversikt av den foreslåtte systemarkitekturen*

Skissen over viser vårt produkts overordnede systemarkitektur. Vi utarbeidet denne skissen tidlig i forbindelse med planleggingsprosessen for å illustrere, overfor oppdragsgiver, hvordan vi ønsket at systemet skulle se ut i grove trekk. Både mobil- og dashboard-brukere henter og lagrer all data via et Web API (forklart i neste seksjon). I bakkant av APIet bruker vi tradisjonelle databaser for datalagring.

Alle komponenter i skissen vil bli forklart og redegjort for i detalj i løpet av dette dokumentet.

## Hva er et Web API?

Da vi planla prosjektet kom vi fram til at vi måtte implementere et system for å tilgjengeliggjøre data og ressurser for klienter av systemet på en standardisert måte. Dette løste vi gjennom å konstruere et såkalt Web API. Dette er et type grensesnitt der intensjonen er å abstrahere måten klienter snakker med tjenester på.

I stedet for at klientene kommuniserer direkte med tjenester (for eksempel mot en database) kontakter de APIet, som knytter sammen de forskjellige ressursene og tilgjengeliggjør disse for klientene på en simpel måte. Dette skjer gjerne via et kall til en webadresse. Konseptet Web API forklares i større detalj i innledningen til kapittel 5.5.

Diagrammet nedenfor viser vår initielle plan for strukturen til APIet, og gjenspeiler også systemets oppsett i dag. Bildet illustrerer konseptet at Web APIet agerer som et bindeledd mellom klienter og forskjellige tjenester og systemer.

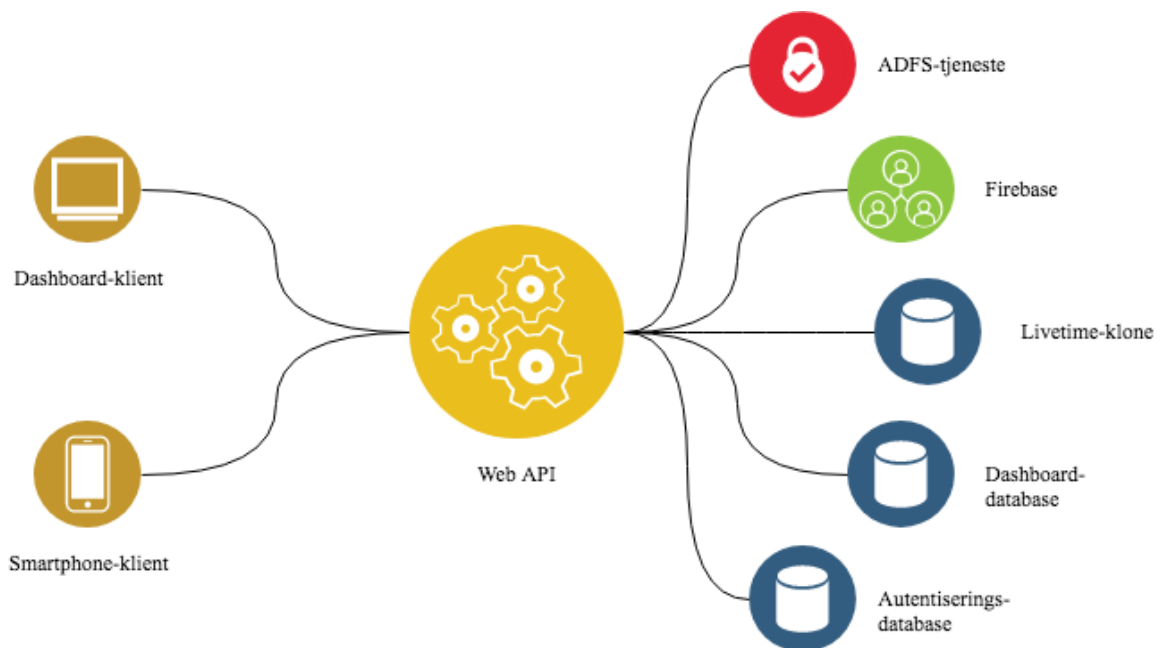


Fig. 3.1.2 - Visuell representasjon av strukturen til prosjektets Web API

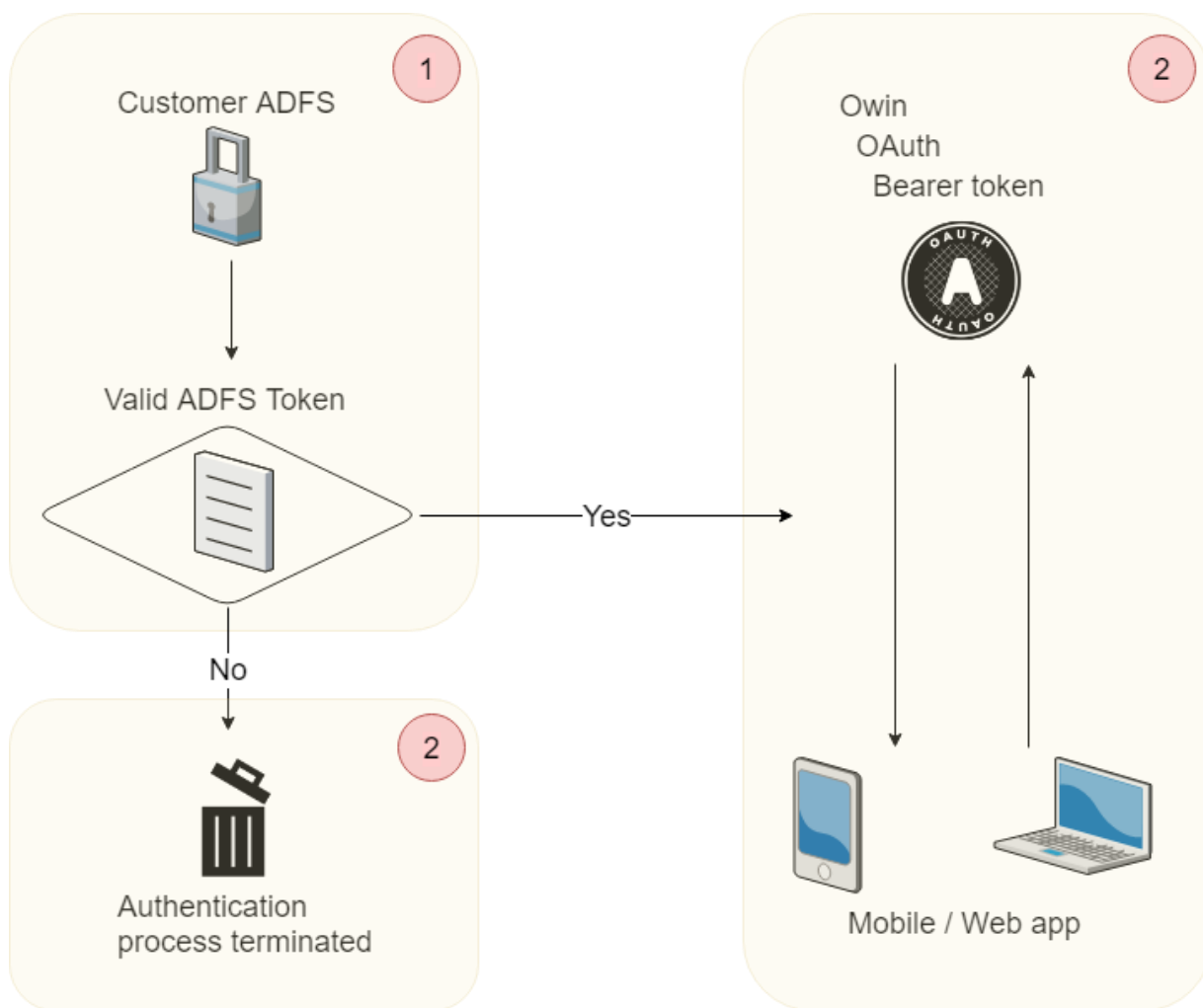


Fig. 3.1.3 - Skisse av den planlagte løsningen for autentisering av brukere

Ettersom sikkerhet er et sentralt (og komplisert) konsept i vårt prosjekt var det viktig for oss å tidlig definere hvordan strukturene for autentisering og sikkerhet skulle implementeres. Vår intensjon var å praktisere Security by Design<sup>1</sup> og dermed sørge for å ha en veldefinert arkitektur for sikkerhet fra starten av, fremfor å bygge inn sikkerheten rundt eksisterende komponenter. Figuren 3.1.3 er resultatet av denne planleggingen, og oppsettet har stort sett vært uforandret siden starten av prosjektet.

For å sette oss inn i autentiseringsmekanismer for ASP.NET (systemet vi brukte for vår serverløsning) har vi lest den offisielle dokumentasjonen fra Microsoft[3]. Vi har også brukt veiledningsserien om sikkerhet i ASP.NET fra Taiseer Joudeh[4], og tilpasset etter våre behov.

<sup>1</sup> Security by Design er en spesifisering fra OWASP[1] som definerer et sett med anbefalinger for å oppnå sikre systemer.



## Prosjektstart

Etter flere møter med 99X har vi kommet frem til såkalte “must-haves” og “nice-to-haves” som definerer funksjonalitet det er ønskelig at mobilapplikasjonen skal ha. “Must-haves” er definert som krav til produktet, mens “nice-to-haves” er de komponentene som ikke er kritiske men ønskelig å ha med.

Referat fra disse møtene kan leses i vedlegg 8.2. Samtalene med 99X har vært konstruktive og vi fikk inntrykk av at de ønsker denne applikasjonen. 99X bekreftet sin interesse gjennom å sette av mye tid og ressurser til vår disposisjon i forbindelse med utviklingen av applikasjonen og dashboardet.

### **Must-haves**

- Push notifications (planlagt vedlikehold og andre hendelser/incidents)
- Real-time-chat (integrasjon med eksisterende system bygget i jQuery/html)
- Call Servicedesk-funksjonalitet (basert på kunde)
- Sende mail/nye saker til Servicedesk
- Rapporter for IT-brukere hos kunder
- Bestille nye passord og låse opp kontoer (ActiveDirectory)
- Statusoversikt for aktive saker (evt. historikk)
- Dokument-uploads (veiledninger, etc.)
- Logging (mail til Livetime - eksisterende saksbehandlingssystem)
- Administrasjonsgrensesnitt (håndtering av informasjon i appen)

### **Nice-to-haves**

- Div. skjemaer for bestilling (brukere, utstyr, tilganger, deaktivering, etc.)
- Mulighet til å endre tekststørrelse i mobilapplikasjonen
- Mulighet til å endre språk i mobilapplikasjonen

## Funksjonelle / ikke-funksjonelle krav

Must haves og Nice-to-haves ble notert ned under planleggingen av prosjektet som en uformell punktliste med krav til tjenesten. Vi kom frem til at vi skulle lage en dashboard-løsning hvor man kan sende nyheter og notifikasjoner til appen. Dashboardet skulle være en nettside som ansatte hos 99X kan logge seg inn i.

Etter møtene valgte vi å systematisere og formatere kravene i en liste med funksjonelle og ikke-funksjonelle krav. Ikke-funksjonelle krav tar for seg alt som ikke har med funksjonene til tjenesten å gjøre, som for eksempel overordnede krav til selve produktet. I tillegg definerte vi ikke-funksjonelle krav ut i fra type, som for eksempel om det var et produktkrav eller organisatorisk krav. Funksjonelle krav spesifiserer krav satt til selve funksjonaliteten til tjenesten. Ved å gå gjennom kravene og definere de sørget vi for at alle på gruppen forstod hva kravene innebar, samt at alle kravene ble generalisert og standardisert. På den måten fikk vi en god og formell oversikt som vi kunne forholde oss til, og ha som utgangspunkt når vi skulle definere en kravspesifikasjon.

#	Funksjonelle krav
1	ADFS-innlogging for å autentisere bruker i applikasjonen og dashboardet
2	Push-notifikasjon for kritiske hendelser eller vedlikehold
3	Nyhetsstrøm med aktuelle nyheter per kunde
4	Registrere saker i sakshåndteringssystemet
5	Se aktive / tidligere saker i sakshåndteringssystemet
6	Ring til servicedesk-funksjonalitet
7	Integrere allerede operativ chat-funksjonalitet
8	Resette passord til brukerens konto
9	Dashboard for administrering av nyheter og hendelser
10	Generere og se rapport over bruk av applikasjonen
11	Gjøre tilgjengelig veiledningsdokumenter og bestillingsskjemaer

#	Ikke-funksjonelle krav	Kravtype
1	Intuitiv og brukervennlig fremstilling av informasjon	Produktkrav
2	Rask og familiær navigasjon mellom forskjellig funksjonalitet	Produktkrav
3	Grundig testet og sikret	Produktkrav
4	Flytte oppgaver over til kunden/sluttbruker	Organisatorisk
5	Systemet må være godt dokumentert.	Produktkrav
6	Applikasjonen skal spare 99X for kostnader	Organisatorisk

## Kravspesifikasjon

Kravspesifikasjonen ble definert i form av en liste med brukerhistorier som vi utarbeidet i samarbeid med oppdragsgiver. I oppstartsfasen kom vi etter flere møter frem til hvilken funksjonalitet 99X ønsket at vi skulle prioriterte under utviklingen av applikasjonen.

Listen beskriver hva en definert rolle i systemet skal kunne gjøre. Vi har videre delt opp brukerhistoriene i to deler hvor en del omhandler mobilapplikasjonen, og en del tar for seg dashboard-webapplikasjonen.

### Definerte roller:

- Mobilapplikasjon
  - Sluttbruker
- Dashboard
  - Teknisk konsulent
  - Markedsfører
  - Administrator

Vi valgte å ha med dette på grunn av at dette gjorde det enklere for arbeidsgiver og få oversikt. Det ble også lettere å få oversikt over hvilken funksjonalitet og roller forskjellige brukere skulle ha.

Figur 3.1.4 nedenfor viser en forenklet og visualisert oversikt over hva hver enkelt rolle i systemet skal ha mulighet til å gjøre. De fullverdige brukerhistoriene finnes i vedlegg 8.1.



**Sluttbruker**

Skal

- Logge på applikasjonen
- Opprette, se, respondere og filtrere saker
- Ringe og chatte med servicedesken
- Motta nyheter og notifikasjoner
- Tilbakestille passord
- Tilpasse applikasjonen



**Teknisk konsulent**

Skal

- Logge på dashbordet (internt og eksternt)
- Se, sende, oppdatere og fjerne notifikasjoner
- Begrense mottakere av notifikasjoner
- Se historikk og statistikk over notifikasjoner
- Benytte ferdige maler for notifikasjoner



**Markedsfører**

Skal

- Se, sende, oppdatere og fjerne nyheter
- Sette utløp og publiseringstid på nyheter
- Laste opp bilder og/eller hente bilde fra galleri

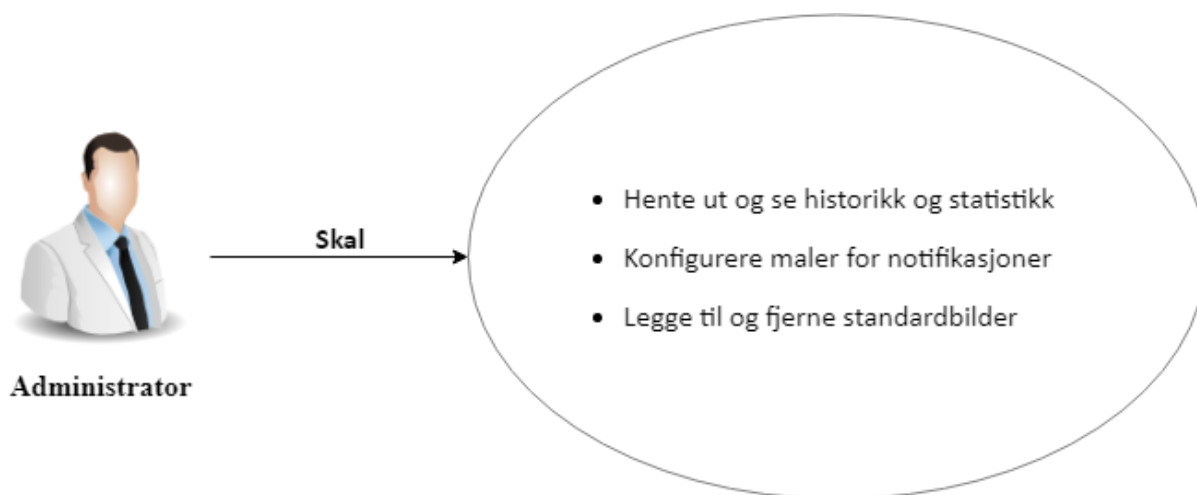


Fig. 3.1.4 - Brukerhistorier for forskjellige roller

## Løsningsalternativer

Tabellen under illustrerer de forskjellige løsningsalternativene gruppen diskuterte. Alternativene ble satt opp ovenfor hverandre for å gi en god oversikt og et grunnlag for diskusjon, slik at vi enklere kunne sammenligne fordeler og ulemper med hvert enkelt alternativ. Oversikten bidro til å reflektere over flere viktige aspekter ved utviklingsprosessen, noe som var avgjørende for hvilket alternativ vi valgte.

Løsning	Fordeler	Ulemper
<p><b>Alternativ 1:</b> Utvikle app via cross-platform teknologi (for eksempel PhoneGap, Ionic eller Cordova).</p>	<ul style="list-style-type: none"> <li>- Utviklingstid blir minimert fordi samme løsning vil fungere på alle operativsystemer.</li> <li>- Mindre og mer oversiktlig kodebase da kode deles mellom implementeringene.</li> </ul>	<ul style="list-style-type: none"> <li>- Bruk av cross-platform teknologier kan gi en noe mindre effektiv app enn en tradisjonelt utviklet app.</li> <li>- Striktere adgang til OS-funksjonalitet og komponenter (Native).</li> </ul>

	<ul style="list-style-type: none"> <li>- Koding i TypeScript, HTML og SCSS (styling-dokumenter) leder til muligheter for rask prototyping.</li> </ul>	<ul style="list-style-type: none"> <li>- Forutsetter at benyttede rammeverk blir vedlikeholdt iht. plattform-endringer</li> </ul>
<p><b>Alternativ 2:</b> Utvikle app parallelt for hvert OS appen skal publiseres til</p>	<ul style="list-style-type: none"> <li>- En tradisjonelt utviklet app kan være noe mer effektiv enn en app utviklet ved hjelp av cross-plattform teknologi.</li> </ul>	<ul style="list-style-type: none"> <li>- Utviklingstiden blir lenger fordi appen må utvikles spesifikt for hvert operativsystem.</li> </ul>
<p><b>Alternativ 3:</b> Utvikle appen for kun ett operativsystem.</p>	<ul style="list-style-type: none"> <li>- Utviklingstid blir minimert og vi trenger kun å forholde oss til én utviklingsmiljø.</li> <li>- Flere av gruppemedlemmene har allerede kompetanse innen området (Android).</li> <li>- Appen vil kun bli publisert til én butikk, som vil minske sannsynligheten for at problemer vil oppstå ved publisering.</li> </ul>	<ul style="list-style-type: none"> <li>- Appen blir utilgjengelig for en stor andel av brukere.</li> </ul>

Etter diskusjon med oppdragsgiver, veileder hos OsloMet og internt i gruppen ble det besluttet at alternativ 1 sannsynligvis ville føre til det beste resultatet. Vi så alternativ 1 som det beste for gruppen på grunn av at vi hadde relativt kort tid til å utvikle applikasjonen. Tidspres gjorde at vi anså Ionic som svært velegnet, ettersom det muliggjorde å skrive kode én gang som var kompatibel for både brukere med operativsystemene Apple iOS og Google Android.

## Arbeidsprosess

Vi brukte webapplikasjonen Asana som vårt prosjektstyringsverktøy. Asana er en webapplikasjon som spesielt egner seg for større gruppeprosjekter. Asana gjør det mulig å holde oversikt over hvilke oppgaver som skal gjøres, når de skal være gjort, hvem som har ansvaret for den, og mye mer.

I Asana finnes det blant annet mulighet for å lage flere prosjekter innad i samme gruppeprosjekt. Dette hadde vi bruk for da vi delte prosjektet inn i delene "App", "Dashboard", "Documentation", "Bugs - app", "Bugs - dashboard", og "General". Vi ble også anbefalt å bruke Asana av vår kontaktperson fra 99X. Ingen i gruppen hadde brukt det før, men vi lærte raskt hvordan det fungerte. Verktøyet ble vår felles plattform med 99X, ettersom vår kontaktperson også fikk tilgang til prosjektet. Vi vurderte å bruke prosjektstyringsverktøyet Trello, som har mye av den samme funksjonaliteten, men valgte heller Asana fordi den er mer omfattende.

I tillegg trengte vi en administrativ arbeidsplan for mer overordnede oppgaver, samt frister satt av OsloMet. Verktøyet ville være vår felles plattform sammen med vår veileder André. Vi valgte å bruke Trello som vår administrative arbeidsplan, både fordi vår veileder hadde erfaring med det, og fordi Trello møtte de kravene vi hadde til en administrativ arbeidsplan.

### **Hvordan vi brukte Asana**

Vi delte alle oppgavene vi skulle utføre inn i delene "App", "Dashboard", "Documentation", og "General". Innad i hver del fordelte vi oppgavene i mer beskrivende seksjoner, som "Must haves", "Nice to haves", "Testing", og "Design". Vi tildelte ansvaret for hver oppgave til en person i gruppen. Alle gruppemedlemmene hadde et hovedansvar, men på noen av oppgavene ga det mening å fordele ansvaret på flere personer. Hver av oppgavene kunne bestå av flere "subtasks" (deloppgaver) som måtte gjøres for at oppgaven skulle registreres som ferdig.

Vi markerte også oppgavene med tags (etiketter), som beskrev hvor langt i prosessen oppgaven hadde kommet. For det meste brukte vi etikettene “Todo” for det som skulle gjøres, “In progress” for det som ble jobbet med, og “Done” for det som var ferdig. Det ble også lagt inn tidsfrister på oppgavene.

Asana var oversiktlig og enkelt å bruke, samtidig som det ga mulighet for å integrere med verktøyet Slack (brukt til chat og kommunikasjon internt i gruppen). Integrasjonen med Slack gjorde at vi fikk oppgaver og status på disse opp i kommunikasjonsverktøyet vårt Slack, som igjen effektiviserte informasjonsflyten og gjorde kommunikasjonen internt i gruppen enklere. Slack diskuteres i neste seksjon.

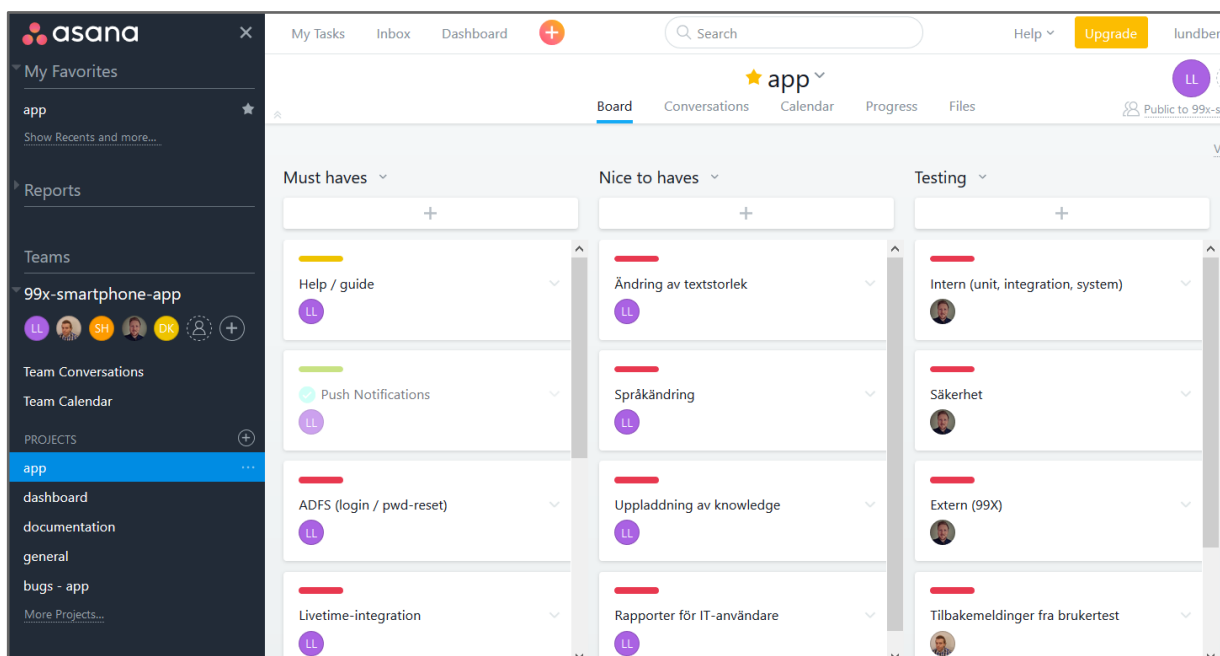


Fig. 3.1.5 - Liste med Kanban-oppgaver for smartphone-appen i verktøyet Asana

### Hvordan vi brukte Slack

Vi brukte Slack som gruppens primære kommunikasjonskanal. Slack fungerer på den måten at det opprettes et prosjekt bestående av en eller flere kanaler som kan defineres fritt. Den som oppretter prosjektet blir administrator og kan invitere de aktuelle teammedlemmene. På figuren 3.1.6 ser man at vi har delt inn prosjektet i flere forskjellige kanaler, “Code”, “General”, “Random” og “Tasks”.



Code ble brukt til å diskutere og løse programmeringsrelaterte utfordringer vi møtte underveis i utviklingen. General brukte vi til prosjektrelatert informasjon som datoer, tidsfrister og lignende. Random ble brukt for å holde ikke-prosjektrelaterte diskusjoner i en egen kanal. Til slutt hadde vi kanalen Tasks for meldinger som genereres fra integrasjonen med Asana.

Det er også mulig å kommunisere direkte til enkelte medlemmer av gruppen via “Direct Messages”. Fordelen med denne funksjonen var at meldingene kun var synlig for de som snakket sammen, noe som var nyttig hvis det eventuelt var snakk om sensitivt innhold eller om gruppemedlemmene hadde behov for å sende filer til hverandre. I tillegg ville ikke de andre i chatten bli overvældet av masse informasjon de ikke hadde behov for.

Bildet under viser også integrasjonen mellom Slack og Asana som lot oss holde oversikt over oppgavene under kanalen “Tasks”.

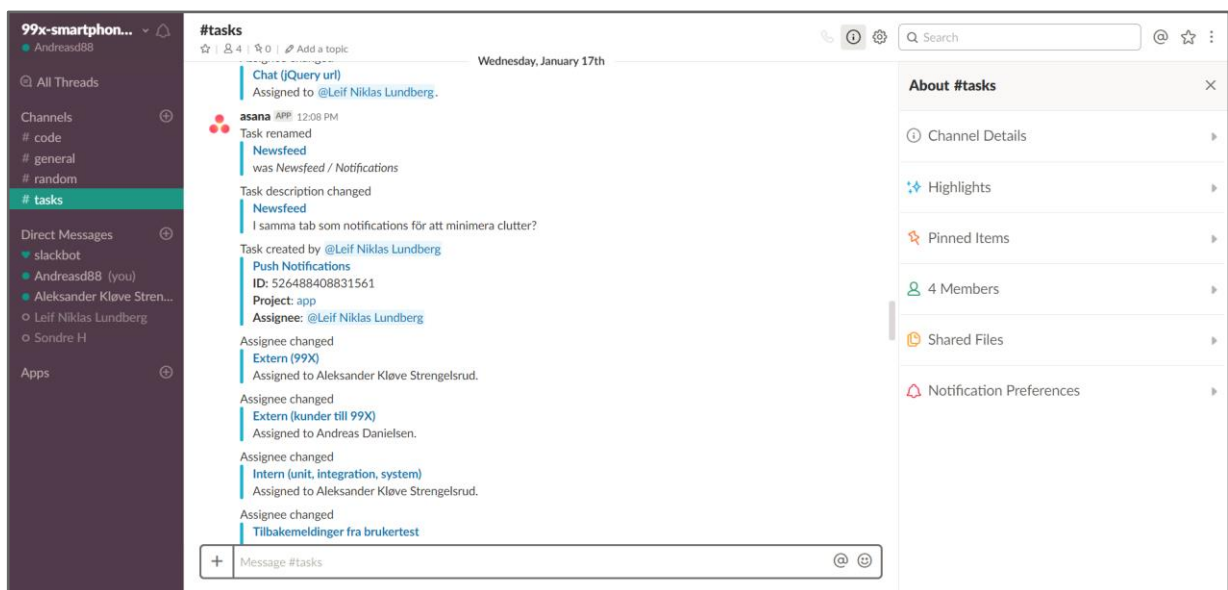


Fig. 3.1.6 - Tasks-kanalen i verktøyet Slack med meldinger sendt fra Asana

## Risikoanalyse

Det var viktig for gruppen å lage en risikoanalyse slik at vi fikk en oversikt over hendelser som kunne påvirke utviklingsprosessen. Dette var veldig nyttig under utviklingen siden analysen gjorde utviklerne klare for eventuelle problemer som kunne oppstå, slik at man kunne sette opp preventive tiltak for flest mulige risikoer.

I Risikoanalysen har vi basert risikoen på en beregning der risikofaktor er sannsynligheten multiplisert med konsekvensen. Risikoanalysen er laget med de punktene vi vektla som mest kritiske for prosjektet. Punktet med høyest risikofaktor har fått prioritet 1, med stigende rekkefølge opp til 13. Enhver risiko har et tilhørende tiltak for å sikre korrekt håndtering av risiko, forhindre gjentakelser og minimere konsekvens.

Prioritet	Kategori	Beskrivelse	Årsak	Risikofaktor
1	Sikkerhet	Applikasjonen er ikke sikker i produksjon	Sikkerhetshull er ikke oppdaget under testing/utvikling	25
2	Integrasjon	Vanskeligheter med å integrere app med dashboard/Livetime	Dårlig dokumentert/konstruert API	16
3	Tidsestimering	Estimere for mye eller for lite tid til prosjektet	Uventede/tidskrevende arbeidsoppgaver som oppstår underveis	15
4	Utviklingsmiljø	Ressurser hos oppdragsgiver er ikke tilgjengelige	Dårlig oppfølging av nøkkelpersoner hos oppdragsgiver	15
5	Kravspesifikasjon	Mangler eller endringer i kravspesifikasjon	Ikke tydelig definert kravspesifikasjon	12
6	Lisens	Verktøy har lisenser som resulterer i kostnader for bedrift	Manglende støtte fra oppdragsgiver å tilby nødvendige lisenser	10
7	Kommunikasjon	Mangelfull kommunikasjon med oppdragsgiver	Dårlig oppmøte, prioritering av andre arbeidsoppgaver	8
8	Kunnskap	Mangel på kunnskap om teknologi som skal benyttes	Mangel på erfaring, erfaring, tid, dokumentasjon	6
9	Langvarig fravær	Grupped medlem blir utilgjengelig over en lenger periode	Sykdom eller død hos grupped medlem eller nøkkelperson hos 99X	5
10	Kommunikasjon	Mangelfull kommunikasjon internt i gruppen	Dårlig oppmøte, useriøse møter, lite bruk av hjelpemidler/verktøy	5
11	Teknologi	Utfasing (deprecation) av anvendt teknologi	Komponenter er ikke lenger supporterte av operativsystemer	5
12	Kommunikasjon	Mangelfull kommunikasjon med veileder	Dårlig oppmøte og/eller oppfølging	4
13	Motivasjon	Forskjellig arbeidsinnsats internt i gruppen	Forskjellig bakgrunn, livssituasjon, innstilling og kompetanse	4

Fig. 3.1.7 - Risikoanalyse med fargekoder for alvorlighetsgrad

En risiko som inntraff var at vår kontaktperson hos arbeidsgiver ble langvarig syk. Dette gjorde at utfordringene vi møtte på under utviklingsprosessen ble mer tidkrevende å håndtere. Når vi henvendte oss til nøkkelpersoner merket vi at det vi trengte ikke ble prioritert på lik linje som om vår kontaktperson hadde vært involvert. Gruppen ble derfor nødt til å bruke unødvendig mye tid og ressurser på å innhente den hjelpen og informasjonen vi trengte på egenhånd, uten et bindeledd til bedriften.

Det var også en del sykefravær innad i gruppen, men vi løste dette på en god måte med god kommunikasjon under hele arbeidet med prosjektet.

Risikoanalysen var svært nyttig for oss sett i ettertid, ettersom alle på gruppen erfarte at det som regel oppstår uforutsette problemer og situasjoner over en så lang periode med arbeid. Erfaringen vil vi ta med oss i andre prosjekter i fremtiden.



Vi forsøkte å følge planen og tidsestimatet så godt det lot seg gjøre, men vi erfarte raskt at det skulle bli vanskelig å gjennomføre det vi hadde planlagt i praksis. Mangel på erfaring knyttet til prosjektarbeid av denne størrelsen, kombinert med upresise estimeringer kan være årsaker til vanskeligheten med å følge planen.

## Dagbok

På oppfordring fra vår veileder begynte vi fra deg én av prosjektet å skrive prosjektdagbok. Dagboken skulle vise seg å bli et nyttig verktøy for oss i arbeidet med å skrive prosjektrapporten. Muligheten for å gå tilbake i tid for å se hva som ble gjort til hvilke tider sørget for at vi ikke glemte eller utelot viktige momenter i sluttrapporten. Det var også interessant og nyttig å ha en oversikt over alle møtene som ble gjennomført med veileder, produkteier og andre som hjalp oss med prosjektet. Dagboken synliggjorde i tillegg at alt ikke gikk som planlagt, som for eksempel at det ble flere sykedager enn vi hadde forventet. Vi fikk også en indikator på arbeidsmengden som hvert gruppe-medlem utførte.

Sett i etterkant kunne vi med fordel ha skrevet enda mer detaljert og samtidig strukturert dagboken bedre for enklere lesbarhet. Det var til tider utfordrende å gå tilbake i dagboken når vi trengte mer informasjon rundt hva som ble gjort i enkelte perioder av prosjektet. Vi opplevde i noen tilfeller at notatene var gjort i for grove trekk i forhold til hva som var utført til hvilke tidspunkt.

	Aleksander	Andreas	Niklas	Sondre	ALLE
15-Jan	Skrive på forprosjekt Skaffe oversikt over bachelorprosjektets krav og standarder	Jobbe med websiden til prosjektet Få oversikt av det som skal med i dokumentasjonen	Satte opp Ionic på nytt (via npm) Initialiserte ny repository på GitHub Testet å legge til plugins (in-app calls)		Diskuterte innhold i app / dashboard med 99X
16-Jan		Så litt på websiden, men fikk ikke endret fargekoder	La in GUI-placeholders i app Basic branding (fargekoder / fonts) Containers for data (från Livetime) Saker i ny page på navigation-stack	Lagde første draft av design til dashboard	
17-Jan	Strukturere flere oppgaver i Asana Følge opp oppgaver i Trello Skrive på Forprosjekt	Jobbe med forprosjektet	Ändrade orientering på timeline Satte opp utvecklingsmiljö på laptop Visade VS Code till de andra Förbättrade roadmap Färgkodade och assignade i Asana	Lagde en veldig enkel prototype av dashboard	Andre møte med veileder
18-Jan	Skrevet og sluttet forprosjekt Booket faste tider og møterom på skolen og hos oppdrags-giver Lagt til og korrigeret saker i Asana Laget risikoanalyse	Skrevet i forprosjektet Sett litt på hjemmesiden Startet litt på risikoanalyse	Jobbade med förprojektet Fullförde riskanalys	jobbet med design av databasen til dashboard	Vi har som gruppe fått utrettet mer enn forventet i dag
19-Jan	Oppdatert og edifert Trello Korrigeret faste møte tider og rom Korrekturest og sendt inn forprosjekt og risikoanalyse Laget skisse over test / prod miljø	Lagt til task trello	Testade på Android-enhet La in placeholders för newsfeed	set error og videreutviklet dashboard	Signere og justere avtale mellom 99X, HiOA og gruppen
20-Jan					
21-Jan					
22-Jan	Skrevet på service description Lagt til oppgaver i trello og asana Jobbet med skisse av test / prod miljø Purret på testmiljø og oppsett av databaser	Laget mal for projektskisse Laget mal for sluttrapport	La in dummy-login / authentication Layoutändringar efter møte med Line	Fått opp dashboard på git. nt med dokumentering av dashb	Møte med Line angående branding / ikoner

Fig. 3.1.9 - Dagbok brukt av gruppe-medlemmene for loggføring av tidsbruk

## Kundekontakt

I forbindelse med oppstarten av prosjektet ønsket vi å komme i kontakt med kontaktpersoner og ledere hos kunder av 99X. Formålet var å kartlegge hvilken funksjonalitet de kunne tenkt seg, bruksområde og hvordan applikasjonen kunne skape ytterligere verdi i hverdagen. Dessverre viste det seg å være utfordrende å få til en dialog rundt dette. Etter gjentatte forsøk på å kontakte flere av de største kundene til 99X, valgte vi å utvikle applikasjonen på bakgrunn av en kravspesifikasjon 99X vurderte som mest verdifull.

## Resultat av planlegging

Planleggingsfasen resulterte i en rekke ressursdokumenter som ble brukt som veiledning for resten av prosjektet. I forprosjektrapporten fikk vi definert mål og rammebetingelser til produktet vi skulle levere, samt en liste over verktøy og teknologier vi hadde planlagt å bruke. De utarbeidede ressursdokumentene, spesifiserte verktøyene og teknologiene ble senere revidert og videreutviklet etter behov. Da vi hadde bestemt oss for hvilke mål produktet skulle oppfylle, kunne vi gå ut ifra det og lage brukerhistorier og en teknisk tegning av gangen til applikasjonen. Listen over verktøy og teknologier ga utgangspunkt for kapittel 3 - Teknologi og miljø i sluttrapporten.

## 3.2 Utviklingsfase

I dette underkapittelet går vi nærmere inn på hvilket rammeverk vi valgte å bruke under utviklingen. Vi snakker også om gjennomgangen gruppen hadde med produkteier, før vi så går igjennom hvordan vi håndterte bugs og problemer vi fikk under utviklingen. Vi avslutter med en gjennomgang av den interne akseptansetestingen vi gjennomførte.

### Utviklingsrammeverk

Ettersom vi ikke fikk tildelt en detaljert og tydelig kravspesifikasjon av oppdragsgiver ble det besluttet at gruppen var nødt til å bruke en agil, eller tilnærmet agil utviklingsmetodikk,

fremfor sekvensielle modeller som for eksempel fossefallsmetoden<sup>2</sup>. Agile prosesser egner seg bedre enn mer rigide alternativer til endringer i kravspesifikasjon, noe vi forventet oss fra starten av prosjektet.

“Scrum” og “Kanban” er blant de mest brukte agile utviklingsmetodikkene. Vi vurderte å bruke Scrum som utviklingsmetodikk, men av flere grunner virket Scrum som et for omfattende rammeverk. Hvis vi skulle fulgt Scrum-metodikken til punkt og prikke er det spesifisert flere regler som skal følges, som for eksempel “Det daglige møtet er ment å vare i ca. 15 minutter” eller “Et team-medlem skal kun svare på tre spørsmål på det daglige møtet”.

Vi vurderte det dit hen at Scrum-metodikken ville komme til å pålegge oss unødvendig mye tid brukt på selve utviklingsrammeverket hver dag, som ellers kunne blitt brukt til jobbing på prosjektet og utvikling. Ettersom Scrum-metodikken er mer avansert, ville det av den grunn også tatt lenger tid å sette seg inn i rutinene enn med Kanban-metoden.

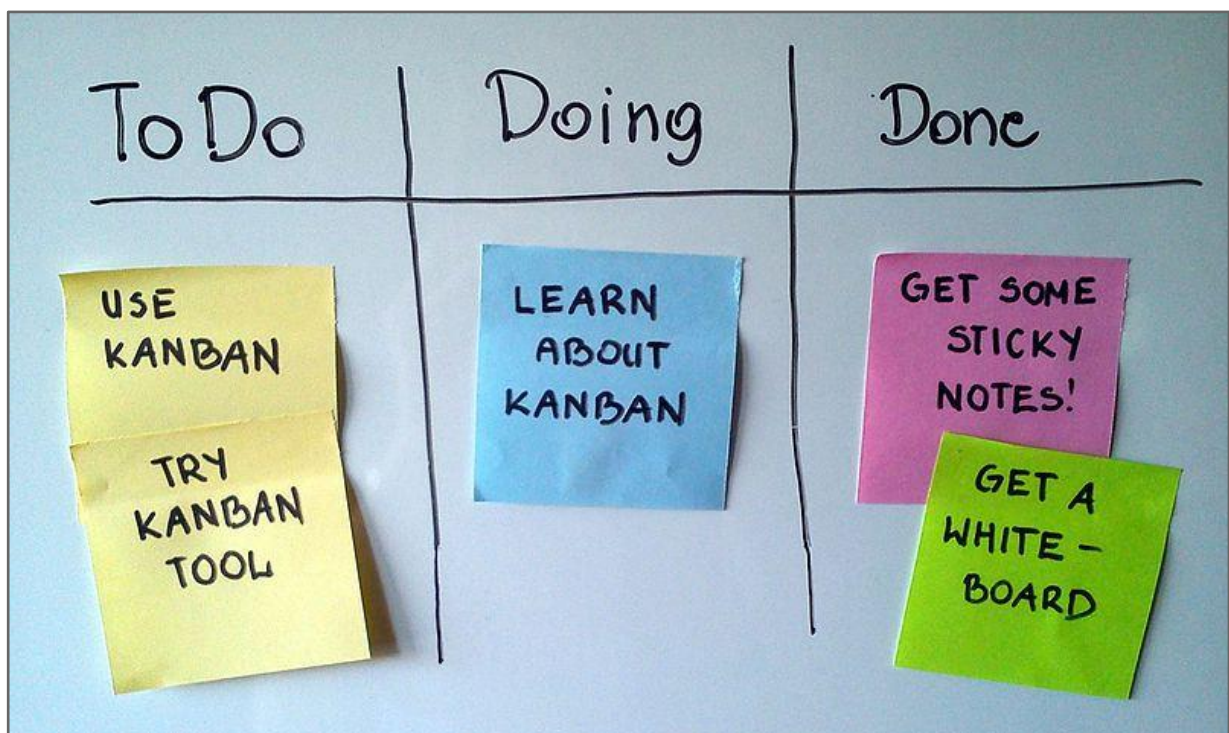


Fig. 3.2.1 - Illustrasjon av oppgave-flyten i Kanban

<sup>2</sup> Fossefallsmetoden er en lineær utviklingsmetode som definerer trinnvise predefinerte faser for utviklingen. Denne metoden skiller seg mot agile utviklingsmetoder i det at den er mindre fleksibel og mer strikt.

Det ble besluttet at vi skulle bruke Kanban til utviklingsprosessen fordi:

- Rammeverket gir mulighet til å håndtere fortløpende endringer i kravspesifikasjonen
- Bruk av "Kanban-boards" (vedvarende tavler med arbeidsoppgaver) gir en oversiktlig fremstilling av planlagte, pågående og avsluttede oppgaver
- Gruppemedlemmene er kjent med utviklingsrammeverket fra før av
- Metodikken egner seg godt til et prosjekt bestående av flere komponenter (ett dedikert kanban-board per prosjekt)
- Kanban muliggjør at ethvert gruppemedlem kan plukke åpne oppgaver når personen har fullført sine aktive oppgaver, noe som forenkler delegeringen av arbeid
- Avhengighet av interne ressurser hos oppdragsgiver kan lede til tidsforsinkelser, noe tidsbaserte rammeverk som for eksempel Scrum ikke håndterer

Kanban har relativt få regler og retningslinjer som vi forsøkte å følge så langt det lot seg gjøre, men likevel har vi fraveket noe fra metodikken. Et viktig konsept med Kanban er at en person ikke skal jobbe med for mange oppgaver samtidig. Dette har vi ikke alltid klart å følge, ettersom vi har hatt flere viktige komponenter som har krevd kontinuerlig arbeid og da ble liggende som oppgaver under arbeid en lenger tid.

Vi brukte Asana til å håndtere våre Kanban-boards (se 3.1 - Arbeidsprosess). Fra start fylte vi inn alle kravene vi hadde satt til produktene i kravspesifikasjonen. I løpet av utviklingen hadde vi ukentlige møter med oppdragsgiver (unntatt når vår kontakt ble langtidssykemeldt) der vi på fortløpende gikk gjennom kravene etterhvert som produktene ble utviklet.

### Gjennomganger med produkteier

I løpet av prosessen med å utvikle både applikasjonen og dashboard-løsningen hadde vi tett dialog med produkteier. Dialogen bestod i hovedsak av regelmessige statusmøter på enten mandag eller fredag. I tillegg hadde vi møter med andre interne ressurser angående testmiljø, integrasjon, design med mer ved behov. Under hvert møte skrev vi korte referat som omhandlet møtets agenda og hva vi kom frem til. Vi forberedte også spørsmål og punkter underveis til neste møte med produkteier, ettersom vi underveis i utviklingen ofte måtte ta avgjørelser hvor vi ønsket å involvere produkteier.

I 8.2 - Møteforberedelse og referat har vi lagt ved eksempler på hvordan et typisk møtereferat og forberedelse til neste møte kunne se ut. I tillegg har vi lagt med en typisk skisse som ble resultatet etter ett av møtene med nettverk og sikkerhetsansvarlig hos 99X.

I oppstartsfasen var det viktig for oss å ha dialog med markedsføringsavdelingen til 99X, som blant annet har ansvar for hvordan 99X fremstår utad mot de eksisterende kundene og potensielle nye kunder. Etter et møte, hvor vi viste frem en prototype av applikasjonen, fikk vi konstruktive tilbakemeldinger på fargetemaer, flyt i applikasjonen, ikoner og oppsett av menyer. Vi går nærmere inn på prototypen i neste avsnitt. Vi jobbet også tett mot produkteier for å få et test- og utviklingsmiljø klart til videre utvikling. I følge risikoanalysen vår så vi utfordringer knyttet til testmiljø og integrasjon som to av de mest kritiske punktene knyttet til prosjektet, og ønsket derfor å legge mye arbeid i å få testmiljøet tidlig på plass.

## Prototyper

Oppdragsgiver ønsket at vi kom tidlig i gang med å lage en prototype av applikasjonen og dashboardet. Prototypen gjorde det enklere for 99X å komme med innspill tidlig og samtidig sørget den for at gruppen raskt kom i gang med utviklingen. Den gjorde det i tillegg enklere å diskutere ideer og konsepter med produkteier, ettersom vi kunne vise en visuell og interaktiv representasjon av den tenkte applikasjonen.

Allerede i slutten av januar 2018 hadde vi klar vår første prototype av mobilapplikasjonen og dashboardet. I forbindelse med å lage prototypen forsøkte vi også å ha et bevisst forhold til hvordan applikasjonen skulle se ut i forhold til design og funksjonalitet. Prototypen inneholdt ingen faktisk funksjonalitet (tilknytning til servere og lignende), men var i stedet utelukkende ment til å gi en visuell fremstilling av våre tanker rundt utseende, navigasjon og tenkt funksjonalitet.

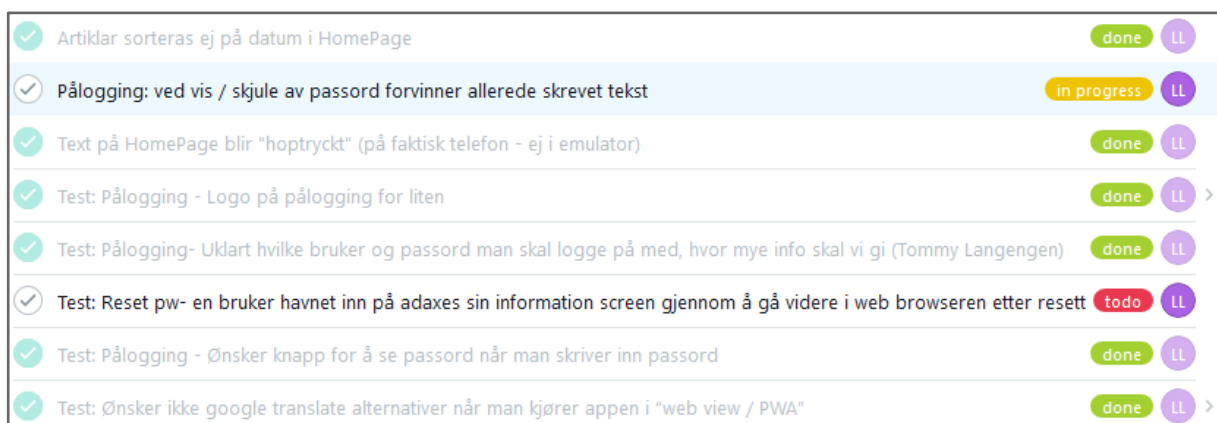
I 8.3 - Prototyper ligger bilder av de omtalte prototypene til både mobilapplikasjonen og dashboardet.



## Bugs og problemer

Vi brukte Asana (vårt Kanban-board) for å håndtere bugs og mangler i applikasjonene. Dette var bugs vi fant når vi programmerte appen eller dashboardet, og vi delte derfor opplistingene av bugs i "App" og "Dashboard". Vi definerte bugs som et problem der vi ikke fant en umiddelbar løsning. Fordelen med å håndtere bugs på denne måten var at alle i gruppen hadde oversikt og kjennskap til problemer som måtte løses, samtidig som vi også fikk historikk på når bugs ble rettet og hvem som utførte endringen.

Små endringsforespørsler fra test av appen ble også inkludert i denne listen, hvis de ikke var omfattende nok til å legges inn som en dedikert task/oppgave.



✓ Artiklar sorteras ej på datum i HomePage	done	LL
✓ Pålogging: ved vis / skjule av passord forvinner allerede skrevet tekst	in progress	LL
✓ Text på HomePage blir "hoptryckt" (på faktisk telefon - ej i emulator)	done	LL
✓ Test: Pålogging - Logo på pålogging for liten	done	LL >
✓ Test: Pålogging- Uklart hvilke bruker og passord man skal logge på med, hvor mye info skal vi gi (Tommy Langengen)	done	LL
✓ Test: Reset pw- en bruker havnet inn på adaxes sin information screen gjennom å gå videre i web browseren etter resett	todo	LL
✓ Test: Pålogging - Ønsker knapp for å se passord når man skriver inn passord	done	LL
✓ Test: Ønsker ikke google translate alternativer når man kjører appen i "web view / PWA"	done	LL >

Fig. 3.2.2 - Liste i Asana over bugs, med status og gruppemedlem ansvarlig for utførelse

## Testing

Testing anses som viktig i prosessen med å utvikle programvare for å oppnå robuste produkter med lite feil per kodelinje. Enkelte former for testing har vi erfart i tidligere fag ved OsloMet kan være svært tidkrevende. I starten av prosjektet ønsket gruppens medlemmer å gjennomføre en fullskala enhets-, integrasjons-, system- og akseptansetesting. Etterhvert som prosjektet utviklet seg ble det klart for oss at vi ikke kom til å ha nok tid og ressurser til å gjennomføre alle de nevnte stadiene av testing. Vi var nødt til å omprioritere basert på tid til rådighet, og valgte derfor å gjennomføre akseptansetest med interne brukere hos oppdragsgiver. Akseptansetesten sørget for at funksjonalitet i både webapplikasjonen og mobilapplikasjonen ble testet så grundig som mulig, med den tiden vi hadde tilgjengelig.

## Akseptansetesting

Akseptansetesten hadde til hensikt å gi oss tilbakemelding på om mobilapplikasjonens funksjonalitet, på testtidspunktet, stemte overens med definert funksjonalitet i kravspesifikasjonen. Vi var også ute etter tilbakemeldinger som omhandler mobil og webapplikasjonens design, struktur av elementer og generell brukervennlighet.

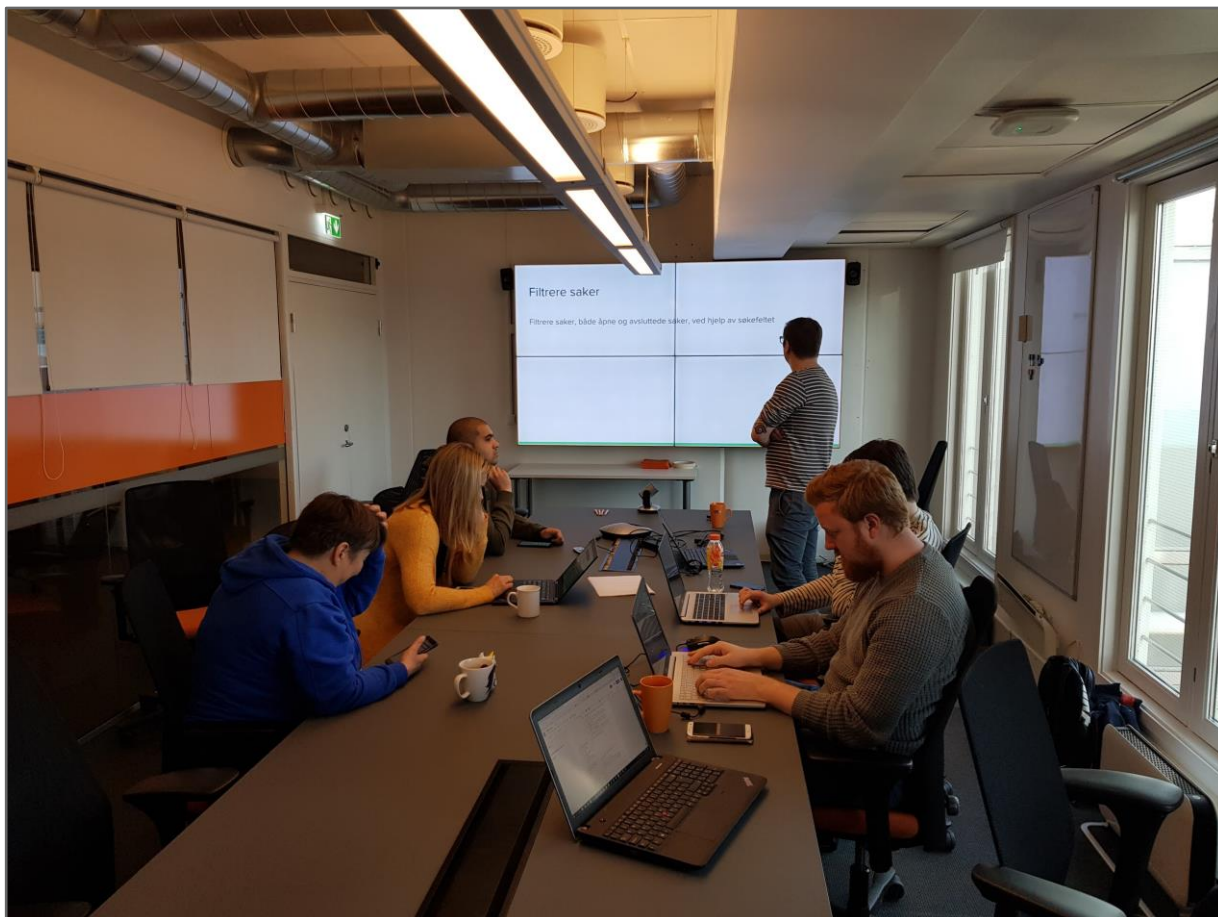
## Testgjennomføring



*Fig. 3.2.3 - Bilde fra utførelse av test hos oppdragsgiveren*

Dagen før gjennomføringen av testingen var hektisk og inneholdt en rekke utfordringer som måtte løses. Den største utfordringen var å få testmiljøet til å fungere tilfredsstillende, ettersom vi ikke hadde hele systemet kjørende i testmiljøet tidligere enn dagen før selve testingen. Når vi flyttet over kodebasen fra lokale maskiner til miljøet var det noen funksjoner som sluttet å fungere, deriblant pålogging. Vi lyktes likevel etter en lang kveld å få testmiljøet klart til testingen dagen etter.

Det var utfordrende å samle alle de påmeldte testkandidatene, selv om de i god tid på forhånd var innkalt til testen. Noen av deltakerne var nødt til å prioritere kundehenvendelser og andre kunne ikke delta som følge av sykdom. Vi fikk beskjed kort tid før testen at noen av kandidatene ikke kunne delta. Da vi fant ut at det kom til å bli færre personer enn forventet inviterte gruppen en ny deltager. Som en konsekvens av frafallet ble testingen utført av fire testkandidater der en av de var med via Skype.



*Fig. 3.2.4 - Bilde fra utførelse av test hos oppdragsgiveren*

Testingen startet ved at vi hjalp testpersonene med å installere applikasjonen på deltakernes telefoner. Deretter fikk deltakerne en kort innføring i hvordan testingen skulle gjennomføres. Gruppen hadde gjort klar en Power Point presentasjon som vist på *fig. 3.2.4*, der det ble forklart hvilke testscenarier testpersonene skulle utføre. For å gjennomføre testing av mobilapplikasjonen benyttet deltakerne sin egen 99X-bruker.

Testingen av hvert enkelt scenario ble gjennomført slik at testpersonene først løste den aktuelle casen, for å så diskutere scenarioet sammen med gruppe-medlemmene. På den måten fikk vi mye konstruktive tilbakemeldinger på hva som måtte forbedres, endres, legges til og/eller hva testpersonene syntes var bra.

Det ble under planlegging av testen satt av én time fra oppdragsgivers side til gjennomføring av testen. Mens vi planla gjennomføringen og utarbeidet testoppgavene ble det klart for oss at tiden sannsynligvis ikke var nok til å komme igjennom alle de planlagte use-casene. På testdagen ble vi ytterligere forsinket ettersom flere av testpersonene ikke møtte til avtalt tidspunkt. Det ble som en konsekvens av tidsnøden kun tid til å teste mobilapplikasjon use-casene, samt en kort gjennomgang av dashboardet. Gjennomgangen ble gjort av gruppen, og vi fikk derfor ikke gjennomført use-casene som var planlagt for dashboardet.

Vi erfarte også at vi med fordel kunne hatt med flere testpersoner under testing av applikasjonen, både interne og eksterne. Under planlegging av testen forsøkte vi å innhente flere personer, både fra 99X og kunder av 99X, men dette lot seg dessverre ikke gjennomføre på grunn av at 99X og de eksterne bedriftene ikke hadde mulighet til å bidra eller avsette flere testpersoner.



Fig. 3.2.5 – En av testpersonene fyller inn skjemaet utlevert etter test

Etter selve gjennomgangen av testingen fikk hver enkelt av testpersonene en skriftlig spørreundersøkelse (se 8.4 - Akseptansetest). I undersøkelsen fikk de en rekke spørsmål som de kunne svare på i etterkant av testen (se fig. 3.2.5). Vi ønsket at testpersonene besvarte spørreundersøkelsen rett etter testgjennomføring, for å sørge for at vi fikk så mye tilbakemeldinger som mulig mens testpersonene hadde gjennomgangen friskt i minnet.

Dessverre første tidsbegrensningen til at spørreundersøkelsen ble sendt med hver enkelt etter testen i stedet for at den ble gjennomført omgående. Som vi mistenkte ble besvarelsen av undersøkelsen nedprioritert når testpersonen forlot testmøtet, og vi var nødt til å purre gjentatte ganger på tilbakemeldinger. Dette resulterte i noe varierende kvalitet på tilbakemeldingene, men enkelte av de tilbakemeldingene vi fikk hjalp oss til å forbedre sluttproduktet.

### Smartphone-applikasjon

<b>Gruppemedlemmer tilstede:</b>	Aleksander, Niklas, Sondre og Andreas
<b>Testpersoner:</b> (interne fra 99X)	Anne-Marit, Line, Oskar (via Skype) og Waqas
<b>Testbrukere benyttet:</b>	Testpersonens reelle brukere i produksjonsmiljø ble benyttet
<b>Periode (fra-til):</b>	fre 13.04.2018 kl. 9:00–10:00
<b>Testoppgaver:</b>	Vi spesifiserte mobilapplikasjon testoppgaver ut i fra brukerhistorier definert i kravspesifikasjonen. (Se 8.4 - Akseptansetest)
<b>Funksjonalitet testet:</b>	<ul style="list-style-type: none"> <li>● Passordtilbakestilling</li> <li>● Pålogging</li> <li>● Finne oversikt over saker (åpne / lukkede)</li> <li>● Finne detaljert saksinformasjon</li> <li>● Filtrere saker</li> <li>● Sende inn en ny suppothenvendelse</li> <li>● Kontakte servicedesk</li> </ul>

	<ul style="list-style-type: none"> <li>● Chatte med servicedesk</li> <li>● Finne og lese nyheter</li> <li>● Finne og lese kritiske hendelser (notifikasjoner)</li> <li>● Finne og endre på innstillinger</li> <li>● Logge ut av mobilapplikasjonen</li> </ul>
<p><b>Oppsummering av tilbakemeldinger:</b></p>	<p>Tilbakemeldingene på testingen av mobilapplikasjonen var veldig bra. Her fikk gruppen en masse punkter på hva vi som utviklere tenkte var klart og tydelig, men som testpersonene ikke nødvendigvis hadde samme oppfatning av. Det var også mange konkrete tilbakemeldinger på hva som kunne forbedres. Testpersonene synes også det produktet de testet så veldig bra ut og gruppen fikk skryt for en god jobb så langt.</p> <p>Viser til 8.4 - Akseptansetest og kolonnen "Testpersonens tilbakemeldinger" for uformaterte tilbakemeldinger notert under selve akseptansetesten.</p>
<p><b>Konklusjon:</b></p>	<p>Testingen var veldig nyttig for oss som gruppe. Vi fikk veldig mange gode tilbakemeldinger på hva som kunne forbedres og hva vi hadde gjort bra, samt at vi lærte at tiden vi har til rådighet for testing må planlegges bedre. Antall testpersoner var et problem for oss da vi fikk frafall på 3-4 stykker på grunn av sykdom. Sykdomsfraværet gjorde testingen mer kvalitativ enn kvantitativ, og kunne føre til at testgruppen ikke fant like mange forbedringer som en større gruppe kanskje hadde funnet. Testing av applikasjonen eksternt er også noe vi svært gjerne ville gjøre, men tiden vi hadde til rådighet satt en stopper for det.</p> <p>Vi er godt fornøyd med gjennomføringen av testingen. Vi erfarte at ting går svært sjelden etter planen og at vi må estimere vesentlig mer tid til gjennomføringen av en test neste gang.</p>

## Dashboard

<b>Gruppede medlemmer tilstede:</b>	Aleksander, Niklas, Sondre og Andreas
<b>Testpersoner:</b> (interne / eksterne)	Anne-Marit, Line, Oskar og Waqas
<b>Testbrukere benyttet:</b>	En testbruker vi lagde under utvikling
<b>Periode (fra-til):</b>	fr 13.04.2018 kl. 9:00–10:00
<b>Testoppgaver:</b>	Vi spesifiserte dashboard testoppgaver ut i fra brukerhistorier definert i kravspesifikasjonen. (Se 8.4 - Akseptansetest)
<b>Funksjonalitet testet:</b>	<ul style="list-style-type: none"><li>● Pålogging</li><li>● Publisere en artikkel</li><li>● Redigere en artikkel</li><li>● Slette en artikkel</li><li>● Se statistikk over publiserte artikler</li></ul>
<b>Oppsummering av tilbakemeldinger:</b>	<p>Vi fikk ikke kjørt noen brukertester av dashboardet, så tilbakemeldingene vi fikk var kun etter en kjapp gjennomgang av hvordan dashboardet fungerte. Vi hadde ønsket oss bedre tilbakemeldinger her, men tiden strakk dessverre ikke til.</p> <p>Viser til dashboard testoppgaver og kolonnen "Testpersonens tilbakemeldinger" for uformaterte tilbakemeldinger notert under selve akseptansetesten.</p>
<b>Konklusjon:</b>	<p>Som nevnt tidligere fikk ikke gruppen avsatt nok tid til testingen. Dette gikk utover testing og gjennomgang av dashboardet.</p> <p>En ting vi lærte av denne prosessen var at vi burde "mast" mer for å fått satt av mer tid til testingen av produktet. Vi burde ha testet mer for å ha levert et sluttprodukt som ikke bare vi som utviklere syntes var intuitivt og bra, men også noe brukerne ville vært fornøyde med.</p>

### 3.3 Evaluering av utviklingsprosessen

I dette kapittelet skriver vi om noen av de problemene vi møtte under utviklingen av mobilapplikasjonen, APIet og dashboardet.

#### Miljøer og tilganger

Arbeidet med å få testmiljøet på plass var krevende ettersom vi var avhengig av å få godkjenning fra ressurser hos oppdragsgiver. Disse var lite tilgjengelig på grunn av en hektisk hverdag. Vi forsøkte over en lengre periode å avtale et møte for å avklare hvordan vi skulle gå frem for å få godkjent og satt opp et testmiljø.

Etter det første møtet med sikkerhet og nettverksansvarlig ble det klart at vi måtte dokumentere en del punkter rundt hvordan vi så for oss at testmiljøet skulle se ut. Det inkluderte hvilke porter vi hadde behov for å åpne, hva som måtte være tilgjengelig internt og eksternt og hvordan dataflyten i det tenkte systemet skulle være.

Når de endelige tilgangene vi hadde bestilt var på plass møtte vi flere nye utfordringer. Det viste seg at brukerne vi hadde fått opprettet manglet tilstrekkelige rettigheter til blant annet databasene, og noen av påloggingene var sperret. Portene vi hadde fått åpnet viste seg å avvike fra standard (for eksempel port 1433 for databasetilkoblinger) når vi forsøkte å nå serveren via feilsøkningsverktøyet telnet. Dette førte til at vi måtte bestille rekonfigurasjon av åpningene i brannmurene. Når løsningen skulle publiseres på internett viste det seg at vi måtte åpne enda flere blokkeringer for å tilgjengeliggjøre dashboardet og APIet for eksternt trafikk.

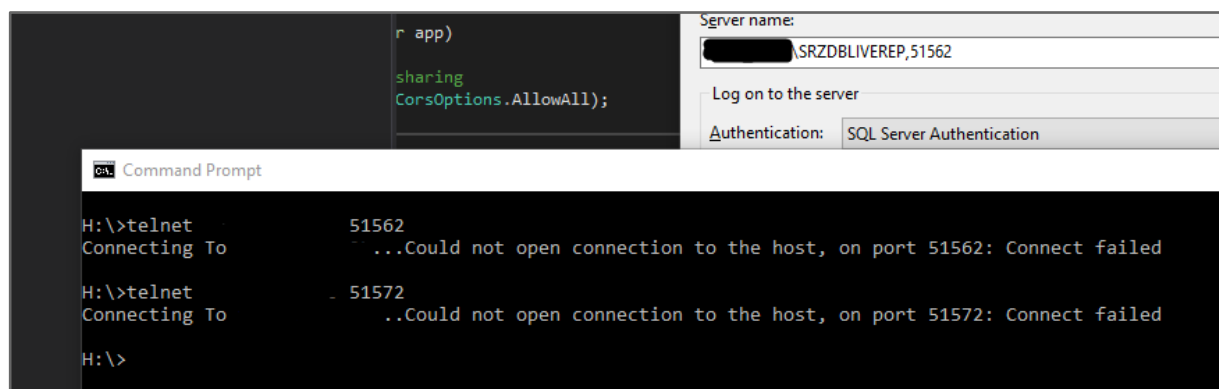


Fig. 3.3.1 - Feilmelding ved test av portåpninger mot databaseserveren



Et ytterligere problem var det at det ofte var vanskelig å vite hvilken person hos oppdragsgiver vi måtte kontakte avhengig av hvilken tilgang det gjaldt; hvem som faktisk hadde system eller administratortilgang og hvem som hadde kompetanse i området. Ofte fikk vi motsigende informasjon om hvem som kunne gjøre hva. Når vi omsider fikk klarhet i hvilke ressurser vi måtte kontakte var det heller ikke alltid lett å få kontakt med disse personene grunnet deres individuelle arbeidsmengder, lokasjoner og arbeidstider.

## Data fra saksbehandlingssystemet

Ettersom applikasjonen henter data fra et eksisterende saksbehandlingssystem måtte vi få tilgang til og sette oss inn i oppsettet til systemets database, for å deretter kunne konstruere og utføre egne spørringer mot denne.

Når alle tilganger var på plass og vi hadde mulighet til å få oversikt over hvilke data vi trengte å hente ut oppdaget vi at databasens struktur var svært dårlig designet. Den inneholdt et flertall ubrukte kolonner fulle av NULL-merker (altså “tomme felter”, noe som er ønskelig å unngå ved design av en databasestruktur). Samtidig var databasen i sin helhet unødvendig komplisert grunnet mengder av uoversiktlige og/eller duplikate tabeller samt forvirrende navngiving. Ettersom vi ikke hadde mulighet til å endre oppsettet til databasen var vi nødt til å akseptere strukturen og bruke en del tid på å finne ut av hvor dataen vi trengte befant seg.

## Publisering av appen til butikker

Et krav vi fikk av oppdragsgiver var at appen skulle kunne publiseres til både Google Play og App Store. Dette er de største app-butikkene på markedet, og ved å kun publisere til én av butikkene ville appen blitt mye mindre tilgjengelig. Google Play er Google sin butikk for apper som bruker Android-operativsystemet, mens App Store er Apple sin butikk for apper som bruker iOS-operativsystemet.

Publisering til Google Play er en enklere prosess enn å publisere til App Store. Man er nødt til å registrere seg i Google Play Console, som er portalen for utviklere som ønsker å publisere sine apper. For å kunne registrere seg er man nødt til å betale en engangsavgift på \$25.

For å publisere til App Store er man nødt til å registrere seg i Apple sitt Apple Developer Program. Apple krever \$99 i året for å kunne være med i programmet. I tillegg må personen som registrerer seg til programmet være en ansatt hos bedriften med rettighet til å ta beslutninger på bedriften sin vegne. Dette vil vanligvis være en prosjektleder eller en ansatt som er i ledelsen. Apple har også et krav om at bedriften som søker om et Apple Developer Program abonnement må ha et D-U-N-S-nummer<sup>3</sup>.

På et ukentlig møte beskrev vi situasjonen til vår kontaktperson for bedriften, David, og gjorde han klar over hva de må gjøre for å kunne publisere til butikker. David sendte en forespørsel for å registrere et D-U-N-S-nummer, og fikk svar i løpet av dagen. Den neste uka hadde han også fått betalt beløpet for å registrere seg i Google Play Console og Apple Developer Program.

Vi fikk dessverre ikke mulighet til å publisere appen, men gjorde derimot 99X bevisste på at appen var ferdig utviklet, og at de kunne publisere den når de hadde mulighet til det. Vi fikk ikke selv publisert appen fordi vår kontaktperson ble langtidssyk, og vi trengte godkjenning fra han før vi hadde mulighet til å publisere.

### Cross-plattform-utvikling via Ionic

Det kan argumenteres for at det er bedre å utvikle applikasjoner som native (altså en dedikert kodebase for hvert operativsystem) *hvis teamet har mulighet for det*. Begrunnelsen for dette er at programmereren da kun bruker metoder og systemer designet av utvikleren av operativsystemet spesifikt for endemålet, som er å utvikle applikasjoner innenfor rammeverket. Dette vil da sannsynligvis redusere forekomsten av bugs som oppstår grunnet “workarounds”, samtidig som ytelse (oppstartstid, responsivitet, etc.) blir forbedret.

Ved bruk av et cross-plattform-rammeverk som for eksempel Ionic ofrer utvikleren disse fordelene, samtidig som noen flere problemfaktorer oppstår. Vi erfarte følgende utfordringer med rammeverket:

---

<sup>3</sup> Et D-U-N-S-nummer er et identifikasjonsnummer til en bedrift, og ble utviklet av det amerikanske firmaet Dun & Bradstreet, Inc. De holder en database over bedrifter fra hele verden, og er en av de mest anerkjente metodene for å identifisere bedrifter.

## **Avhengighet av plugins**

For å tilby mulighet for native-funksjonalitet (for eksempel tilgang til telefonens kamera eller andre interne ressurser) benytter Ionic seg av såkalte plugins. Disse er små komponenter, laget for Cordova (rammeverket som Ionic bygger på), som importeres og integreres i applikasjonen ved behov. Problematikken med denne løsningen er at det ikke er garantert at hver plugin er vedlikeholdt, da de ikke alltid er utviklet av hverken Ionic eller Cordova. For vårt prosjekt betyr dette at hver plugin måtte vurderes om den var av høy kvalitet og kompatibel med vår løsning.

## **Ytelse**

En annen negativ faktor med cross-platform-utvikling er at applikasjonen aldri kommer til å være like rask som en applikasjon utviklet til spesifikke operativsystemer. Årsaken til dette er at det er mer ressurskrevende for en telefon å håndtere grafiske elementer kodet i HTML5 og CSS, fremfor å tegne opp grensesnitt via de lettere plattformspesifikke elementene.

Vi har dog klart å mitigere problemet ved hjelp av å ta i bruk "lazy-loading" - en populær teknikk som innebærer at ressurser kun lastes inn når de faktisk kreves. Når appen startes vil kun det første skjermbilde og tilhørende ressurser bli lastet inn, noe som viste seg å redusere oppstartstiden drastisk.

## **Styling**

I likhet med ytelsen kan også bruken av HTML5- og CSS-elementer ha innvirkning på utseendet til applikasjonen, og det kan være vanskelig å oppnå en "native-følelse". Heldigvis har de fleste av komponentene fra Ionic støtte for grafiske elementer som automatisk blir tilpasset iOS, Android og Windows Phone. For eksempler blir ikoner fra rammeverket automatisk endret til korrekt typ avhengig av aktuelt operativsystem (sånn som Material Design for Android). Bakdelen med disse komponentene er at de ofte er vanskelige å tilpasse.

## Dokumentasjon av rammeverket

Ettersom ingen av oss hadde brukt Ionic tidligere førte dette til at det ble en stor del leting etter dokumentasjon og eksempler på nett. Versjon 3 av Ionic, som vi valgte å benytte til utviklingen, er fortsatt forholdsvis ny (april 2017) og derfor omhandler den største delen av diskusjon, dokumentasjon og eksempler på internett eldre versjoner av Ionic.

Utfordringen ble enda større i og med de endringene som ble gjort fra versjon 2 til 3. Rammeverket byttet fra AngularJS til Angular, noe som førte til at programmeringsspråket ble endret fra JavaScript til TypeScript, og gamle eksempler ble da enda vanskeligere å konvertere til Ionic 3.

## Kodekompleksitet i dashboardet

Vi hadde tidligere dashboardet og APIet i samme prosjekt, men av flere grunner valgte vi å dele de opp i to separate prosjekter. En av grunnene var at det ville være mest brukervennlig for brukerne av systemet. Det skulle være mulig å kommunisere med APIet uten å gå gjennom dashboardet, for eksempel hvis man ville hente alle nyheter som var publisert.

En annen grunn til at vi delte opp prosjektene var for å forenkle koden som kommuniserte med APIet. Det ble vurdert at all kode som ble brukt av både smartphone-applikasjonen og dashboardet egnest seg best som et separat prosjekt. På den måten ble kodebasene for APIet og dashboardet mindre og dermed lettere å feilsøke, vedlikeholde og sikre.

## Autentisering av sluttbrukere

Av den grunn at et krav fra oppdragsgiver var at sluttbrukere av applikasjonen måtte kunne logge på med sine vanlige domenebrukere (samme konto som bruker til å logge seg inn på jobb-pc eller email) var vi nødt til å implementere en smart løsning for dette problemet.

Fordi samtlige kunder av oppdragsgiver benytter seg av Microsoft-miljøer, og dermed systemet Active Directory for håndtering av påloggingsinformasjon og annen brukerdata, hadde vi kun et system å forholde oss til.

Det viste seg at selve prosessen med å autentisere en bruker via ekstern kode var mye mer komplisert enn hva vi hadde forutsett. En rekke av de eksemplene vi fant online passet enten ikke vår situasjon eller var utdaterte og/eller dårlig dokumenterte.

Etterhvert viste det seg at et eksternt utviklerfirma allerede hadde implementert en lignende løsning for en annen applikasjon i oppdragsgiverens system, der det var mulig å logge på med domenebrukere fra forskjellige kunder. Vi tok kontakt med en utvikler hos dem, som hjalp oss på rett vei gjennom å besvare noen uklarheter. Han delte også et kodeeksempel som var kjernen i løsningen. Denne passet ikke vår situasjon ideelt, men ga oss et grunnlag for å implementere vår egen autentiseringsmetode. Implementasjonen blir grundig forklart i kapittel 5.3.

### Kontaktpersontilgjengelighet

Vi fikk lenge i prosjektet svært god oppfølging av vår kontaktperson hos oppdragsgiver. Ved spørsmål eller problemer, enten det var knyttet til testmiljø eller systemer vi skulle integrere, så fikk vi raskt hjelp eller ressurser på plass som kunne bidra. Dessverre endret situasjonen seg mot slutten av prosjektet, ettersom vår kontaktperson ble sykemeldt. Vi fikk ikke beskjed fra vår kontaktperson om hvem vi skulle forholde oss til i hans fravær, noe som ga oss en del utfordringer knyttet til slutfasen.

Det var heller ikke andre tilgjengelige ressurser som fikk delegert et direkte ansvar for oss som gruppe i den resterende perioden. Konsekvensen av fraværet ble at vi var nødt til å ta en del valg på egenhånd. Disse valgene var knyttet til design, men også spørsmål angående prioriteringer, sikkerhet, publisering av applikasjonen og hva som kunne brukes til presentasjonen av applikasjonen.

### Sikkerhetsgjennomgang

Vi avtalte med oppdragsgiver under planleggingen av prosjektet at vi skulle gjennomføre en sikkerhetsgjennomgang av applikasjonen. Gjennomgangen skulle utføres av et eksternt selskap som spesialiserer seg på området. Gruppen hadde et sterkt ønske om å gjennomføre en slik gjennomgang, da vi hadde sett frem til å få en profesjonell kvalitetssjekk av løsningen.

Dessverre fikk vi ikke gjennomført dette, på grunn av det nevnte sykefraværet hos vår kontaktperson, som skulle organisere denne gjennomgangen for oss. Etersom et av våre fokusområder var å utvikle en sikker mobil- og webapplikasjon synes vi det er leit å ikke kunne legge ved resultatet av en slik sikkerhetsgjennomgang i oppgaven vår.

For å minimere risikoen for sikkerhetshull har vi dog gjennomført kontinuerlig code review (kodegjennomganger), der en annen person enn den som har laget selve koden ser over den etter feil og mulige forbedringer. Dette er noe vi har gjort før hver større commit (oppdatering av kodebasen). Code reviews er et enkelt tiltak som foruten om oppdagelsen av sikkerhetshull og feil kan bidra til å styrke kompetansen til alle i gruppen da alle setter seg inn i de forskjellige systemene og løsningene vi bruker.

## 4 Teknologi og miljø

I tekniske prosjekter er valg av teknologier, støttesystemer og utviklingsmiljøer en svært viktig komponent for å sikre en god gjennomføring. Det er viktig å kartlegge forskjellige teknologier og verktøy, samt evaluere hvilke som kanskje passer bedre eller mindre godt til prosjektet. Når vurdering av nødvendig teknologi ble utført, brukte vi følgende kriterier for å sikre vedlikeholdbarhet, forutsigbarhet, kompatibilitet og nytteverdi for prosjektet:

- Er teknologien godt dokumentert og finnes det ressurser for teknologien?
- Er teknologien brukt i anerkjente produkter og har den bestått “test of time”?
- Kan teknologien integreres med andre valgte teknologier?
- Er teknologien aktivt vedlikeholdt?
- Kan vi bruke teknologien iht. lisens?

Med et godt dokumentert prosjekt forventer vi at utvikleren av teknologien har utarbeidet en oversiktlig dokumentasjon om hvilken funksjonalitet teknologien tilbyr og hvordan det er tenkt at den skal brukes. Hvis teknologien har åpen kildekode er det også ønskelig at selve koden er leselig og av høy kvalitet. Utenom den offisielle dokumentasjonen er det også fordelaktig hvis det finnes ressurser som offentlige forum, veiledninger, diskusjoner på Stack Overflow (en meldingstavle for kodeproblemer) og lignende.

“Test of time” innebærer at selve teknologien har vært i bruk i reelle prosjekter/produkter over en lengre periode, og at eventuelle bugger og andre feil har blitt korrigert. Hvis mange profilerte aktører også velger å bruke det samme rammeverket kan man anta at teknologien er sikker og godt analysert. Om de overnevnte kriteriene er innfridd har vi ikke sett noen problemer med å benytte slik teknologi i vårt prosjekt.

Det var også viktig for oss at vi brukte teknologier som var kompatible med hverandre. Når vi for eksempel skulle velge rammeverk for smartphone-appen var det viktig at rammeverket var kompatibelt med alle de forskjellige løsningene for push-notifikasjoner, chat-funksjonen, henting av nyheter og så videre som vi benytter.

Ettersom produktet blir overlevert oppdragsgiver er det mulig at de ønsker å videreutvikle prosjektet internt eller via et eksternt firma. I den forbindelse har vi satt som kriterium at teknologien må være under aktiv utvikling, for å sikre at den er fremtidsrettet. Hvis det for eksempel lanseres en ny versjon av Android eller iOS som ikke er kompatibel med vår app (noe som er veldig sannsynlig) er det utvikleren av teknologien som må sørge for at kompatibiliteten blir vedlikeholdt.

I forbindelse med vurdering av lisenser har det vært viktig at de ikke hindrer oss i utviklingen eller under levering til oppdragsgiver. Vi har valgt teknologier med mer tillatende lisenser som MIT[5] fremfor GPL, da den sistnevnte krever at det som utvikles legges under samme lisens[6].

Vi har vurdert helheten til samtlige av verktøyene nevnt videre i kapittelet, og i de neste seksjonene (4.1 og 4.2) ser vi nærmere på noen av de mest kritiske av de utvalgte teknologiene.

## 4.1 Front-end-teknologier

Front-end (også kalt presentasjonslaget) er de grensesnittene og applikasjonene som benyttes av sluttbrukere for å kunne kommunisere med back-end (begrepet blir forklart i neste seksjon). I vårt prosjekt regner vi både smartphone-applikasjonen og dashboardet som front-end-systemer.

For å kunne tilby en brukervennlig og lett forståelig mobil- og webapplikasjon var det viktig for oss å bruke velkjente rammeverk som tilbyr en høy grad av grafisk tilpasning. Som vi vil komme tilbake til var det også et krav om at mobilapplikasjonen skulle være kompatibel med både Google Android og Apple iOS, noe som gjenspeiles i valg av front-end teknologi.

Tabellen nedenfor beskriver teknologier brukt i forbindelse med utvikling av grensesnittet til mobilapplikasjonen og administrasjonsgrensesnittet (dashboard).



Vi har brukt Ionic til å utvikle smartphone-applikasjonen, bootstrap og CSHTML for det visuelle uttrykket i dashboard-løsningen, og Puzzel for å integrere chat-funksjonalitet fra applikasjonen til servicedesken hos oppdragsgiver. JSON-formatet brukes til å definere all data som sendes over internett.

Ionic har åpen kildekode og er tilgjengelig under MIT-lisensen. Det er svært enkelt å integrere Ionic rammeverket mot vårt valg av ASP.NET back-end teknologi (se neste seksjon). Ettersom Ionic bygger på HTML5 og Angular, som er nettbaserte teknologier finnes det integrert funksjonalitet for å kommunisere med webtjenester. Samtidig har Ionic støtte for å tilpasse sitt grensesnitt basert på typen av mobil enhet (Android, iOS, etc.) det kjøres på, noe som er svært gunstig for brukervennlighet. Lignende løsninger som for eksempel PhoneGap tilbyr ikke denne funksjonaliteten, og dermed har vi valgt å bruke Ionic.

Navn	Bruksområde	Beskrivelse
Ionic <sup>4</sup>	Apputviklings-rammeverk	Ionic er et rammeverk bygget på Cordova fra Apache, som muliggjør utvikling til forskjellige operativsystemer med samme kodebase
Bootstrap <sup>5</sup>	Styling av dashboard	Bootstrap er et bibliotek bygget på Javascript og CSS som tilbyr en rekke GUI-elementer for nettsteder
jQuery <sup>6</sup>	Dynamisk nettside	jQuery forenkler funksjoner som brukes ofte i javascript
Modernizr <sup>7</sup>	Oversikt over versjon til biblioteker	Modernizr er et verktøy som finner hvilke versjoner av biblioteker en nettleser er kompatibel med

<sup>4</sup> Dokumentasjon for Ionic er tilgjengelig på nettsiden, <https://ionicframework.com/>

<sup>5</sup> Dokumentasjon for Bootstrap er tilgjengelig på <https://getbootstrap.com/>

<sup>6</sup> Dokumentasjon for jQuery er tilgjengelig på <https://jquery.com/>

<sup>7</sup> Dokumentasjon for Modernizr er tilgjengelig på <https://modernizr.com/docs>

CSHTML <sup>8</sup>	Presentasjon av dashboard	CSHTML er en videreutvikling av markup-språket HTML som tillater noen C#-funksjonalitet i webdokumentet
Puzzel Chat <sup>9</sup>	Chat mot servicedesk	Puzzel Chat er et javascript-tillegg som muliggjør chat i realtid med konsulenter på servicedesken
JSON <sup>10</sup>	Datautveksling	JSON er et simpelt og standardisert format for utveksling av data

## 4.2 Back-end-teknologier

Som forklart i forrige seksjon er front-end det bruker ser og benytter for å kommunisere med back-end. På samme måte kan back-end (eller data-access-laget) ses på som det som er skjult fra vanlige brukere når de benytter applikasjonene. All kode som kjøres på serveren, unntatt det grafiske grensesnittet til dashboardet, definerer vi som back-end-systemer i vårt prosjekt.

Vi har brukt ASP.NET MVC for dashboard-webapplikasjonen, Entity Framework for modellering av databaseobjekter og spørringer, Microsoft SQL for kall til databasen for saksbehandlingssystemet og Windows Server for publisering og hosting av løsningene til nett. For å muliggjøre og understøtte autentisering benyttet vi rammeverket OWIN. I likhet med vår front-end bruker også vår back-end JSON-formatering når data skal sendes over nett.

Utvikling av back-end ble gjort ved hjelp av anerkjent programvare utviklet og levert av Microsoft. Ved å benytte teknologi fra Microsoft sørger vi for å oppfylle de allerede fleste kriteriene som ble nevnt i starten av kapitlet.

---

<sup>8</sup> CSHTML er en del av ASP.NET. Mer informasjon om CSHTML er tilgjengelig på <https://www.asp.net/>

<sup>9</sup> Mer informasjon om Puzzel Chat er tilgjengelig på <https://www.puzzel.com/uk/press-releases/webchat/>

<sup>10</sup> Dokumentasjon for JSON er tilgjengelig på <https://www.json.org/>

Ytterligere begrunnelser for valget av Microsoft produkter er at 99X er en Microsoft Partner (de har allerede lisenser og oppsett for serverløsningene), samt at grupped medlemmene fra før av har erfaring med mange av disse verktøyene via utdanningen.

Neste tabell beskriver verktøy brukt for utvikling og drift av back-end-funksjonalitet for administrasjonsgrensesnittet (dashboard).

Navn	Bruksområde	Beskrivelse
ASP.NET MVC <sup>11</sup>	Dashboard-rammeverk	ASP.NET MVC er et rammeverk for utvikling av webapplikasjoner
Entity Framework <sup>12</sup>	Modellering av databasentiteter	Entity Framework er et rammeverk som muliggjør lagring av kodeobjekter i databaser via mapping
Microsoft SQL <sup>13</sup>	Dashboard-database og spørringer	Microsoft SQL brukes for å lagre data og gjennomføre databasespørringer
Windows Server <sup>14</sup>	Operativsystem	Windows Server er et operativsystem designet for en rekke server-oppgaver
OWIN <sup>15</sup>	Middleware for ASP.NET-applikasjoner	OWIN (Open Web Interface for .NET-applications) er en standard som forenkler kommunikasjon mellom server og applikasjon
JSON	Datautveksling	JSON er et standardisert format for utveksling av data

<sup>11</sup> Dokumentasjon for ASP.NET MVC er tilgjengelig på nettsiden, <https://www.asp.net/mvc>

<sup>12</sup> Dokumentasjon for Entity Framework er tilgjengelig på <https://docs.microsoft.com/en-us/ef/>

<sup>13</sup> Mer informasjon om Microsoft SQL er tilgjengelig på <https://www.microsoft.com/en-us/sql-server>

<sup>14</sup> Mer informasjon om Windows Server er tilgjengelig på <https://www.microsoft.com/en-us/cloud-platform/windows-server>

<sup>15</sup> Dokumentasjon for OWIN er tilgjengelig på <http://owin.org/>

### 4.3 Utviklingsverktøy

Det er svært viktig å ha intuitive, intelligente og effektive utviklingsverktøy i en prosess hvor man skal produsere en større mengde med kode. Vi valgte derfor å benytte utviklingsmiljøet Visual Studio 2017 fra Microsoft for dashboardet og APIet, da denne har god synergi med teknologien vi benytter i back- og front-end av tjenesten. Samtidig hadde vi ikke noe valg, da ASP.NET-applikasjoner ikke kan utvikles i andre systemer, da det er et proprietært Microsoft-produkt. For refaktorering av kode (omstrukturering) benyttet vi ReSharper, ettersom de integrerte verktøyene i Visual Studio 2017 er begrensede på området.

Node Packet Manager ble brukt til installasjon, oppdateringer av og oversikt over tillegg til Ionic. Mozilla Firefox ble brukt for å teste under utviklingen av mobilappen. Fordelen med å bruke en nettleser til testing er at vi under utvikling ikke trengte å kjøre applikasjonen i en telefon-emulator, noe som er langt mindre tidseffektivt enn å bruke en nettleser.

For utvikling av koden til smartphone-applikasjonen benyttet vi kodeeditoren Visual Studio Code. Programmet er en lettvekts-variant av Visual Studio som kutter ut mye av den tyngre funksjonaliteten i Visual Studio 2017 som vi ikke trengte til applikasjonsutviklingen, og da gjør editoren raskere og enklere å forholde seg til. Samtidig har den god støtte for verktøy som bruker kommandolinje istedenfor grafiske grensesnitt (Node Packet Manager og Ionic).

Tabellen nedenfor inneholder verktøyene vi har brukt for koding av applikasjonene.

Navn	Bruksområde	Beskrivelse
Visual Studio Code <sup>16</sup>	Kodeeditor	Kodeeditor med innebygget støtte for en rekke programmeringsspråk, samt Git og Node Packet Manager
Visual Studio 2017 <sup>17</sup>	Integrert utviklingsmiljø	Komplett utviklingsmiljø for en lang rekke forskjellige applikasjoner

---

<sup>16</sup> Dokumentasjon for Visual Studio Code er tilgjengelig på nettsiden, <https://code.visualstudio.com/>

<sup>17</sup> Dokumentasjon for Visual Studio er tilgjengelig på <https://www.visualstudio.com/>

ReSharper <sup>18</sup>	Koderefaktorering	Plugin til Visual Studio 2017 som blant annet forenkler spesifisering av namespaces og endring av navn i kode
Node Packet Manager <sup>19</sup>	Håndtering av pakker	NPM er et verktøy for å automatisere installasjon av tilleggsverktøy (brukt til å installere plugins til Ionic)
Mozilla Firefox <sup>20</sup>	Debugging	Mozilla Firefox er en nettleser med åpen kildekode og integrerte systemer for debugging av webkode

## 4.4 Støttesystemer

Vi benyttet en rekke støttesystemer som forenklet organisering og prosjektstyring under hele utviklingsprosessen. Det var viktig for oss å til enhver tid ha oversikt over kodebasen, oppgaver og dokumentasjonen gjennom bruk av anerkjente verktøy og systemer. I tillegg var det nødvendig å benytte systemer for å administrere og sette opp både testmiljø og produksjonsmiljø.

Ettersom systemene hos 99X ligger bak flere sikkerhetssystemer og brannmurer måtte vi bruke Citrix Receiver og Remote Desktop for å få tilgang til serverne vi skulle teste og publisere løsningen på. For planlegging og styring av prosjektet brukte vi Trello sammen med veileder hos OsloMet og Asana internt i gruppen for å holde oversikt over oppgaver.

For oversikt over deling av og historikk i kodebasen brukte vi Git og GitHub, samtidig som GitHub Pages ble brukt for å publisere nettsiden knyttet til bachelorgruppen. Dokumentasjon av prosjektet ble primært gjort i Google Docs og Slack ble brukt til diskusjoner og deling av kode.

---

<sup>18</sup> Dokumentasjon for ReSharper er tilgjengelig på nettsiden, <https://www.jetbrains.com/resharper/>

<sup>19</sup> Dokumentasjon for Node Packet Manager er tilgjengelig på <https://www.npmjs.com/>

<sup>20</sup> Mer informasjon om Firefox er tilgjengelig på <https://developer.mozilla.org/en-US/docs/Tools/Debugger>

For å muliggjøre sending av push-meldinger til mobile enheter brukte vi Google sin Firebase-tjeneste. Programmet Postman ble brukt til å simulere og teste programkall til vårt dashboard-API.

Tabellen nedenfor inneholder hovedstøttesystemene vi har brukt under utviklingen av applikasjonene.

Navn	Bruksområde	Beskrivelse
Citrix Receiver <sup>21</sup>	Tilgang til virtuell desktop	Citrix Receiver er en applikasjon som gir tilgang til virtuelle desktops over nett
Remote Desktop <sup>22</sup>	Tilgang til virtuelle maskiner fra Citrix	Remote Desktop muliggjør fjernstyring av Windows-maskiner over et nettverk
GitHub <sup>23</sup>	Distribuert versjonskontroll	GitHub er en hosted Git repository, dvs. et nettsted som muliggjør deling og sikkerhetskopi av kode med versjonskontroll
GitHub Pages <sup>24</sup>	Hosting av nettside	GitHub Pages muliggjør publisering av et repository som et nettsted
Asana <sup>25</sup>	Kanban-board og bugtracking	Asana muliggjør opprettelse, oversikt og tildeling av oppgaver
Trello <sup>26</sup>	Oversikt over milepæloppgaver i samarbeid med veileder	Trello muliggjør opprettelse, oversikt og tildeling av oppgaver

<sup>21</sup> Dokumentasjon for Citrix Receiver er tilgjengelig på nettsiden, <https://www.citrix.com/>

<sup>22</sup> Dokumentasjon for Remote Desktop er tilgjengelig på <https://www.citrix.com/>

<sup>23</sup> Dokumentasjon for GitHub er tilgjengelig på <https://github.com/>

<sup>24</sup> Dokumentasjon for GitHub Pages er tilgjengelig på <https://pages.github.com/>

<sup>25</sup> Dokumentasjon for Asana er tilgjengelig på <https://asana.com/>

<sup>26</sup> Dokumentasjon for Trello er tilgjengelig på <https://trello.com/>

Google Docs <sup>27</sup>	All skriftlig dokumentasjon	Google Docs. er en samling av verktøy for dokumentasjon
Slack <sup>28</sup>	Intern chat for gruppen (deling av kode-eksempler, informasjon fra Asana, etc.)	Slack er en chat-applikasjon med mulighet for integrering med andre tjenester
Firebase <sup>29</sup>	Utsendelse av push-notifikasjoner til telefoner	Firebase er en tjenestetilbyder for en rekke online-tjenester
Postman <sup>30</sup>	Testing av webtjenester (API)	Postman er et verktøy som brukes til å sende forskjellige typer meldinger til en bestemt nettsadresse, for å kunne teste funksjonaliteten til en tjeneste / API.
Telnet <sup>31</sup>	Test av nettverk	Telnet har et begrenset antall funksjoner for å teste nettverksoppsett og porter

## 4.5 Utviklingsmiljø

Med miljø menes settingen hvor et produkt har blitt utviklet, testet og publisert. Selv om det jobbes med samme produkt har de forskjellige miljøene ulike egenskaper. Utviklings- og testmiljøet er de miljøene som utviklerne og testerne jobber med, mens produksjonsmiljøet er det miljøet sluttbrukeren har tilgang til.

---

<sup>27</sup> Dokumentasjon for Google Docs er tilgjengelig på nettsiden, <https://www.google.com/docs/about/>

<sup>28</sup> Dokumentasjon for Slack er tilgjengelig på <https://slack.com/>

<sup>29</sup> Dokumentasjon for Firebase er tilgjengelig på <https://firebase.google.com/>

<sup>30</sup> Dokumentasjon for Postman er tilgjengelig på <https://www.getpostman.com/>

<sup>31</sup> Dokumentasjon for Telnet er tilgjengelig på <https://docs.microsoft.com/>

I de neste seksjonene fortelles det om hvilke miljøer vi har brukt i utviklingen av smartphone-applikasjonen og dashboard-løsningen. Vi diskuterer også hvordan vi har satt opp miljøene for testing og publisering av APIet og dashboard-siden, herunder portåpninger og sikkerhet, serverkonfigurasjoner, nettverksoppsett, med mere.

## Dashboard og API

Dashboardet og APIet ble utviklet med Microsoft sin utviklingsplattform Visual Studio. Visual Studio er et "Integrated Development Environment", også kalt en IDE. En IDE er et program som gjør det mulig å skrive kode, kjøre koden, teste den, feilsøke den, og mye mer.

Dashboardet er en nettside som publiseres via en webserver. APIet (Application Programming Interface) er en tjeneste som håndterer instruksjoner til databasen, som publiseres på samme server som dashboardet. Mens vi utviklet dashboardet var vi også nødt til å bruke en lokal server for å se hvordan nettsiden så ut, noe som er innebygget i Visual Studio. Det vil si at nettsiden og serveren ble kjørt på samme maskin, for å simulere hvordan nettsiden vil fungere for sluttbrukeren.

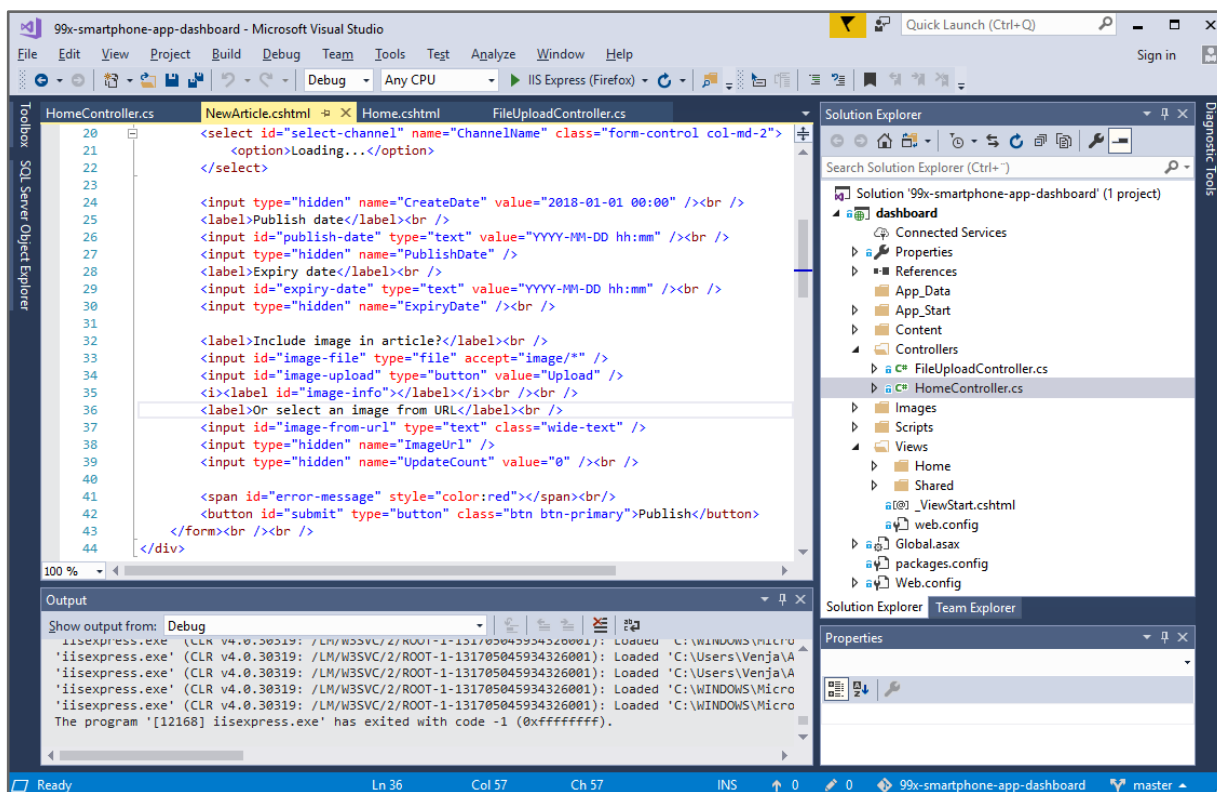


Fig. 4.5.1 - Skjerm bilde fra Visual Studio



## Smartphone app

For å utvikle applikasjonen brukte vi Visual Studio Code. Som nevnt i kapittel 3.3 (Utviklingsverktøy) er Visual Studio Code en lettere kode-editor med mindre funksjonalitet enn Visual Studio 2017. Vi trengte en rask kode-editor med støtte for verktøy som bruker kommandolinjen (Ionic og Node Packet Manager), samt versjonshåndteringssystemet Git. Visual Studio Code oppfylte disse kravene.

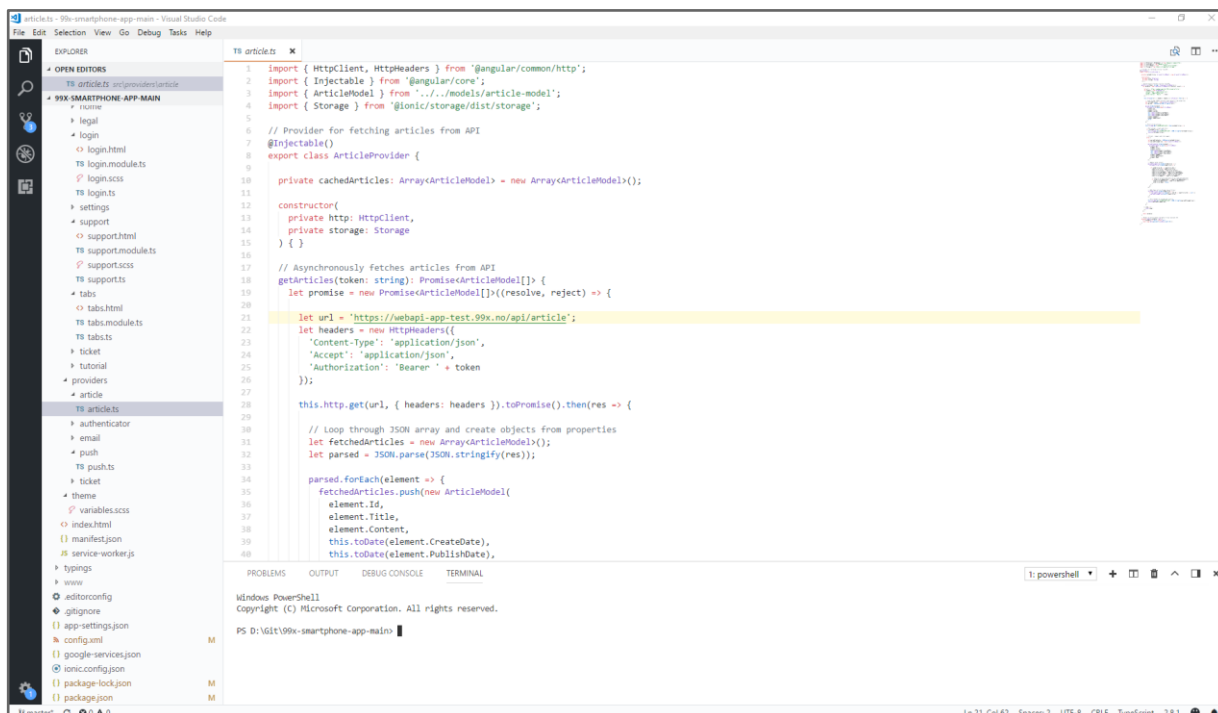
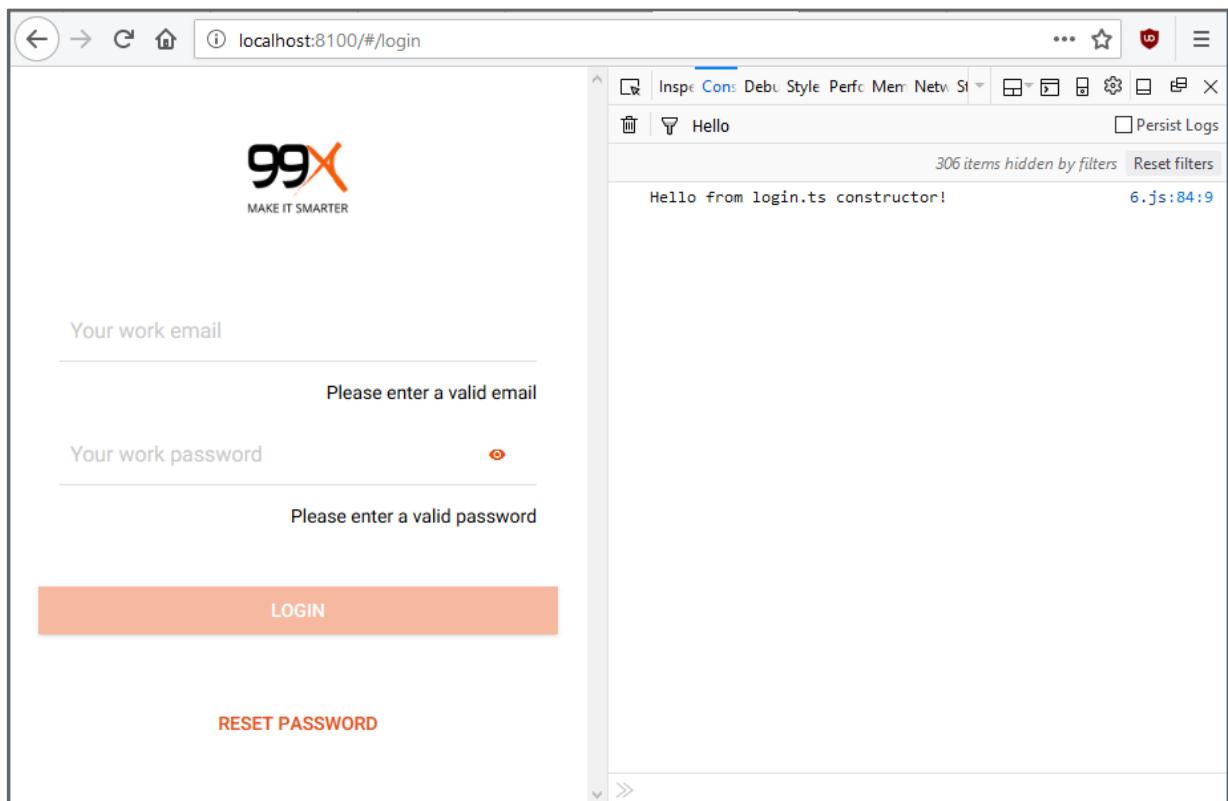


Fig. 4.5.2 - Skjerm bilde fra Visual Studio Code

For debugging brukte vi Ionic sitt kommandolinjegransesnitt via Visual Studio Code for å editere og kjøre koden i realtid. Ved kommandoen `ionic serve` blir applikasjonen publisert lokalt på maskinen som en nettside, og man kan da aksessere og teste denne via en nettleser. Når endringer i koden utføres og lagres transpileres<sup>32</sup> koden automatisk, og det publiserte “nettstedet” blir oppdatert.

<sup>32</sup> Transpilering innebærer at kode blir omvandlet til et annet språk. I vårt tilfelle innebærer dette at Typescript-kode blir oversatt til Javascript-kode (som en nettleser kan tolke).

Denne prosessen gjør debugging og utvikling veldig effektiv, da endringer i koden blir reflektert i realtid i nettleseren. Bildet nedenfor fra Firefox illustrerer dette. Her kan man se at applikasjonen kjører som et nettsted, der konsollutskrifter fra koden blir vist i nettleseren sin debug-konsoll. En annen positiv side er at det er enkelt å analysere nett-trafikk, da Firefox (og andre nettlesere) ofte har konsoller der man kan monitorere og se innholdet i nettverksoverføringer.



*Fig. 4.5.3 - Test av applikasjonen i Firefox, med synlig konsollutskrift fra programkoden*

For å teste at all funksjonalitet også fungerer på Android-enheter har vi i tillegg brukt Google sin offisielle Android-emulator. Denne er også integrert i Ionics kommandolinje og kan kalles med `ionic cordova run android`. Kommandoen starter en emulator, installerer den siste versjonen av applikasjonen på denne og simulerer bruket på en virkelig Android-telefon. Svakheten med dette er at vi ikke får tilgang til konsoll-output som i det forrige eksemplet (via nettleser), samt at det ikke er like raskt å gjøre endringer under kjøring.

## 4.6 Test- og produksjonsmiljø

Test- og produksjonsmiljøene for APIet og dashboardet ble satt opp som tilnærmet identiske speilinger av hverandre. Dette var et bevisst valg, da vi på denne måten hadde mulighet til å teste applikasjonene i testmiljøet som om de hadde vært de ferdige produktene. Etter fullført testing kunne vi då flytte over løsningen til produksjonsmiljøet, med minimale endringer.

Smartphone-applikasjonen ble utelukkende utviklet lokalt på gruppe-medlemmenes private maskiner, og dermed er ikke denne seksjonen aktuell for dette.

Illustrasjonen nedenfor viser oppsettet brukt i test- og produksjonsmiljøene. Netscaler<sup>33</sup> ruter trafikk til riktig server basert på nettadresse (<https://webapi-app-test.99x.no/api> for test eller <https://webapi-app.99x.no/api> for produksjon):

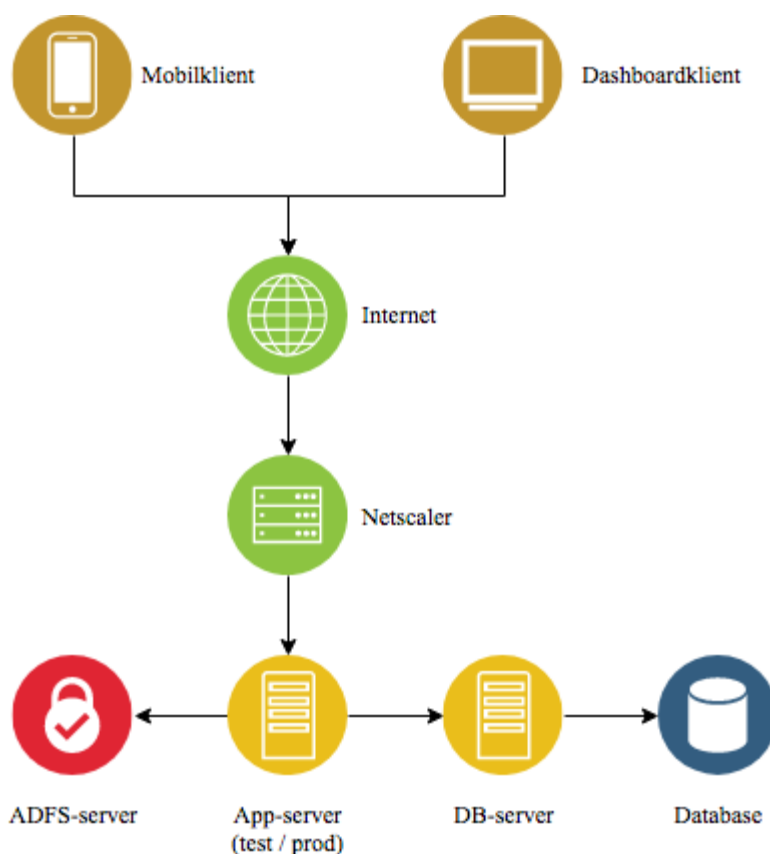


Fig. 4.6.1 - Diagram som illustrerer systemoppsettet brukt i test- og produksjonsmiljøene

<sup>33</sup> Netscaler er en tjeneste som blant annet kan styre ekstern trafikk til riktig sted i et internt nettverk

Ettersom databasen for vårt dashboard ikke eksisterte fra før, hadde vi ikke behov for en dedikert test-database. I stedet brukte vi samme database for test og produksjon, og fjernet så all testdata fra databasen ved migrering. For den klonede databasen til saksbehandlingssystemet trengte vi heller ikke noen test-kopi, da vi ikke utfører noen endringer mot denne, men kun bruker den til å hente og lese data.

Vedrørende sertifikater bruker både test- og produksjonsmiljøene domene-sertifikatet fra 99X for å kryptere trafikk.

## 5 Teknisk produktdokumentasjon

Gjennom fem måneder har gruppen utviklet en applikasjon, med tilhørende API og dashboard, som skal gjøre det enklere for arbeidsgiver og deres kunder å holde oversikt over support-saker, viktige nyheter og push-varsler.

### 5.1 Kravspesifikasjon vs. levert produkt

I dette delkapitlet skal vi snakke om hvordan det endelige produktet vi leverte differensierer fra produktet vi hadde planlagt å lage ut i fra kravspesifikasjonen. Vi snakker også om hvorfor vi ikke kunne implementere visse komponenter.

#### Smartphone-applikasjon

Ettersom vi utarbeidet listene over must- og nice-to-haves sammen med oppdragsgiver hadde vi muligheten til å sette opp funksjonalitetsmål. Målene anså vi som oppnåelige med tanke på den tiden vi hadde til rådighet og kunnskapene internt i gruppen. Det som ble lagt i listen for must-haves fokuserte vi på å få ferdig raskest mulig. Følgende oppgaver vedrørende smartphone-applikasjonen ble fullført:

- Real-time-chat
- Mulighet å ringe til servicedesken fra appen
- Innsending av nye saker til servicedesken
- Tilbakestilling av passord
- Statusoversikt for aktive saker (og avsluttede)
- Korrespondanse mot saksbehandlingssystemet
- Utsendelse av nyheter

Etterhvert innså vi at vi ikke skulle klare å levere alle must-haves. Vi besluttet da sammen med kontaktpersonen hos oppdragsgiver at noen av funksjonaliteten måtte prioriteres ned til nice-to-haves, for å heller sikre at et brukbart produkt ble levert. Til slutt rakk vi heller ikke å implementere disse, da vi måtte fokusere på å fullføre denne rapporten. Denne funksjonaliteten var:

- Rapporter for IT-brukere hos kunder
- Dokument-uploads (veiledninger, etc.)

Hvis vi ble ferdige med alle must-haves var vår intensjon å bruke den resterende utviklingstiden på å implementere nice-to-haves. Dessverre tok noen av oppgavene som nevnt lenger tid enn først forventet, og vi måtte da ofre noe av denne funksjonaliteten. Følgende punkter ble *ikke* levert i applikasjonen:

- Bestillings skjemaer
- Endring av tekststørrelse
- Endring av språk

## Dashboard

### **Notifikasjoner / push-meldinger**

Da vi definerte kravspesifikasjonen relatert til sending av push-meldinger hadde vi ikke satt oss spesielt mye inn i hva man har mulighet til å gjøre, og hvor begrensningene ligger. I kravspesifikasjonen skrev vi at en teknisk konsulent blant annet skal kunne se, sende, oppdatere og fjerne push-meldinger (se 3.1 - Kravspesifikasjon). Senere, da vi skulle implementere løsningen, kom vi frem til at en push-melding mer eller mindre fungerer som en SMS. Vi hadde derfor ikke mulighet til å for eksempel oppdatere eller fjerne push-meldinger.

I vår endelige løsning har en teknisk konsulent mulighet til å sende push-meldinger, begrense mottakere av push-meldinger basert på grupper, og se historikk over sendte push-meldinger.

Punktene under er kravspesifikasjonene i henhold til push-meldinger som vi ikke hadde mulighet til å implementere, eller ikke rakk å implementere:

- Teknisk konsulent skal kunne oppdatere push-meldinger
- Teknisk konsulent skal kunne fjerne push-meldinger
- Teknisk konsulent skal kunne se statistikk over push-meldinger
- Teknisk konsulent skal kunne benytte ferdige maler for push-meldinger

### **Definere maler**

I kravspesifikasjonen har vi definert at administrator skal kunne konfigurere maler for notifikasjoner, og legge til og fjerne standardbilder. En mal for notifikasjoner kunne vært en fotnote på slutten av meldingen, som for eksempel en hilsen fra 99X. Standardbilder hadde også vært en del av malen. Det hadde vært et sett med bilder som kunne vært inkludert i artikler, for eksempel 99X sin logo. Vi definerte disse funksjonalitetene for å gjøre det lettere for brukerne av dashboardet å sende ut nyheter. Disse kravene hadde vi ikke prioritert veldig høyt, da det ikke hadde vært nødvendig å ha det med for å få et ferdig produkt.

## 5.2 Struktur og kodeeksempler

I dette underkapittelet vil vi gå inn på databasestrukturen og hvordan denne samspiller med applikasjonen. Det vil også være ER-diagrammer og kodeeksempler som vil utdype og forklare hvordan systemet henger sammen.

### Databasestrukturer

ER-diagrammet (Entity Relationship) nedenfor viser hvilke objekter vi lagrer i vår database, samt hvilke relasjoner entitetene har til hverandre. For å oppnå funksjonaliteten i henhold til kravspesifikasjon trenger vi ikke en avansert database, så vi endte opp med kun tre entiteter:

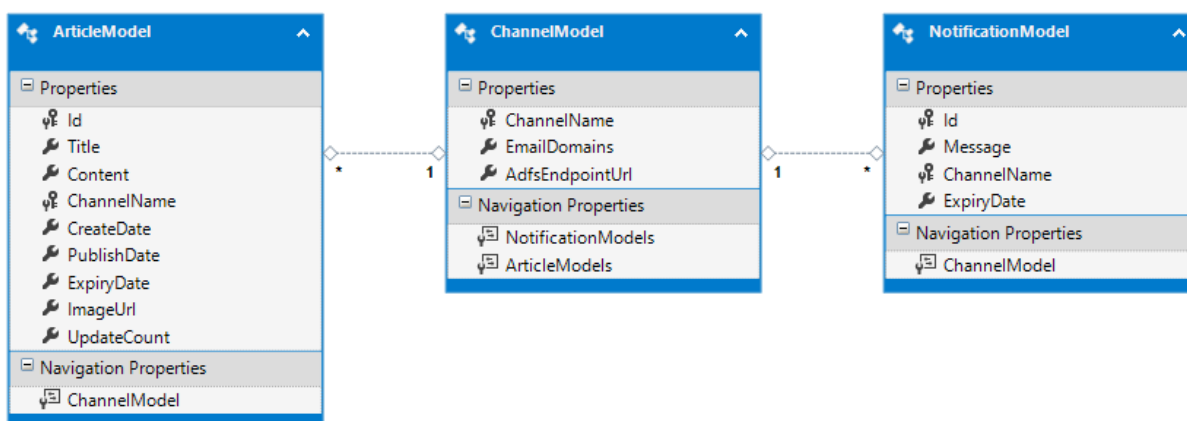


Fig. 5.2.1 - Entity-Relationship-diagram for APIet

**ArticleModel** representerer artikler opprettet i dashboardet og sendt ut til mobilappen. Disse inneholder for eksempel artikkeltekst, tittel og datoer for tilgjengeliggjøring/utløpstid.

**ChannelModel** er en tabell der vi lagrer informasjon om kundene sine kanaler. Dette blir brukt for å sikre at push-meldinger og artikler blir sendt ut til riktige mottakere. Samtidig brukes denne entiteten til å bestemme hvor sluttbrukeren skal rutes ved pålogging (dette blir forklart under seksjonen Autentiseringsmekanismer).

**NotificationModel** inneholder historikk over sendte push-varsler, som innhold, mottakere (Channel) og hvor lenge meldingen skal være mulig å laste ned.



For autentisering bruker vi en dedikert database fordi det er ønskelig å separere denne dataen fra ting som ikke er relatert til autentisering (for eksempler artikler). Det er viktig at sikkerhetsdata håndteres på en forsvarlig måte og det gir mening og holde dette isolert fra annen kode og data som kan vanskeliggjøre oversikt.

Vi valgte å bruke identitetssystemet som allerede eksisterer som en del av ASP.NET og Entity Framework (ASP.NET Identity) og bruker da de grunnleggende database-entitetene som automatisk blir opprettet. Fordelene med dette er at det ble enklere å integrere med token-flyten nevnt i neste seksjon. Funksjonaliteten vi ikke bruker ( for eksempel to-faktor-autentisering) lar vi ligge som en del av modellene, men gjør dem ikke tilgjengelige for sluttbrukerne eller systemet:

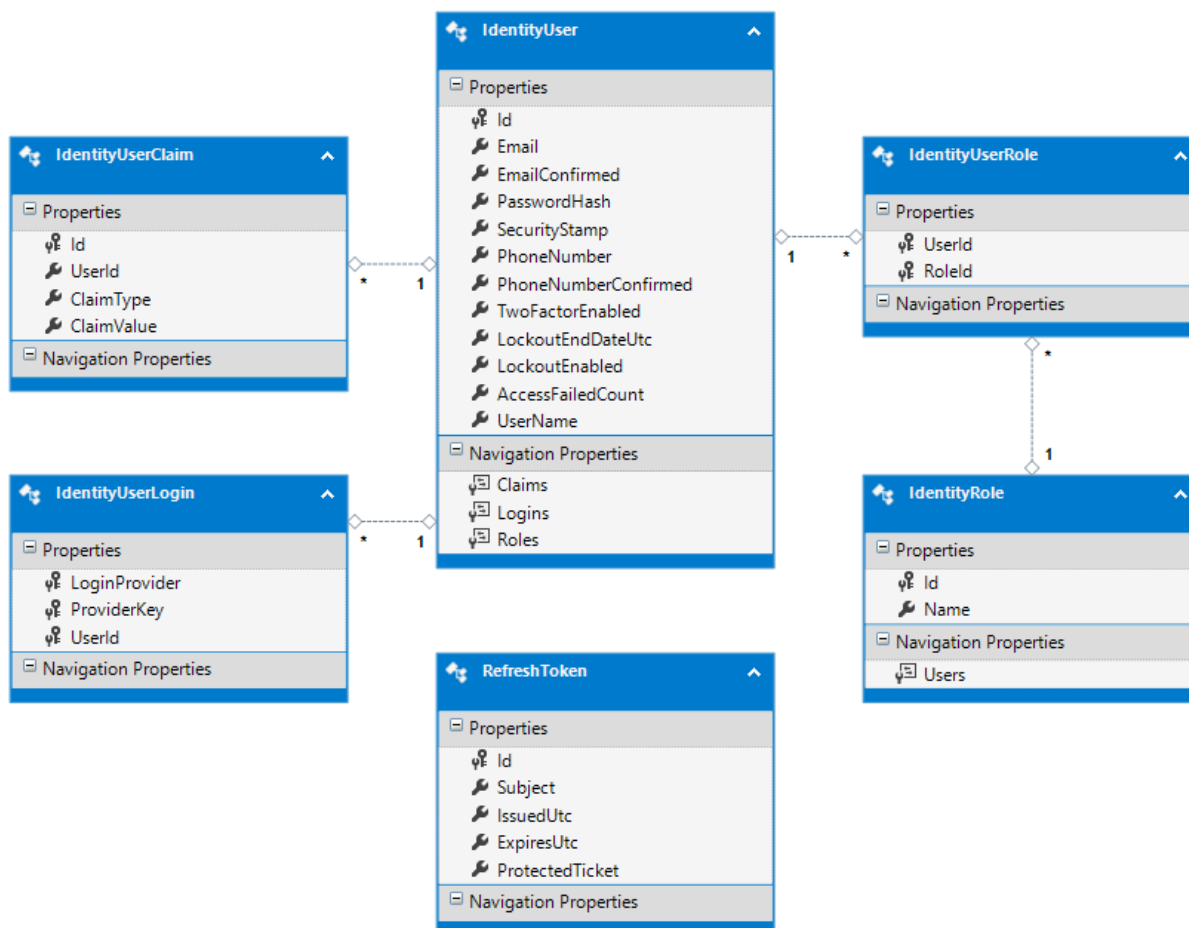


Fig. 5.2.2 - Entity-relationship-diagram for autoriseringsystemet

**RefreshToken** er en egen frittstående modell som vi bruker til å lagre informasjon om hvem det er som eier en spesifikk refresh-token og hvor lenge denne er gyldig. Det kan aldri eksistere mer enn én RefreshToken for hver brukeridentitet.

Alle andre entiteter i diagrammet er standardmodeller fra Entity Framework, og vi bruker dem for å lagre informasjon om brukere av systemet. Hvis en bruker aldri har vært pålogget tidligere opprettes en identitet for brukeren. Vi lagrer kun informasjon om påloggingsforsøk, rettigheter til ressurser og brukernavn. Sluttbrukerens passord blir ikke lagret, men blir lagt til som en randomisert tekststreng. Ansvar for brukerens faktiske passord ligger hos enhver kunde sin ADFS-server (se neste seksjon).

Nedenfor er et kodeeksempel som illustrerer det ovennevnte konseptet. Koden sjekker først om bruker og passord stemmer med kundens system. Hvis bruker er autorisert opprettes en separat bruker for tokenhåndtering og logging av påloggingsforsøk, hvis brukeren ikke allerede har en slik bruker fra før. Koden oppdaterer også mislykkede påloggingsforsøk:

```
// Sjekk hvis bruker eksisterer og at passordet samsvarer med kunden sitt system
if (ADFS.GetADFS-Token(context.UserName, context.Password) == null)
{
    // Hvis vi ikke fikk et svar betyr dette at passordet er feil
    // Antallet mislykkede påloggingsforsøk økes og bruker sperres etter fem feil
    using (var repo = new AuthRepository())
        repo.IncreaseFailedLoginCount(context.UserName);

    // Stopp videre eksekvering
    return;
}

// Autorisering ok - opprett bruker i database slik at tokens kan styres
using (var repo = new AuthRepository())
{
    var user = new UserModel
    {
        UserName = context.UserName,
        Password = Helpers.RandomString(64)
    };
    await repo.RegisterUser(user);
}

// Her sørges det for at feil blir nullstilt ved godkjent pålogging
using (var repo = new AuthRepository())
    repo.ResetFailedLoginCount(context.UserName);
```

## 5.3 Mekanismer for sikkerhet og autentisering

Det var viktig for oss å levere et sikkert produkt til oppdragsgiver, da applikasjonen skal benyttes av ansatte i flere roller fra flere forskjellige firmaer. Sensitiv data som for eksempel passord, brukernavn og informasjon vedrørende saker må håndteres og utveksles på en fornuftig måte mellom brukere og servere. Samtidig var det viktig at tiltakene for sikkerhet ikke gjorde applikasjonen mindre brukervennlig. For dette har vi tatt i bruk en rekke teknikker.

### Tokens

En token kan sees på som en elektronisk nøkkel som inneholder data nødvendig for å få tilgang til en sikret ressurs, og dette begrepet vil bli flittig brukt i denne seksjonen. Vi bruker to forskjellige typer av tokens i vårt prosjekt: access-tokens og refresh-tokens.

Access-tokens benyttes til å be om tilgang til en bestemt funksjon. Dette kan for eksempel være å få hentet en liste over artikler fra vår server. Uten en gyldig access-token får ikke brukeren tilgang til ressursen. Access-tokens har en kort levetid (kun 15 minutter) og blir deretter gjort inaktive og kan da ikke gjenbrukes. Skjermbildet nedenfor viser henting av en ressurs med hjelp av en access-token:

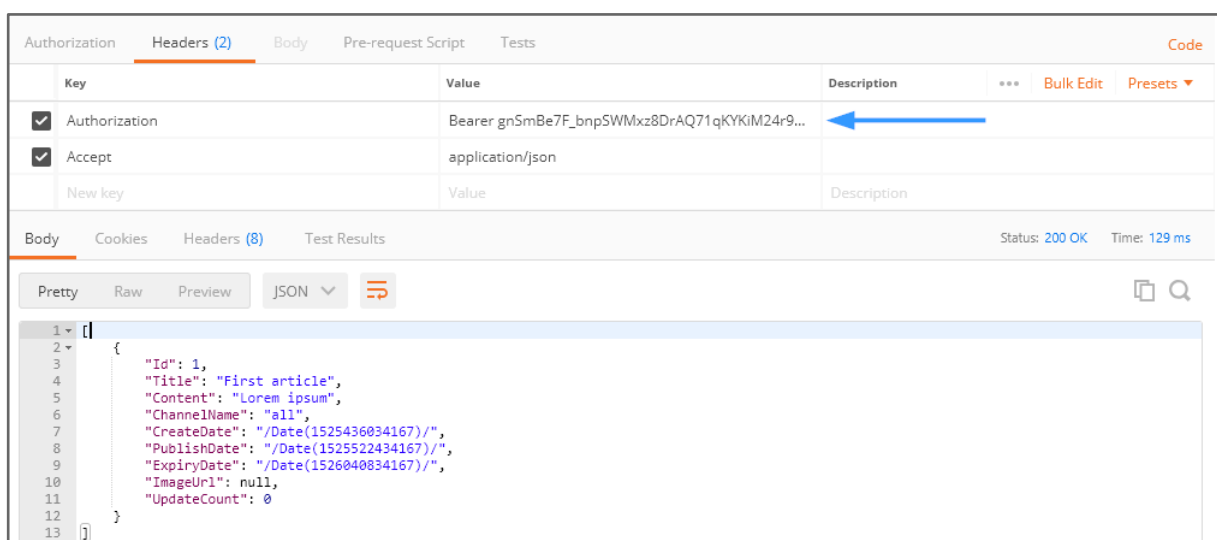


Fig. 5.3.1 - Figuren viser en simulert forespørsel til serveren via verktøyet Postman. Autoriseringstoken brukt i forespørselen er markert med blå pil

Refresh-tokens benyttes til å hente nye access-tokens og nye refresh-tokens. Refresh-tokens har en levetid på 30 dager. Hvis en bruker skal ha tilgang til en sikret ressurs må brukeren først spørre om å få en ny refresh-token. Hvis brukeren allerede har en refresh-token kan denne brukes til autentisering, og hvis ikke må brukeren autentisere seg med email og passord. Eksemplet nedenfor illustrerer førstegangspålogging (med email og passord):

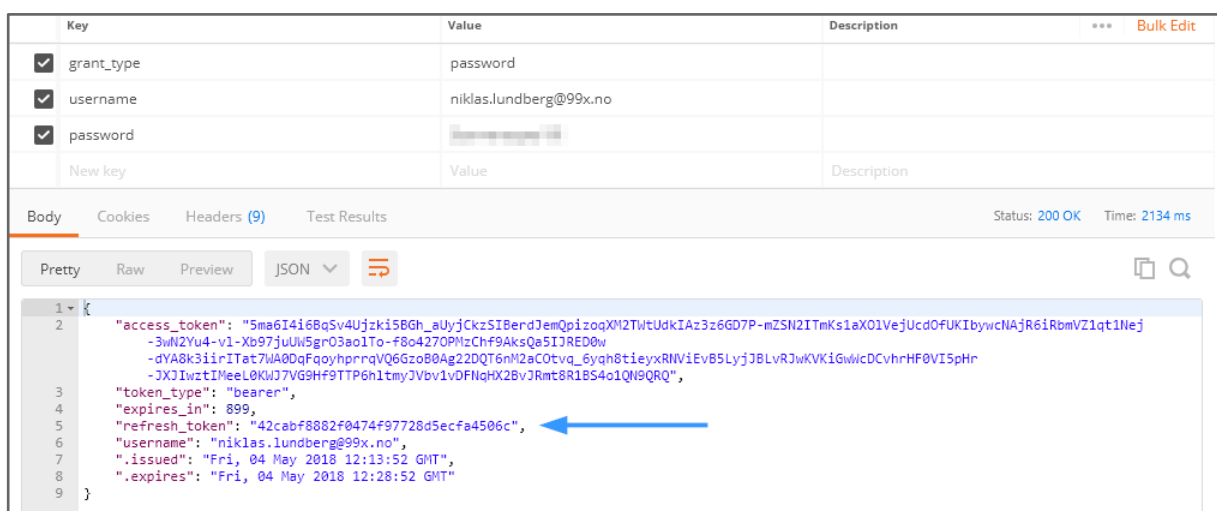


Fig. 5.3.2 - Simulert Postman-forespørsel som viser returen av en ny refresh-token ved pålogging med email og passord. Tokenet er markert med en blå pil

Forutsatt at refresh-tokenen eller påloggingskravene er godkjent blir brukeren tildelt en ny refresh-token samtidig som brukeren får en access-token som kan brukes til å hente ressurser. Neste bilde viser henting av en ressurs med kun refresh-token som autentisering:

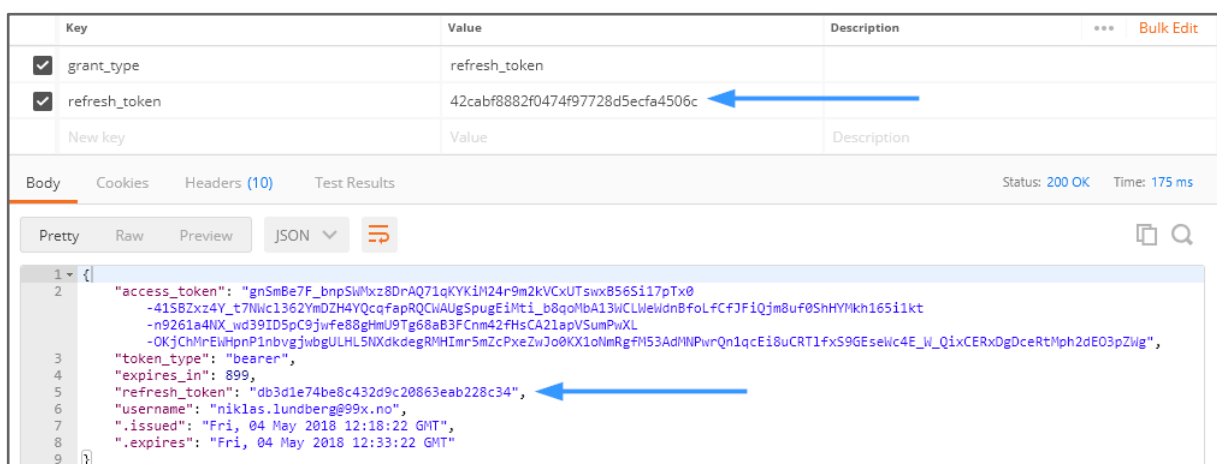


Fig. 5.3.3 - Figuren viser en spørring til tokenjenesten der refresh-token fra fig. 5.3.2 brukes til å hente en ny access- og refresh-token. Her blir ikke email eller passord sendt med i forespørselen

Følgende bilde viser returen fra serveren når en gyldig access-token ikke er sendt med i spørringen:

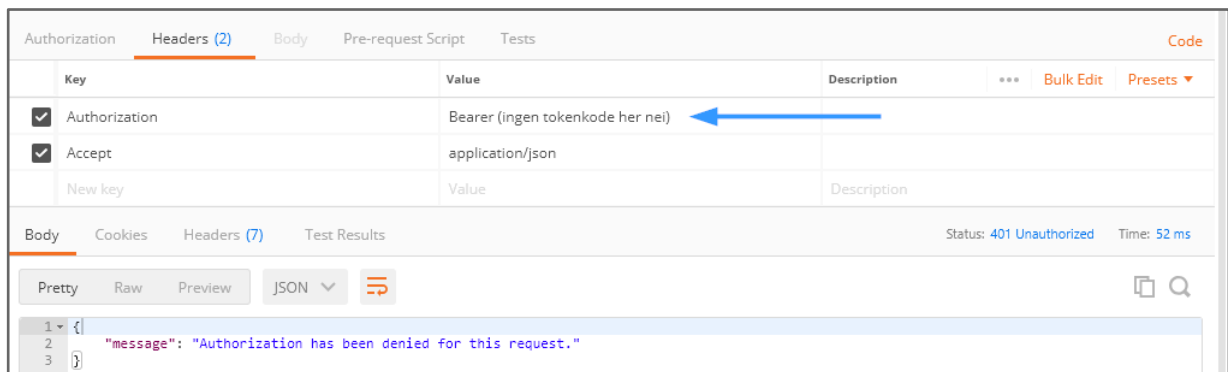


Fig. 5.3.4 - Bildet viser returen fra serveren når en gyldig token ikke blir sendt med i spørringen (blå pil). Svaret fra serveren er synlig i bunnen av bildet

Sekvensdiagrammet nedenfor illustrerer token-flyten, der alt 1 indikerer første pålogging og alt 2 viser bruken av refresh-tokens og henting av ressurser:

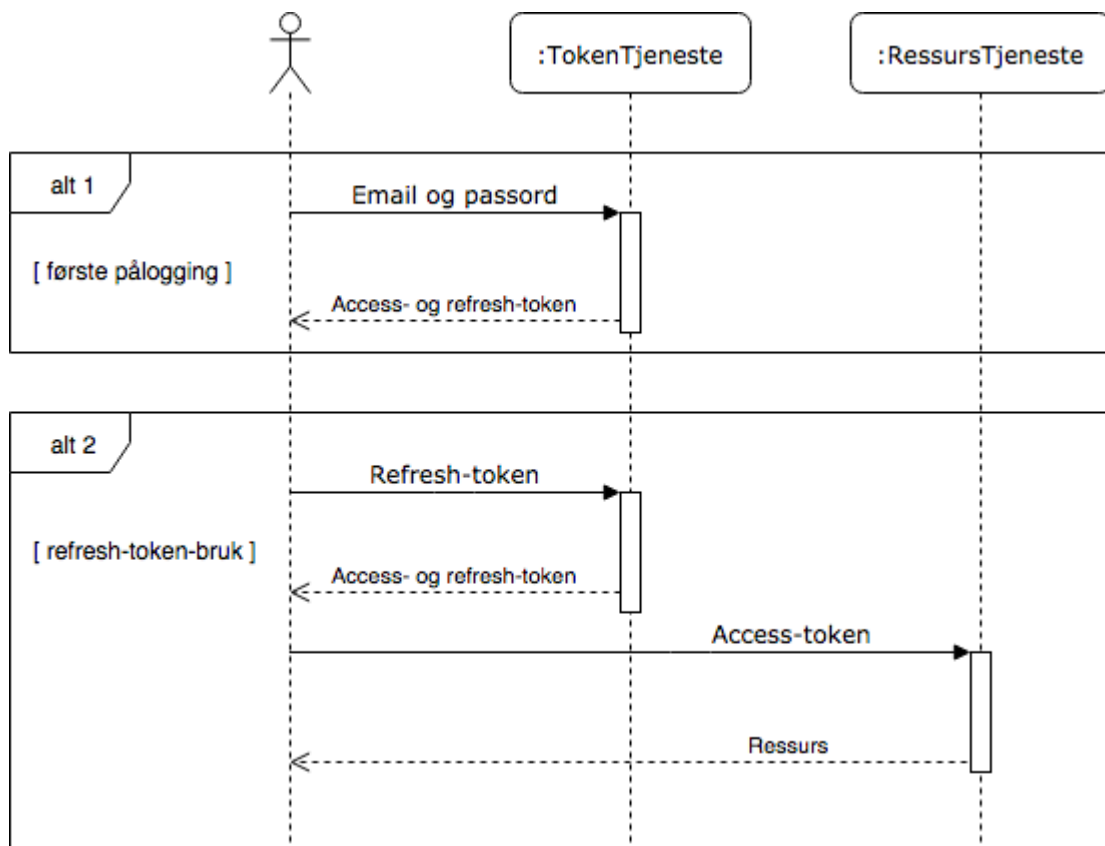


Fig. 5.3.5 - Sekvensdiagram som viser flyten for de to autoriseringsformene

Med denne løsningen trenger brukere kun å oppgi email og passord ved første pålogging (se alt 1 i diagrammet), og kan deretter i stedet sende med token-koden for å bekrefte sin identitet (alt 2). På denne måten unngår vi at brukernavn og passord sendes mer enn én gang - noe som potensielt kan være en sikkerhetsrisiko, samtidig som applikasjonen blir mer brukervennlig da vi ikke må vise en login-skjerm for hver start.

En ytterligere effekt denne løsningen gir er at vi enkelt kan sperre tilgangen for brukere gjennom å gjøre deres refresh-token inaktiv (brukere kan da ikke lenger innhente nye access-tokens).

## Bruk av firmakontoer

ADFS (Active Directory Federation Services) er et identitetssystem som håndterer påloggingsdata i Microsoft-miljøer. Samtlige kunder til oppdragsgiver benytter dette, og derfor integrerte vi dette i applikasjonen for å kunne autentisere brukere. På den måten oppnådde vi høy sikkerhet gjennom å bruke et robust system utviklet av Microsoft (fremfor å benytte en egen løsning). I tillegg muliggjør det for sluttbrukerne å logge på med samme bruker de benytter til å for eksempel å logge på sin jobb-PC eller email.

Vi møtte på noen utfordringer da vi skulle implementere ADFS-integrasjonen. For det første måtte vi vite hvor hver enkelt bruker skulle omdirigeres ved pålogging, slik at riktig server som kjører ADFS for det aktuelle firmaet blir kontaktet. For å løse dette la vi til en metode som utleder riktig server-adresse basert på domenet i oppgitt mailadresse (for eksempel `ola.nordmann@sio.no` eller `kari.nordmann@99x.no`).

Adressene til kundenes ADFS-servere lagres i den samme databasen som dashboardet benytter, slik at når oppdragsgiver får nye kunder kan nødvendig informasjon legges inn i databasen. Brukere hos det nye firmaet kan da enkelt få adgang til å logge på mobilapplikasjonen.

Dette kodeeksempel viser hvordan systemet bestemmer hvilken autoriseringsserver hos kunden det er som skal kontaktes. Først ekstraheres domenet fra emailadressen brukt til pålogging. Deretter søker koden etter en match for domenet i alle kundekanaler som finnes i databasen (hver kanal har en liste med domener knytte til denne). Hvis en match ble funnet returnerer metoden kanalen sin `AdfsEndpointUrl`, som er en adresse til den riktige påloggingsserveren hos kunden:

```
// Hjelpemetode for å finne riktig ADFS-server basert på email
private static string GetEndpointAddress(string userEmail)
{
    // Ekstraher domenen fra email
    var userDomain = new MailAddress(userEmail).Host.ToLower();
    var endpoint = string.Empty;

    // Hent channels (kunder) fra database og søk for en match basert på domene
    using (var repo = new ChannelRepository())
    {
        var channels = repo.Get();
        foreach (var channel in channels)
        {
            var domains = channel.EmailDomains.Split(',');
            if (domains.Any(d => d.ToLower().Equals(userDomain)))
            {
                endpoint = channel.AdfsEndpointUrl;
            }
        }
    }

    // Hvis match ble funnet returneres adressen til korrekt ADFS-server
    // Hvis ikke vil metoden returnere en tom streng (bruker kan da ikke logge på)
    return endpoint;
}
```

En annen utfordring var at dataen som kommer i retur fra ADFS ved vellykket autentisering er i form av SAML-tokens (Security Assertion Markup Language). Disse egner seg ikke til å sendes over internett da de inneholder svært mye data som vi ikke trenger[7]. For å sikre rask og effektiv utveksling av tokens valgte vi å kun benytte ADFS. Vi brukte ADFS kun for å se at vår server faktisk får en gyldig token i retur fra ADFS, for så å slette denne uten å sende den til sluttbrukeren. Når dette er gjort lages og sendes en egen mindre token, som kun har den informasjonen vi trenger. Hvis vi derimot ikke får en token fra ADFS ser vi på dette som et mislykket inloggingsforsøk, og sender da ingen token til sluttbrukeren.

Sekvensdiagrammet nedenfor illustrerer den fullstendige ADFS-flyten, med alternativ for vellykket og mislykket verifisering:

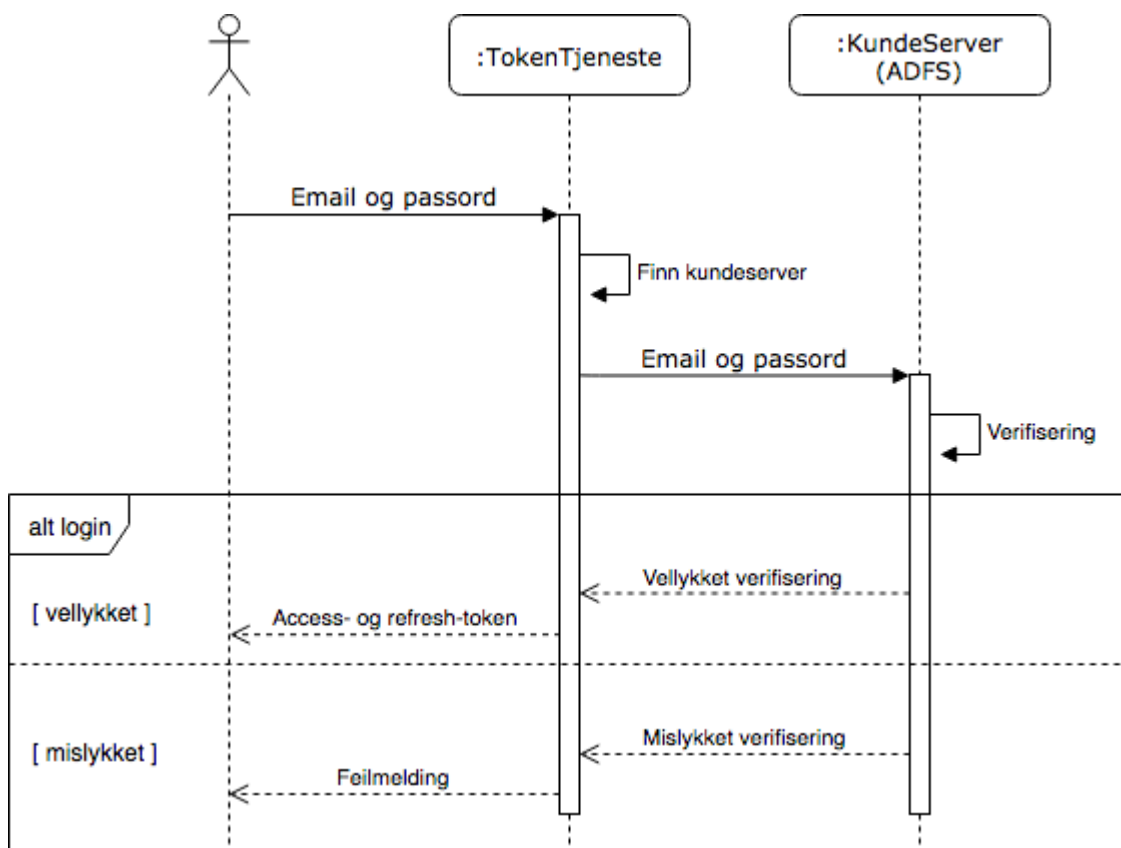


Fig. 5.3.6 - Sekvensdiagram for autoriseringsflyten i samspill med andre påloggingssystem (ADFS)

## Tiltak mot SQL-injection

Ettersom vår API gjør spørringer direkte til databasen for saksbehandlingssystemet, med informasjon sendt fra brukerens enhet, var vi nødt til å sikre løsningen mot såkalte SQL-injections. Denne typen av angrep innebærer at en angriper for eksempel sender med kode i et felt, for å så lure databasen til å utføre dette som om det hadde vært kode fra applikasjonen. På denne måten kan en angriper hente ut, slette, modifisere data, etc.

Et vanlig mottiltak for SQL-injection er bruk av såkalte “parameterized queries”, eller parametriserte spørringer. Dette betyr at vi spesifiserer at input fra brukeren kun skal håndteres som data (og ikke kode), samt hvilken type av data vi forventer oss å motta. På denne måten blir det svært vanskelig å utføre denne typen av angrep.



Kodesekvensen nedenfor viser denne typen av forberedelse for en databasespørring. Først blir en kommando opprettet. Parameteren `id` sendt ved metodekall til `GetNotes(int id)` legges til i selve spørringen først når typen til parameteren blir spesifisert. Når dette er utført kan kommandoen eksekveres via databasetilkoblingen:

```
public HttpResponseMessage GetNotes(int id)
{
    // ... autorisering, åpning av databasetilkobling, ...

    // Her forberedes en sql-spørring/kommando
    // Dette skal etterhvert sendes til databasen
    // @id er kun tekst og ikke id-parameteren fra oven
    SqlCommand command = new SqlCommand
    {
        CommandType = CommandType.Text,
        Connection = connection,
        CommandText =
            "select NOTE_ID, NOTE_DATE, NOTE_TEXT from INCIDENT_NOTE " +
            "where CASE_ID = @id " +
            "and HIDDEN = 0"
    };

    // Her definerer vi typen til parameteren id
    // Vi forventer en integer (heltall) med maksimal lengde på 10 tegn
    // Etter dette er gjort legges dette til i spørringen
    SqlParameter idParam = new SqlParameter("@id", SqlDbType.Int, 10);
    idParam.Value = id;
    command.Parameters.Add(idParam);
    command.Prepare();

    // Nå er det trygt å utføre spørringen mot databasen
    SqlDataReader reader = command.ExecuteReader();

    // ... lesing av retur, lukking av tilkobling, ...
}
```

## 5.4 Smartphone app

Selve smartphone-applikasjonen er som tidligere nevnt utviklet i rammeverket Ionic, som bruker Angular og webteknologier som for eksempel HTML5 og Typescript for å lage plattformuavhengige applikasjoner. Vår applikasjon er bygd opp med noen forskjellige typer komponenter:

### Pages

Pages i applikasjoner er sider som brukeren kan navigere til. Disse består i seg selv av et HTML-dokument som representerer bildet eller siden som sluttbrukeren ser. I dokumentet defineres plasseringen av elementer (tabeller, knapper, lister), innhold i tekster og lignende. Utseendet til elementene styles av en tilhørende SCSS-fil.

Aktivitetsdiagrammet på neste side (fig. 5.4.1) viser flyten vedrørende pages i applikasjonen. Noter at navigasjonen i appen skjer med hjelp av en stabelstruktur, det vil si at bruker alltid kan navigere opp i strukturen fra nåværende element (unntatt login-siden). Slutten (exit) er ikke representert i diagrammet, da applikasjonen kan lukkes uansett hvor bruker befinner seg:

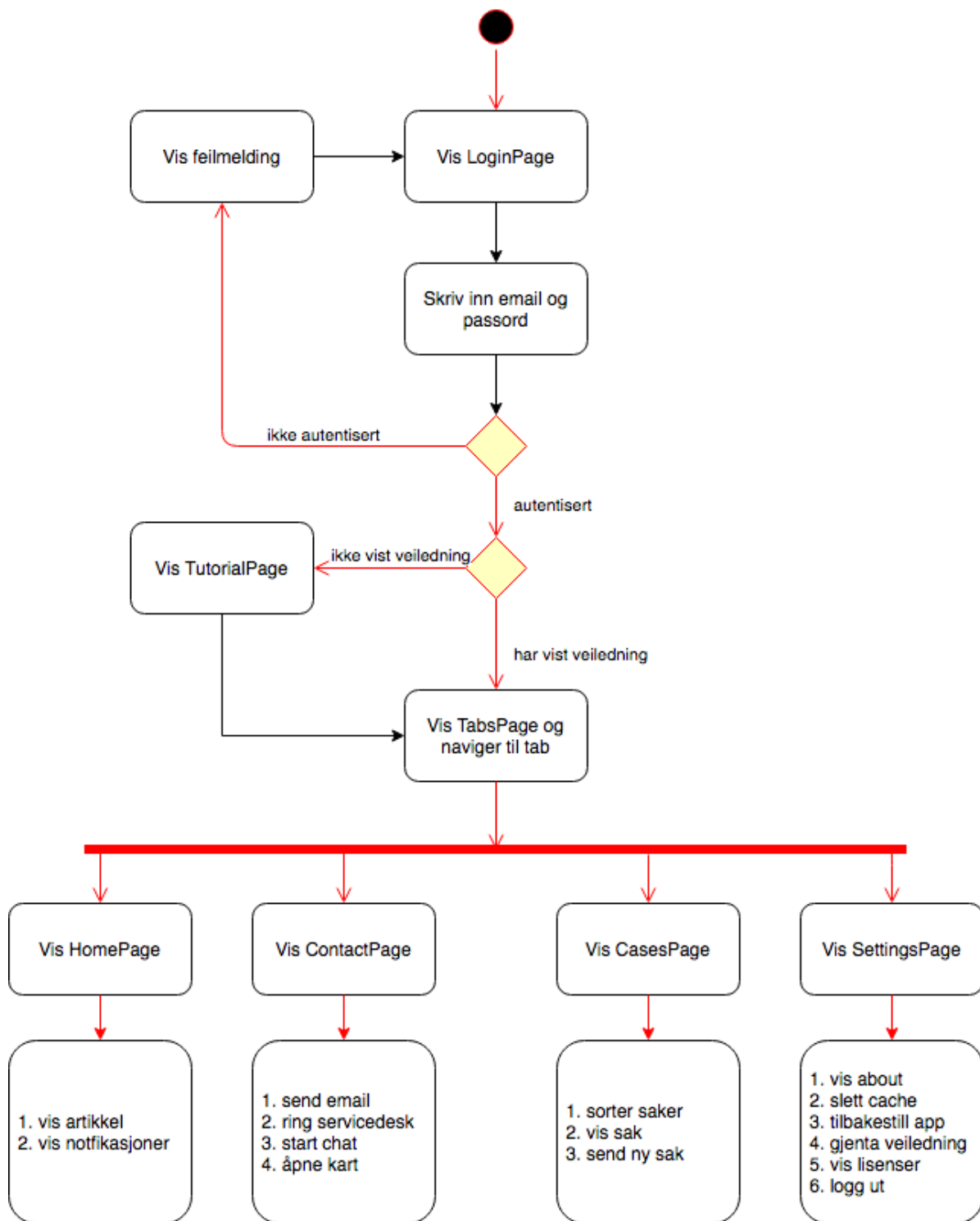


Fig. 5.4.1 - Aktivitetsdiagram som viser navigasjonsflyten i smartphone-appen. Symbolet for programslutt er ikke med, da bruker alltid kan lukke applikasjonen uansett posisjon

Pages har også en Typescript-fil som inneholder kode for forskjellige formål. Dette kan for eksempel være kode som skal kjøres når bruker aktiverer et element på siden (ved klikk på denne knappen, naviger til en annen side), eller henting av data for å så legge det inn i HTML-dokumentet brukeren kan se.

Den siste komponenten i en page er modul-filen. Denne brukes kun til å muliggjøre såkalt "lazy-loading". Dette betyr som tidligere nevnt at sider lastes inn når de først trengs istedenfor at samtlige sider lastes inn ved start av applikasjonen.

De tre følgende kodesnuttene viser samspillet mellom HTML, SCSS og Typescript for pagen(siden) "Home". Den første er HTML-filen der vi oppgir at vi skal vise en liste med artikler. Selve datalisten med artiklene (`weekArticles`) forventes ligge i Typescript-filen. Det forventes også at det finnes en metode med navnet `clickArticle(article)` som kalles når bruker berører elementet:

```
<ion-list padding>

  <h4 text-center>Last 7 days</h4>

  <p *ngIf="weekArticles?.length == 0" ion-text text-center margin-bottom
    color="light">No articles published</p>

  <div *ngFor="let article of weekArticles">
    <ion-item (click)="clickArticle(article);">
      <ion-avatar item-start>
        
      </ion-avatar>
      <h2>{{article.Title}}</h2>
      <ion-icon *ngIf="!article.Read;" name="radio-button-on" item-right
        color="primary"></ion-icon>
    </ion-item>
  </div>

  <!-- (sløyfet) -->

</ion-list>
```

Neste kodebit viser stilingen som skal appliseres på HTML-siden. Dette dokumentet er veldig kort da vi kun ønsker å overstyre det som ikke skal være i henhold til standard-styling. Ionic benytter et tema-dokument for formatering av alle elementer og det er dette vi overstyrer:

```
page-home {
  ion-list {
    margin-bottom: 8% !important;
  }

  .urgent-button {
    margin-top: 0 !important;
  }
}
```

Det siste kodeeksemplet er en sløyfet versjon av Typescript-filen. Denne filen inneholder opprinnelig funksjoner som blant annet henter data fra en tilbyder/provider, men dette blir videre diskutert i seksjonen Providers og er dermed fjernet. Noter at objektet `weekArticles` og funksjonen `clickArticle(article)` samsvarer med objektet i HTML-filen:

```
// ...

// Muliggjør Lazy-Loading
@IonicPage()
@Component({
  selector: 'page-home',
  templateUrl: 'home.html'
})

export class HomePage {

  articles: ArticleModel[] = new Array<ArticleModel>();
  weekArticles: ArticleModel[];
  monthArticles: ArticleModel[];
  oldArticles: ArticleModel[];

  // ...

  // Denne blir kalt ved aktivering av element i HTML-dokumentet
  clickArticle(article: ArticleModel) {
    this.setRead(article, true);
    this.navCtrl.push('ArticlePage', { article: article });
  }
}
```

```

// Denne funksjonen oppdaterer listen i denne filen
// Samtidig blir det bruker ser automatisk oppdatert
private sortArticlesByDate(unsorted: Array<ArticleModel>) {
    this.weekArticles = new Array<ArticleModel>();
    this.monthArticles = new Array<ArticleModel>();
    this.oldArticles = new Array<ArticleModel>();

    let lastWeek = new Date();
    let lastMonth = new Date();
    lastWeek.setDate(lastWeek.getDate() - 7);
    lastMonth.setMonth(lastMonth.getMonth() - 1);

    unsorted = unsorted.sort((a, b) =>
        b.PublishDate.getTime() - a.PublishDate.getTime()
    );

    unsorted.forEach(x => {
        if (x.PublishDate >= lastWeek) this.weekArticles.push(x);
        else if (x.PublishDate >= lastMonth) this.monthArticles.push(x);
        else this.oldArticles.push(x);
    });
}
}

```

## Modeller

For å sikre at mobilapplikasjonen bruker samme format for data-objekter som APIet og Dashboardet, har modellene fra APIet blitt brukt som mal til å definere hvordan samme objekter må struktureres i applikasjonen. Disse er implementert i mobilapplikasjonen som modeller. Koden nedenfor viser et eksempel på en slik struktur. Hvis vi for eksempel skal hente en artikkel fra APIet forventer vi å kunne opprette et objekt som nedenfor basert på returen, og hvis ikke så vil det vises en feilmelding:

```

// Container for en artikkel - denne må samsvare med returen fra APIet
export class ArticleModel {
    constructor(
        public Id: number,
        public Title: string,
        // ...
        public ExpiryDate: Date,
        public ImageUrl: string,
        public UpdateCount: number,

        // Sendes ikke fra APIet men brukes internt for å merke artikler som lest
        public Read: boolean
    ) {}
}

```

## Providers

For å unngå å ha alt for mye kode på hver side har lange kodesekvenser blitt flyttet til såkalte providers. Disse fungerer som tilbydere av data eller annen kodefunksjonalitet og har ikke noen visuelle elementer knyttet til seg.

Hvis en page(side) skal ha muligheten til å for eksempel kunne hente data fra APIet kan en provider injiseres i pagen(siden) sin typescript-fil. På den måten får pagen(siden) tilgang til funksjonaliteten uten å måtte spesifisere den i samme fil. Hvis flere pages skal bruke samme funksjonalitet kan også kode deles på denne måten.

Vår applikasjon følgende providere, med hver sitt dedikerte ansvarsområde:

- `ArticleProvider`
- `AuthenticatorProvider`
- `EmailProvider`
- `PushProvider`
- `TicketProvider`

`ArticleProvider` inneholder all kode relatert til å hente artikler publisert via dashboardet, samt å lagre (cache) disse på enheten som kjører applikasjonen. Hvis artikler ikke skulle bli lagret på enheten hadde det ikke vært mulig å for eksempel markere artikler som lest (dette skjer lokalt på telefonen), eller sammenligne allerede hentede artikler for å oppdage endringer. `ArticleProvider` bruker fremst funksjonen `getArticles()`, som er tatt med som et kodeeksempel lenger ned i denne seksjonen.

`AuthenticatorProvider` har ansvar for all kode relatert til autentisering. Hvis en page(side) eller en annen provider trenger en access-token for å kunne kommunisere med APIet injiseres `AuthenticatorProvider` i den aktuelle koden, og deretter kan det andre objektet spørre den injiserte provideren om å hente nye tokens. `AuthenticatorProvider` inneholder også funksjoner for å hente brukernavnet til den autentiserte brukeren, samt en hjelpemetode for å bekrefte at bruker faktisk er logget inn. Den sistnevnte funksjonen brukes i sider for å sjekke om brukeren får lov til å se innholdet, eller hvis bruker skal kastes tilbake til login-siden.

`EmailProvider` bruker et Ionic-tillegg (`EmailComposer`) for å muliggjøre strukturering av mailer før de blir åpnet i brukerens email-applikasjon. Dette lar oss spesifisere innholdet i felter i mail som for eksempel mottaker, tittel, innhold og så videre. Provideren sjekker også at applikasjonen faktisk har tilgang til å kalle mail-applikasjonen til systemet, og ber om tillatelse hvis dette mangler.

`PushProvider` sørger for at telefonen abonnerer på og henter push-varslere. Den gjør også brukeren oppmerksom på disse via notifikasjoner i telefonens statuslinje eller med hjelp av dialoger. Provideren lagrer alle mottatte notifikasjoner slik at disse kan leses i etterkant. Se seksjonen Push i dette kapittelet for mer detaljert beskrivelse.

`TicketProvider` fungerer på samme måte som `ArticleProvider`, og brukes til å hente saker og tilhørende notater/korrespondanse fra APIet.

Koden nedenfor viser hvordan providers injiseres i andre filer for å utføre spesifikke oppgaver. I dette eksemplet bruker `HomePage` (listen over artikler) providerne `AuthenticatorProvider` og `ArticleProvider`. `AuthenticatorProvider` sjekker om bruker er pålogget, det vil si at bruker har en token som kan brukes. `ArticleProvider` henter så artiklene fra APIet og returnerer en liste som brukes av `HomePage`.

```
export class HomePage {  
  
  // ...  
  
  // Providers blir injisert i konstruktøren (Article og Authenticator)  
  constructor(  
    public navCtrl: NavController,  
    public events: Events,  
    private provider: ArticleProvider,  
    private alertCtrl: AlertController,  
    private auth: AuthenticatorProvider,  
    private storage: Storage,  
    private modalCtrl: ModalController  
  ) { }  
  
  // Her verifiserer AuthenticatorProvider at bruker er logget inn  
  ionViewCanEnter() {  
    return this.auth.isAuthenticated();  
  }  
}
```



```

// ...

// Her kalles ArticleProvider for å hente artikler fra APIet
// Token som brukes i getArticles hentes fra auth (AuthenticationProvider)
refresh(refresher: Refresher) {

    // Funksjonen getArticles blir videre forklart i neste eksempel
    this.provider.getArticles(this.auth.getToken()).then(fetchedItems => {

        // Oppdater antall nye (uleste) artikler
        // Dette blir brukt for å oppdatere brukergrensesnittet
        this.unreadCount = 0;
        fetchedItems.forEach(newItem => {
            if (!newItem.Read) {
                this.unreadCount++;
                this.events.publish("article:unreadCount", this.unreadCount);
            }
        });

        // Bytt ut liste med den nye
        this.articles = fetchedItems;

    }).catch(msg => {

        // Vis feilmelding for bruker hvis artikler ikke ble hentet
        let alert = this.alertCtrl.create({
            title: 'Warning',
            message: 'Could not fetch articles from server',
            buttons: ['Close']
        });

        alert.present();

    }).then(x => {

        // Sorter de nye artiklene
        this.sortArticlesByDate(this.articles);

        // Stopp refresh-indikator
        if (refresher !== null) refresher.complete();
        this.refreshing = false;
    });
}

// ...
}

```

Neste kodeeksempel viser provideren `ArticleProvider` som ble brukt i forrige eksempel (`HomePage`). `ArticleProvider` brukes som nevnt til å hente artikler fra APIet. Provideren sammenligner også artikler som allerede har blitt lastet ned tidligere for å se om endringer på selve artiklene er gjort. For å holde oversikt over hvilke artikler som allerede er lest blir disse lagret på telefonens internminne.

Funksjonen som henter artikler er også “asynkron”, noe som betyr at den kan kalles uten å stoppe opp flyten i andre deler av programmet. Dette er meget ønskelig, fordi at bruker da ikke må vente på at koden kjører ferdig før personen fortsetter å bruke applikasjonen. Denne typen av funksjon gjør ofte koden litt mer kompleks (typisk da asynkrone funksjoner inneholder andre asynkrone funksjoner). Vi har til tross for dette valgt å bruke asynkrone metoder alle steder der venting eller treghet kan oppstå, slik at vi heller sikrer en brukervennlig applikasjon.

```
// @Injectable() gjør det mulig å injisere denne koden i andre klasser
@Injectable()
export class ArticleProvider {

    // Henter artikler fra APIet (autentisering via access-token) via HTTP
    getArticles(token: string): Promise<ArticleModel[]> {

        // En promise gjør funksjonen asynkron, da det går at “vente” på en promise
        let promise = new Promise<ArticleModel[]>((resolve, reject) => {

            // Her forberedes et HTTP-kall til APIet
            let url = 'https://webapi-app-test.99x.no/api/article';
            let headers = new HttpHeaders({
                'Content-Type': 'application/json',
                'Accept': 'application/json',
                'Authorization': 'Bearer ' + token
            });

            // Telefonen sender deretter kallet til APIet og venter på svar
            this.http.get(url, { headers: headers }).toPromise().then(res => {

                // Her har serveren svart og vi må opprette objekter fra returen
                let fetchedArticles = new Array<ArticleModel>();
                let parsed = JSON.parse(JSON.stringify(res));
                parsed.forEach(element => {
                    fetchedArticles.push(new ArticleModel(
                        element.Id,
                        element.Title,
                        // resterende attributter ...
                    ));
                });
            });
        });
    }
}
```

```

// Nå henter vi Lagrede artikler fra telefonens minne
this.storage.get("CACHED_ARTICLES").then(cachedArticles => {

    // Aldri lagret før - Lagre og returner artiklene
    if (cachedArticles == null) {
        this.storage.set("CACHED_ARTICLES",
            JSON.stringify(fetchedArticles));

        // Resolve av en promise kan sees på som en retur
        // Vi trenger ikke å gjøre mer så vi stopper funksjonen her
        resolve(fetchedArticles);
    }

    // Det finnes lagrede artikler - sammenlign og oppdater
    else {
        let parsedFromCache = JSON.parse(cachedArticles);
        let parsedStorageArray = new Array<ArticleModel>();
        parsedFromCache.forEach(element => {
            parsedStorageArray.push(new ArticleModel(
                element.Id,
                element.Title,
                // resterende attributter ...
            ));
        });

        // Oppdater den lagrede strukturen
        parsedStorageArray.forEach(oldArticle => {
            fetchedArticles.forEach(newArticle => {

                if (oldArticle.Id === newArticle.Id) {
                    oldArticle.Title = newArticle.Title;
                    oldArticle.Content = newArticle.Content;
                    // resterende attributter ...

                    // Marker artikkel som ulest hvis endringer er gjort
                    if (oldArticle.UpdateCount != newArticle.UpdateCount) {
                        oldArticle.UpdateCount = newArticle.UpdateCount;
                        oldArticle.Read = false;
                    }
                }
            });
        });

        // Hvis en ny artikkel oppdages må denne legges til i listen
        fetchedArticles.forEach(newArticle => {
            if (parsedStorageArray.find
                (old => old.Id == newArticle.Id) == null) {
                parsedStorageArray.unshift(newArticle);
            }
        });

        // Lagre artiklene og returner den nye strukturen
        this.storage.set("CACHED_ARTICLES", JSON.stringify(parsedStorageArray));
        resolve(parsedStorageArray);
    }
});

```

```

    // Fang opp feil og send disse som retur
    }, msg => {
        reject(msg);
    });
});

// Hele kodesekvensen fra begynnelsen av funksjonen returneres
return promise;
}

// Hjelpemetode for å konvertere datoer fra APIet til riktig format
private toDate(jsonDate): Date {
    if (jsonDate === null) return null;
    return new Date(parseInt(jsonDate.substr(6)));
}
}
}

```

## Push

Som nevnt om provideren `PushProvider` har denne ansvaret for å hente notifikasjoner sendt ut fra Firebase (tjenesten vi bruker for å sende meldinger). For å sikre at brukeren kun mottar notifikasjoner sendt til dennes sitt firma bruker vi såkalte kanaler/channels.

Hver enhet vil lytte til to kanaler: "all", som innebærer alle brukere av mobilapplikasjonen, samt en firmaspesifikk (for eksempel "99x" eller "SIO"). Hvis brukeren for eksempel jobber på 99X vil push-tjenesten kun lytte etter meldinger på kanalene "99x" og felleskanalen "all".

Kodeeksemplet nedenfor viser en sløyfet versjon av `PushProvider`. Denne koden kjøres først når bruker har logget på applikasjonen (for å unngå at brukere som ikke er autentiserte kan hente meldinger):

```

@Injectable()
export class PushProvider {
    private pushToken: string = '';

    constructor(
        private fcm: FCM,
        private auth: AuthenticatorProvider,
        private storage: Storage,
        private alertCtrl: AlertController,
        private events: Events,
        private platform: Platform
    ) {

```

```

// Lytt på varsler sendt til alle app-brukere
this.fcm.subscribeToTopic('all');

// Ekstraher domenet fra email og lytt på varsler på denne kanalen
const domain = this.auth.getEmail().toLowerCase().replace(/.*@/, '');
this.fcm.subscribeToTopic('domain');

// Sørger for at enheten har en gyldig token fra Firebase
this.fcm.getToken().then(token => {
  this.pushToken = token;
});

// Definerer hva som skal skje når en push blir mottatt
this.fcm.onNotification().subscribe((data) => {

  // Dette betyr at applikasjonen kjører når meldingen ble mottatt
  // Her viser vi en alert-dialog slik at bruker blir informert
  // Hvis dette ikke skjer blir notifikasjonen lagt til i status-listen
  if (!data.wasTapped) {
    let alert = this.alertCtrl.create({
      title: 'Urgent message!',
      subTitle: data.date,
      message: data.message,
      buttons: ['Got it']
    });
    alert.present();
  }

  // Lagre informasjonen lokalt på enheten
  this.storageHelper(data);
});

// Oppdaterer lagret token
this.fcm.onTokenRefresh().subscribe(token => {
  this.pushToken = token;
});
}

// Hjelpemetode for å lagre meldinger lokalt
private storageHelper(data) {
  this.storage.get("PUSH_ARRAY").then(storedArray => {

    // Første kjøringen / første melding mottatt
    if (storedArray == null) {

      let tempArray = new Array<string>();
      data.read = false;
      let item = JSON.stringify(data);

      tempArray.unshift(item);
      this.storage.set("PUSH_ARRAY", JSON.stringify(tempArray))
        .then(x => this.events.publish("alert:new"));
    }
  }
}

```

```

    // Legg til i eksisterende liste
    else {

        let tempArray = JSON.parse(storedArray);
        data.read = false;
        let item = JSON.stringify(data);

        tempArray.unshift(item);

        this.storage.set("PUSH_ARRAY", JSON.stringify(tempArray))
            .then(x => this.events.publish("alert:new"));
    }
});
}
}
}

```

Før en enhet kan lytte etter push-varslere fra Firebase må enheten først autentiseres mot Firebase sin API. Dette gjøres via en nøkkel lagret i filen `google-services.json`:

```

{
  "project_info": {
    "project_number": "9541_XXXXX_23867",
    "firebase_url": "https://smartphone-app-99x.firebaseio.com",
    "project_id": "smartphone-app-99x",
    "storage_bucket": "smartphone-app-99x.appspot.com"
  },
  "client": [
    {
      "client_info": {
        "mobilesdk_app_id": "1:95_XXXXX_23867:android:9a49a694f89c51fe",
        "android_client_info": {
          "package_name": "guru.lundberg.app99x"
        }
      },
      "oauth_client": [
        {
          "client_id": "954109723867-9v4odpardrr_XXXXX_.apps.googleusercontent.com",
          "client_type": 3
        }
      ],
      "api_key": [
        {
          "current_key": "AIzaSyC8NnJaop_XXXXXX_ZjxZzMankC4bCaw"
        }
      ],
      // .....
    }
  ]
}

```

## 5.5 Web API

API står for Application Programming Interface (eller programmeringsgrensesnitt) og kan sees på som en samling definisjoner om hvordan andre komponenter og/eller systemer kan kommunisere med serveren, applikasjonen eller tjenesten som tilbyr grensesnittet.

Web API er en videre definisjon av konseptet, der det forutsettes at kommunikasjonen til APIet skjer over internett via HTTP (Hypertext Transfer Protocol). Når man surfer på internett gjøres HTTP-kall av forskjellige typer (for eksempel GET eller POST) av nettleseren til stedet man besøker. Serveren som kjører nettstedet sender da ressurser tilbake til nettleseren. Skjermbildet nedenfor viser det som skjer i bakgrunnen når en person besøker oslomet.no:

Status	Method	File	Domain	Cause	Type	Transferred
●	GET	Vi-er-blitt-OsloMet-storbyuniversitetet	www.hioa.no	document	html	0 GB
▲ 304	GET	da876c0edcacbe616bc0946b747eabf8.js	www.hioa.no	script	js	cached
▲ 304	GET	SitesterSurvey_1750_no_bugfix.min.js	www.hioa.no	script	js	cached
● 200	GET	7b538ffe42df86820aef7f4d9a239a30_all.css	www.hioa.no	stylesheet	css	41.11 KB
● 200	GET	print.css	www.hioa.no	stylesheet	css	1.97 KB
● 200	GET	upgrade_rearrange_fixes.css	www.hioa.no	stylesheet	css	735 B
▲ 304	GET	jquery.min.js	ajax.googleapis.com	script	js	cached
▲ 304	GET	Hioa-Logo-s_h-orig.png	www.hioa.no	img	png	cached
▲ 304	GET	hioa-meny-knapp_off.png	www.hioa.no	img	png	cached
▲ 304	GET	hioa-sok-knapp_off.png	www.hioa.no	img	png	cached

Fig. 5.5.1 - Output i nettverks-fanen i Firefox som viser spørringer til og fra serveren

### Controllere og ruter

På samme måte som en nettleser ber om ressurser fra et nettsted kommuniserer vår smartphone-applikasjon med ressurs-serveren via et Web API. Når en bruker av appen ønsker å hente artikler fra serveren gjøres et HTTP-kall av typen GET til APIet, som i sin tur sender et svar med en liste av artikler.

APIet består av fire controller-klasser. Controllere styrer handlinger basert på ruten/adressen og er en del av ASP.NET-rammeverket. Hvis man skal ha tak i en nettside som ikke bruker en controller, kan man for eksempel gjøre et kall til [www.eksempel.no/sider/hjemmeside.php](http://www.eksempel.no/sider/hjemmeside.php). På serveren kan det da eksistere en mappe som heter "sider" og som inneholder filen `hjemmeside.php`.

I en ASP.NET-applikasjon derimot kan et kall gjøres til en controller, for eksempel `www.eksempel.no/Home/index`. På serveren vil det da eksistere en controller som heter Home, som så vil vise "index" siden gjennom å kalle en metode som tilhører controlleren.

I denne seksjonen går vi igjennom hvordan controllerne er satt opp, og hvilke kall de støtter. Merk at samtlige controllere er underklasser av klassen `AbstractController`. Denne inneholder kun metoder for logging av feilsituasjoner og formatering av svar på HTTP-kall.

## Article

Metode	Adresse	Parametere	Beskrivelse
GET	<code>/article</code>		Henter alle artikler
GET	<code>/article/{id}</code>	numerisk id	Henter en bestemt artikkel
POST	<code>/article</code>	artikkel	Oppretter en ny artikkel
PUT	<code>/article</code>	artikkel	Oppdaterer en artikkel
DELETE	<code>/article/{id}</code>	numerisk id	Sletter en artikkel

I `ArticleController` er HTTP-kallene over satt opp som offentlige metoder. De returnerer en `HttpResponseMessage`, som består av en tilbakemelding og en statuskode. Tilbakemeldingen kan være en etterspurt ressurs i JSON-format, eller en feilmelding hvis noe gikk galt. En feil som kan oppstå kan for eksempel være «Connection to server failed».

Statuskoden er et tresifret tall som beskriver tilbakemeldingen. Disse er universelle og brukes av alle nettsider. En server kan for eksempel returnere koden 200, som sier at operasjonen var vellykket, eller 401 som indikerer at en bruker ikke er autorisert. Den mest kjente statuskoden er kanskje feilkoden 404, som sier at ressursen ikke ble funnet.

Under er metoden for å hente en spesifikk artikkel. Attributtene spesifiserer at adressen skal avsluttes med ID, for eksempel `/article/3`, og at HTTP-metoden GET skal brukes. I koden hentes ønsket artikkel fra objektet `_repo`, som er av typen `ArticleRepository` og håndterer kommunikasjon med databasen. Hvis artikkelen ikke eksisterer lages et kall til metoden `Respond`, som ligger i forelder-klassen `AbstractController`. Der vil `ResponseNotFound` referere til statuskoden 404. Hvis artikkelen finnes returneres artikkelen sammen med statuskoden 200:



```

[Route("{id}")]
[HttpGet]
public HttpResponseMessage Get(int id)
{
    var article = _repo.Get(id);
    if (article == null)
        return Respond(null, ResponseNotFound);
    var json = Serializer.Serialize(article);
    return Respond(json, ResponseOk);
}

```

Kodebiten nedenfor viser de forhåndsdefinerte svarskodene som ligger i `AbstractController`. Metoden `Respond` brukes av alle kontrollere som arver fra denne klassen. Denne metoden lar oss ha større kontroll på feilmeldinger som serveren sender:

```

public abstract class AbstractController : ApiController
{
    // ...

    // Predefinerte svarskoder
    protected const int ResponseOk = 1;
    protected const int ResponseOkNoJson = 2;
    protected const int ResponseNotFound = 3;
    protected const int ResponseInvalidModel = 4;
    protected const int ResponseSaveFailed = 5;
    protected const int ResponseUnauthorized = 6;
    protected const int ResponseConnectionFailure = 7;

    protected HttpResponseMessage Respond(string json, int type)
    {
        var response = new HttpResponseMessage();

        switch (type)
        {
            // Kode for noe som ikke er funnet i for eksempel databasen
            case ResponseNotFound:
            {
                response.Content = new StringContent("Could not find ID. " + json);
                response.StatusCode = HttpStatusCode.NotFound;
                break;
            }

            // Andre svarskoder (sløyfet) ...
        }

        return response;
    }
}

```

I tillegg består artikkel-controlleren av en privat hjelpemetode kalt `validateArticle`. Den brukes både når man skal lage en ny artikkel (POST), og når man skal endre en artikkel (PUT). Metoden returnerer en tuppel (tuple), som er en samling av flere objekter. Vi bruker dette slik at vi får returnert artikkelen, etter at den har blitt validert, sammen med en eventuell tilbakemelding. Hvis en feil oppstår returneres POST- eller PUT-metoden med en feilmelding.

I metoden sjekkes først om attributten `PublishDate` er satt. Hvis ikke vil `PublishDate` settes til dagens dato. Men andre ord vil artikkelen bli publisert med en gang den har blitt laget. Deretter sjekkes det om tittelen og artikkelteksten er satt, som det er obligatoriske å ha med. Hvis de ikke er satt returneres en tuppel med en feilmelding. Hvis artikkelen ikke inneholder feil returneres artikkelen uten feilmelding.

```
private Tuple<string, ArticleModel> ValidateArticle(ArticleModel article)
{
    if (article.PublishDate == null)
    {
        article.PublishDate = DateTime.Now;
    }

    if (article.Title == null || article.Content == null)
    {
        return Tuple.Create("The title and article text needs to be filled out", article);
    }

    // ...

    return Tuple.Create("", article);
}
```

## Channel

Metode	Adresse	Parametere	Beskrivelse
POST	/channel	channel	Oppretter en ny channel
GET	/channel		Henter alle channels
DELETE	/channel/{navn}	channel-navn	Sletter en channel

ChannelController inneholder metoder for å legge til, hente og slette kanaler. Disse inneholder ingen nevneverdige løsninger som avviker fra ASP.NET WebAPI-standarder, og vi velger derfor ikke å ta med noen eksempler eller forklaringer.

## Notification

Metode	Adresse	Parametere	Beskrivelse
GET	/notification		Henter alle notifikasjoner (pushvarsler)
POST	/notification	notification	Oppretter og sender en ny notifikasjon

NotificationController sender notifikasjoner til brukerne i form av push-varsler. Fra dashboardet lages notifikasjonen med en meldingstekst, en mottakergruppe, og en eventuell tidsbegrensning på hvor lenge den skal være aktiv. I controlleren gjøres så notifikasjonen om til et formatert kall til tjenesten Firebase Cloud Messaging som så sender push-varsler til brukerne av appen.

Firebase har et eget dashboard som kan benyttes til å sende push-varsler. Vi har dog valgt å heller implementere dette i vår dashboard-løsning, for å samle funksjonaliteten på ett sted. For dette bruker vi Firebase sitt API.

Neste kodesekvens viser utsendelsen av et push-varsel. Koden lager et JSON-objekt ut ifra notifikasjonen som ble sendt med. Først gjøres en sjekk på om `ExpiryDate` er satt, som bestemmer hvor lenge notifikasjonen skal være aktiv. Hvis `ExpiryDate` ikke er satt blir den satt til 12 timer. Hvis en mobil-bruker ikke har fått meldingen innen 12 timer vil vedkommende ikke få varselet.

Mottakergruppen av notifikasjonen settes i attributtet `topics` i JSON-objektet. Dette objektet er det som sendes til Firebase. Det er sammensatt av de to strukturene `notificationValues` og `dataValues` som spesifiserer innholdet og konfigurasjonen til meldingen. `dataValues` har også et felt for dato og tid, som vil bli satt til tidspunktet hvor notifikasjonen ble sendt.

```

public HttpResponseMessage Post(NotificationModel message)
{
    if (ModelState.IsValid)
    {
        // antall sekunder notifikasjonen vil være tilgjengelig:
        int timeToLiveSeconds;

        if (message.ExpiryDate == null)
        {
            // settes til 12 timer:
            timeToLiveSeconds = 43200;
        }
        else
        {
            TimeSpan timeToLive = (DateTime)message.ExpiryDate - DateTime.Now;
            timeToLiveSeconds = (int)Math.Floor(timeToLive.TotalSeconds);
        }

        // Lager et JSON-objekt som skal sendes til FCM:
        JObject notificationValues = new JObject(
            new JProperty("body", message.Message),
            new JProperty("click_action", "FCM_PLUGIN_ACTIVITY")
        );

        JObject dataValues = new JObject(
            new JProperty("message", message.Message),
            new JProperty("date", DateTime.Now.ToShortDateString() + " " +
                DateTime.Now.ToShortTimeString())
        );

        JObject jsonObject = new JObject(
            new JProperty("to", "/topics/" + message.ChannelName),
            new JProperty("notification", notificationValues),
            new JProperty("data", dataValues),
            new JProperty("time_to_live", timeToLiveSeconds)
        );
    }
}

```

I tillegg til å sende notifikasjonen via Firebase vil controlleren også lagre notifikasjonen i en database. Dette har vi gjort for at brukerne av dashboardet skal få en historikk på hvilke notifikasjoner som er sendt ut. Hvis lagring av meldingen feiler blir den fortsatt sendt ut, ettersom disse meldingene ofte er kritiske og må ut til brukerne raskest mulig.

```

if (_repo.Add(message))
{
    return Respond("Successfully sent push notification!", ResponseOk);
}

return Respond("The message was sent, but could not be logged in the database",
    ResponseInvalidModel);

```

## Ticket

Metode	Adresse	Parametere	Beskrivelse
GET	/ticket/gettickets		Henter alle saker fra saksbehandlingssystemet for autentisert bruker
GET	/ticket/getnotes/{id}	ticket-id	Henter alle notater på en bestemt sak fra saksbehandlingssystemet

Controlleren for Ticket har ansvaret for å hente ut data fra den klonede databasen for saksbehandlingssystemet Livetime. Den avviker fra de tidligere nevnte controllerne, i det at den ikke bruker vår egen database (via Entity Framework) for å behandle eller hente ut data. Ettersom vi ikke trenger å modifisere dataen i saksbehandlingssystemet har vi løst integrasjonen gjennom å gjøre spørringer direkte til databasen fra kontrolleren.

Kodeeksemplet for metoden `GetTickets()` nedenfor viser hvordan `TicketController` kommuniserer med den klonede databasen. Grunnet lengde har vi delt opp eksemplet i tre deler. Først sjekker koden om bruker er autorisert. Deretter opprettes en tilkobling mot den klonede databasen (adressen til databasen er lagret i en konfigurasjonsfil). Hvis tilkoblingen mislykkes lagres det en logg på serverens filsystem:

```
// Autorisering av bruker (token)
var user = HttpContext.Current.User.Identity;
if (!user.IsAuthenticated || user.Name == null) return Respond(null,
ResponseUnauthorized);

// Forbered og åpne en tilkobling til databasen
SqlConnection connection = new
SqlConnection(WebConfigurationManager.ConnectionStrings["LivetimeReplicaConnection"].Conne
ctionString);

try
{
    connection.Open();
}

catch (Exception e)
{
    Debug.WriteLine(e.Message);
    Logger.LogError("TicketController\tfailed to connect\t" + e.Message);
    return Respond(null, ResponseConnectionFailure);
}
```

I neste moment spesifiserer vi spørringen som skal utføres mot databasen. For å unngå at brukere får tilgang til saker som tilhører andre brukere benytter vi emailadressen til den autentiserte brukeren. Brukernavnet i Livetime-systemet skal ifølge 99X alltid samsvare med enhver sluttbrukers primære mailadresse, og således kan vi da bruke det som nøkkel i spørringen. Vi sikrer også spørringen mot SQL-injections, noe som blir forklart i kapittel 5.3.

```
// Forbered spørring
SqlCommand command = new SqlCommand
{
    CommandType = CommandType.Text,
    Connection = connection,
    CommandText =
        "select i.CASE_ID, i.CUST_CC, i.OPEN_DATE, i.CLOSE_DATE, i.SUBJECT, " +
        "i.QUESTION_TEXT, s.STATUS_TYPE_NAME, i.TYPE " +
        "from INCIDENT as i " +
        "inner join STATUS_TYPE as s " +
        "on i.STATUS_TYPE_ID = s.STATUS_TYPE_ID " +
        "where i.CLIENT_ID = (" +
        "select CLIENT_ID from CLIENT where EMAIL = @username) " +
        "and i.DELETED = 0"
};

// Sikre spørringen mot SQL-injection
SqlParameter userNameParam = new SqlParameter("@username", SqlDbType.VarChar, 100);
userNameParam.Value = user.Name;
command.Parameters.Add(userNameParam);
command.Prepare();
```

Etter at spørringen er forberedt utføres spørringen mot databasen. For å kunne lese gjennom returen oppretter vi en dataleser (`SqlDataReader`), som lar oss iterere gjennom resultatet rad for rad. For å sikre at alle parametere blir satt lager vi først standardverdier for alle felter vi skal lese. Deretter sjekker vi hver kolonneindeks i hver rad for å unngå nullmerker. Dette må vi gjøre ettersom Livetime-databasen som nevnt inneholder svært mye nullmerker, og disse kan få selve dataleser-objektet til å krasje.

For hver rad som ble lest oppretter vi et DTO (Data Transfer Object) og legger dette i en liste. Når dataleseren er ferdig sørger vi for å lukke selve leseren og databasetilkoblingen. Til sist serialiserer vi listen med DTOs (konverterer dem til en tekststreng) og returnerer resultatet.

```

// Kjør spørringen og Les resultatet
SqlDataReader reader = command.ExecuteReader();
var items = new List<TicketDTO>();
if (reader.HasRows)
{
    while (reader.Read())
    {
        int id = -1;
        string cc = string.Empty;
        DateTime? opened = null;
        DateTime? closed = null;
        string subject = string.Empty;
        string text = string.Empty;
        string status = string.Empty;
        int type = -1;

        // Håndter nullmerker fra databasen
        if (!reader.IsDBNull(0)) id = reader.GetInt32(0);
        if (!reader.IsDBNull(1)) cc = reader.GetString(1);
        if (!reader.IsDBNull(2)) opened = reader.GetDateTime(2);
        if (!reader.IsDBNull(3)) closed = reader.GetDateTime(3);
        if (!reader.IsDBNull(4)) subject = reader.GetString(4);
        if (!reader.IsDBNull(5)) text = reader.GetString(5);
        if (!reader.IsDBNull(6)) status = reader.GetString(6);
        if (!reader.IsDBNull(7)) type = reader.GetInt32(7);

        // Lag et data transfer object and legg til i listen over saker
        var dto = new TicketDTO
        {
            CASE_ID = id,
            CUST_CC = cc,
            OPEN_DATE = opened,
            CLOSE_DATE = closed,
            SUBJECT = subject,
            QUESTION_TEXT = text,
            STATUS_TYPE_NAME = status,
            TYPE = type
        };

        items.Add(dto);
    }
}

// Steng resultat-leser og tilkobling
reader.Close();
connection.Close();

// Send listen i retur i JSON-format
var json = Serializer.Serialize(items);
return Respond(json, ResponseOk);

```

## Token

Metode	Adresse	Parametere	Beskrivelse
POST	/token	grant_type user_name password	Autentiserer bruker via brukernavn og passord og returnerer deretter tokens ved vellykket autentisering
POST	/token	grant_type refresh_token	Autentiserer bruker via en refresh-token og returnerer deretter tokens ved vellykket autentisering

Token er en dedikert rute for å håndtere autentisering av brukere og for utveksling av tokens. Ruten har ikke en controller knyttet til seg, men håndteres i stedet av autentiseringssystemet vi benytter (ASP.NET Identity). Dette blir forklart i neste seksjon.

## Autentisering i APIet

For å hindre at hvem som helst kan bruke controller-metodene i listene ovenfor var vi nødt til å sperre adgang basert på hvilken rolle en bruker har (dette kalles ofte rollebasert access-policy eller RBAC). En kunde-bruker skal for eksempel aldri kunne slette en artikkel, mens en administrator skal kunne gjøre det. Heldigvis er dette innebygget i ASP.NET Identity og vi kunne da spesifisere tilgang for hver enkelt metode.

Neste kodeeksempel viser tilgangsdelegering i controller-klassen for artikler. Noter at vi øverst i klassen spesifiserer at kun brukere i rollene Administrator og/eller Customer skal ha tilgang til alle metodene i klassen. På Post, Delete og Put spesifiserer vi deretter at kun Administrator skal ha tilgang (og ikke Customer):



```

[Authorize(Roles = "Administrator, Customer")]
[RoutePrefix("Article")]
public class ArticleController : AbstractController
{
    private readonly ArticleRepository _repo = new ArticleRepository();

    // Henting av artikler
    // Alle har tilgang (hvis autentisert som administrator eller kunde)
    [Route("")]
    [HttpGet]
    public HttpResponseMessage Get()
    {
        // ...
    }

    // Henting av spesifikk artikkel - alle har tilgang
    [Route("{id}")]
    [HttpGet]
    public HttpResponseMessage Get(int id)
    {
        // ...
    }

    // Opprettelse av artikkel - kun autentiserte administratorer har tilgang
    [Authorize(Roles = "Administrator")]
    [Route("")]
    [HttpPost]
    public HttpResponseMessage Post(ArticleModel article)
    {
        // ...
    }

    // Sletting av artikkel - kun autentiserte administratorer har tilgang
    [Authorize(Roles = "Administrator")]
    [Route("{id}")]
    [HttpDelete]
    public HttpResponseMessage Delete(int id)
    {
        // ...
    }

    // Oppdatering av artikkel - kun autentiserte administratorer har tilgang
    [Authorize(Roles = "Administrator")]
    [Route("")]
    [HttpPut]
    public HttpResponseMessage Put(ArticleModel article)
    {
        // ...
    }

    // ...
}

```

## Repositories

For å oppnå et lag med abstraksjon mellom database- og API-logikk har vi implementert et såkalt repository-pattern[8]. Dette gir oss muligheten til å holde koden i controllermetodene kort og oversiktlig, fordi database-relatert kode flyttes til repository-klasser. Kodeeksemplet nedenfor viser et utsnitt av repository-klassen knyttet til autentiseringsdatabasen:

```
public class AuthRepository : IDisposable
{
    private readonly AuthContext _ctx;
    private readonly UserManager<IdentityUser> _userManager;

    // ...

    // Legger til en ny refresh token i databasen
    // Hvis token allerede er tildelt bruker blir den gamle slettet
    public async Task<bool> AddRefreshToken(RefreshToken token)
    {
        var existingToken = _ctx.RefreshTokens.SingleOrDefault(
            r => r.Subject == token.Subject
        );

        if (existingToken != null) await RemoveRefreshToken(existingToken);
        _ctx.RefreshTokens.Add(token);
        return await _ctx.SaveChangesAsync() > 0;
    }

    // ...

    // Her sørges det for at ressurser blir frigjort
    public void Dispose()
    {
        _ctx.Dispose();
        _userManager.Dispose();
    }
}
```

En annen positiv effekt av å bruke repository-pattern er at vi enkelt kan implementere grensesnittet `IDisposable`, som gjør det mulig å bruke funksjonen “using” i C#. Kort fortalt sørger dette for at alle opprettede ressurser automatisk frigjøres etter at kodesekvensen er fullført. Nedenfor er et eksempel på bruk av “using” for repository-klassen i forrige kodeeksempel:

```
// Her opprettes en ny repository (inkl. databasekobling)
using (var repo = new AuthRepository())
{
    const int ttl = 60 * 24 * 30;

    var token = new RefreshToken
    {
        Id = Helpers.GetHash(refreshTokenId),
        Subject = context.Ticket.Identity.Name,
        IssuedUtc = DateTime.UtcNow,
        ExpiresUtc = DateTime.UtcNow.AddMinutes(Convert.ToDouble(ttl))
    };

    context.Ticket.Properties.IssuedUtc = token.IssuedUtc;
    context.Ticket.Properties.ExpiresUtc = token.ExpiresUtc;
    token.ProtectedTicket = context.SerializeTicket();

    // Her blir metoden fra AuthRepository kalt
    var result = await repo.AddRefreshToken(token);
    if (result) context.SetToken(refreshTokenId);
}

// Her er ressursene frigjort (bland annet databasekobling)
```

## 5.6 Dashboard

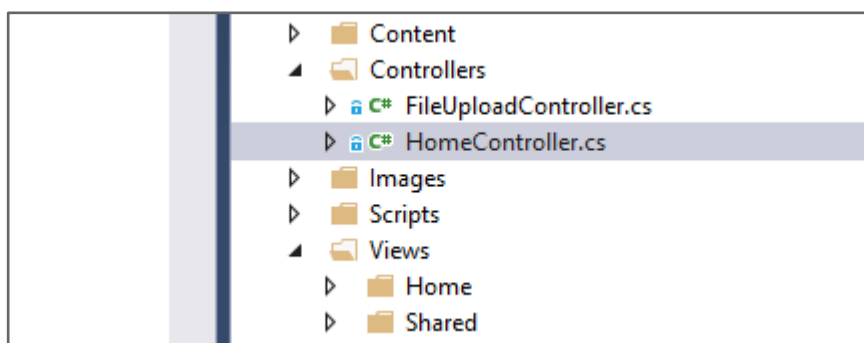


Fig. 5.6.1 - Screenshot fra dashboard-prosjektet i Visual Studio som viser mappeoppsettet til løsningen

Dashboardet er en ASP.NET MVC-applikasjon. De viktigste komponentene i dashboardet er mappene “Views”, “Content”, “Scripts” og “Controllers”. Views-mappen inneholder cshtml-sidene som blir vist på dashboard-websiden. Content-mappa spesifiserer designet til dashboardet, og inneholder CSS-filer og bakgrunnsbilder. Scripts-mappa inneholder alle JavaScript-filene. Controllers-mappen inneholder controller-filene som brukes fra server-siden av applikasjonen.

Under går vi igjennom noen deler av koden som er av interesse. Vi har gjort noen endringer på navn og data for å fjerne sensitiv informasjon.

### Views

#### **Shared layout**

Views-mappa består av en mappene “Home” og “Shared”. Shared-mappa inneholder en shared-layout fil kalt `_Layout.cshtml`, som inneholder felles html- og C#-kode for alle sidene i dashboardet. Den består av en head-tag som henter nødvendige biblioteker, og en navigation-bar og footer som vises på alle sidene. Fordelen med shared layout er at vi kan dele kode som kan gjenbrukes mellom forskjellige sider, og dermed slipper vi å kopiere den samme koden til flere sider.

```

<!-- Navigation bar: -->
<nav class="navbar navbar-expand-sm navbar-dark bg-dark">
  <a class="navbar-brand" href="~/Home/Home">Smartphone Dashboard</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse"
    data-target="#navbarSupportedContent"
    aria-controls="navbarSupportedContent" aria-expanded="false"
    aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse navbar-nav" id="navbarSupportedContent">
    <div class="navbar-nav">
      <a class="nav-item nav-link" href="~/Home/NewArticle">New Article</a>
      <a class="nav-item nav-link"
        href="~/Home/Notifications">Notifications</a>
      <a class="nav-item nav-link" href="~/Home/ListArticles">
        List articles
      </a>
      <a class="nav-item nav-link" href="~/Home/Settings">Settings</a>
    </div>
  </div>
</nav>

<!-- The current page will be rendered here -->
@RenderBody()

<!-- Footer: -->
<br/><br/>
<div class="footer">
  <hr class="hr-swadow"/><br/>
  <label>&nbsp;&copy; @DateTime.Now.Year - 99X</label>
  
</div>

```

## Home

Under Home-mappen ligger alle cshtml-sidene som man kan navigere til i dashboardet. Når man har logget inn er Home.cshtml den første siden man kommer til. Siden viser statistikk om artiklene som er publisert, for eksempel nyeste artikkel. Dette gir brukeren av systemet en overordnet forståelse av artiklene som har blitt publisert.

```

<div class="row">
  <div class="col-sm-1"></div>
  <div class="col-sm-3">
    Most recent article:
  </div>
  <div id="newest-article" class="col-sm-3"></div>
</div>

```

Når siden først lastes inn vil feltene for dataen først settes til "loading..." i JavaScript.

```
$("#number-articles").html("loading...");
$("#newest-article").html("loading...");
$("#next-to-expire").html("loading...");
```

Dette gjøres fordi det tar en stund før dataene blir hentet inn via et ajax-kall<sup>34</sup>.

```
$.ajax({
  url: 'https://server-url-here.no/api/article/',
  type: 'GET',
  success: successCallback,
  error: errorCallback,
  timeout: 10000,
  headers: { 'Authorization': '@System.Web.HttpContext.Current.Session["our-token-key"]'
}
});
```

For å kunne hente artiklene fra APIet kreves riktig autorisasjon. Når en administrator logger inn i dashboardet lages en sesjon<sup>35</sup> som inneholder et token. Tokenen hentes inn i koden og sendes med i ajax-kallet. Hvis ajax-kallet var vellykket returneres alle artiklene fra APIet til funksjonen `successCallback(elements)`. Funksjonen finner den nyeste artikkelen av de returnerte elementene og setter verdien i html feltet.

```
function successCallback(elements) {
  var mostRecentArticle = elements[0];
  var currentlyMostRecentDate = new
    Date(parseInt(mostRecentArticle.PublishDate.substr(6)));
  for (var i = 0; i < elements.length; i++) {
    var publishDate = new Date(parseInt(elements[i].PublishDate.substr(6)));

    // hvis nåværende publiseringsdato er nyere:
    if (publishDate < (new Date()) && publishDate > currentlyMostRecentDate) {
      mostRecentArticle = elements[i];
      currentlyMostRecentDate = new
        Date(parseInt(mostRecentArticle.PublishDate.substr(6)));
    }
  }
}
```

---

<sup>34</sup> Ajax er en teknikk for å sende http/https forespørsler asynkront; altså uten å måtte laste inn en ny side.

<sup>35</sup> En sesjon/session er et sett med data som lagres både i en nettleser og på en server, og tillater at en klient kan kommunisere med en server over en periode.

## New article

På siden `NewArticle.cshtml` kan man sende en ny artikkel til APIet fra et skjema. I skjemaet er det blant annet et dropdown element hvor man kan velge hvilken kanal man skal publisere artikkelen til (se 2.2 - New article).

```
<label>Select channel</label><br />
<select id="select-channel" name="Channel" class="form-control col-md-2">
  <option>Loading...</option>
</select>
```

Et ajax-kall henter kanalene fra APIet:

```
$.ajax({
  url: 'https://server-url-here.no/api/channel/',
  type: 'GET',
  success: successCallback,
  error: errorCallback,
  timeout: 10000,
  headers: { 'Authorization': '@System.Web.HttpContext.Current.Session["our-token-key"]'
}
});
```

Hvis ajax-kallet var vellykket får returneres alle kanalene fra APIet til funksjonen `successCallback(elements)`. Funksjonen fjerner så teksten "Loading..." og bytter ut med en liste av kanaler i dropdown-elementet.

```
function successCallback(elements) {
  var selectValues = "";

  for (var i in elements) {
    selectValues += "<option>" + elements[i].Name + "</option>";
  }

  $("#select-channel").html(selectValues);
}
```

## Content

Mappen "Content" inneholder alle CSS-filene, herunder biblioteket Bootstrap. Bootstrap brukes for å designe profesjonelle og responsive nettsider på en enkel måte. Vi har brukt Bootstrap for å unngå å bruke mye unødvendig tid på å lage egne CSS-klasser for å designe dashboardet, og fordi bruk av biblioteket gir et uniformt design likt slik vi forestilte oss utseende til dashboardet.

På "List articles"-siden bruker vi Bootstrap for å gjøre table-elementet responsivt, altså at kolonnene i tabellen vil endre størrelse basert på størrelsen på vinduet.

```
<div class="table-responsive-sm">  
  <table class="table">
```

I visse tilfeller var ikke Bootstrap tilstrekkelig; enten fordi vi ønsket en funksjonalitet som biblioteket ikke kunne tilby, eller for å endre på en funksjonalitet. Vi lagde derfor filen `site.css`. Her har vi blant annet definert bakgrunnsbildet og footer-elementet til siden.

```
body {  
  background-image: url(https://server-url-here.no/dashboard/Content/brushed.png);  
}  
  
.footer {  
  position: fixed;  
  left: 0;  
  bottom: 0;  
  width: 100%;  
  text-align: left;  
  background-image: url(https://server-url-here.no/dashboard/Content/brushed.png);  
}
```



## Scripts

“Scripts”-mappen inneholder JavaScript-biblioteker vi har tatt i bruk, og JavaScript-filer vi har skrevet selv. Vi har blant annet brukt bibliotekene jQuery, Bootstrap, og Modernizr.

jQuery er et bibliotek som forenkler funksjoner som brukes ofte i javascript, blant annet ajax-kall, som vi snakker om under “Views”-mappen.

Bootstrap består av CSS og javascript. Vi snakket om CSS-delen i “Content”-mappen.

Bootstrap bruker javascript for å gi funksjonalitet som man vil forvente å finne på en moderne nettside.

Modernizr er et bibliotek som sjekker hvilken versjon en nettleser er og utifra det finner ut hvilke biblioteker den er kompatibel med. Hvis en bruker har en eldre nettleser som ikke er kompatibel med den nyeste versjonen av bibliotekene vi bruker vil Modernizr finne en eldre versjon som nettleseren er kompatibel med.

De fleste sidene vi har i dashboardet bruker JavaScript som er spesifikt for den siden (koden ligger da i CSHTML-filen). I tillegg har vi en felles fil med funksjoner, kalt `form-logic.js`. Filen har blant annet funksjonalitet laget for skjemaene vi bruker på sidene. Ved å sette disse funksjonene i en felles fil slipper vi å gjenta samme kode på flere sider, for eksempel å sette formatet på date-time-picker (se 2.2 - New article).

```
$("#publish-date").datetimepicker({  
  format: "Y-m-d H:i",  
  minDate: 0,  
  minTime: 0,  
  lang: "en"  
});
```

Date-time-picker er en plugin som viser et dialogvindu hvor man kan velge dato og tid. Vi setter `minDate` og `minTime` til 0, som betyr at man ikke kan velge en dato som er i fortiden. Vi satte formatet på datoen til “Y-m-d H:i”, som for eksempel kan være “2018-05-17 12:00”.

## Controllers

“Controllers”-mappen inneholder controller-klasser (se 5.5. - Controllere og ruter) som ligger på serversiden, altså utilgjengelig for brukerne. Mappen består av controllerne

FileUploadController Og HomeController.

### File Upload Controller

FileUploadController håndterer opplasting av bilder som man kan ha med i artikler.

Controlleren består av en offentlig metode for opplasting av bilder, og en egendefinert StreamProvider-klasse, som brukes for å lagre bildefilen på serveren.

Når et bilde lastes opp kalles den asynkrone metoden `PostAsync()`. At metoden er asynkron betyr at man kan holde seg på samme side når man laster opp et bilde til en artikkel som skal publiseres. Vi har valgt denne metoden for å gjøre det mer brukervennlig ved å ha bilde-opplasting og publisering av artikler på samme side.

Metoden returnerer et `Task` objekt, som brukes til asynkrone operasjoner. Objektet inneholder en liste med to tekststrenger, der første streng er filplasseringen til bildet, og andre streng er en tilbakemelding som skal vises for brukeren.

```
public async Task<List<string>> PostAsync()
{
    if (Request.Content.IsMimeMultipartContent())
    {
        // ...
    }
}
```

Først sjekkes det om filen som har blitt sendt med er av typen MIME Multipart. MIME er en internett standard som tillater flere typer filer, i stedet for bare rene tekstfiler, noe som tillater binære filer som bilder[9]. Multipart spesifiserer at forespørselen som har blitt sendt inneholder flere deler, som kan være navnet på filen, filtypen, og selve bildefilen[10].

```

try
{
    string uploadPath = HostingEnvironment.MapPath("~/Images");
    CustomStreamProvider streamProvider = new CustomStreamProvider(uploadPath);
    await Request.Content.ReadAsMultipartAsync(streamProvider);
    FileInfo fileInfo = new FileInfo(streamProvider.FileData.First().LocalFileName);

    List<string> returnInfo = new List<string>
    {
        fileInfo.Name, fileInfo.Name + " (" + fileInfo.Length + " bytes) uploaded"
    };
    return returnInfo;
}

```

Videre hentes mappen hvor bildet skal lagres på serveren, og en egendefinert `StreamProvider` lages (se under). Etter filen har blitt lagret returneres en suksess-melding som inneholder filnavnet og størrelsen på filen. Hvis filen som ble lastet opp ikke var av et bildeformat kastes en `IOException`, og en feilmelding blir sendt til brukeren.

```

catch(System.IO.IOException)
{
    List<string> returnInfo = new List<string>
    {
        "", "You can only upload image files"
    };
    return returnInfo;
}

```

En ny instans av klassen `CustomStreamProvider` opprettes når et bilde skal lagres. I klassen sjekkes det om bildefilen er av riktig type.

```

string[] filePath = headers.ContentDisposition.FileName
    .Replace("\"", "").Split('.');
string fileName = filePath[filePath.Length - 2];
string extension = filePath[filePath.Length - 1];

// Checks if the file is not an image-type
if (!extension.Equals("png", StringComparison.OrdinalIgnoreCase)
    && !extension.Equals("jpg", StringComparison.OrdinalIgnoreCase)
    && !extension.Equals("jpeg", StringComparison.OrdinalIgnoreCase)
    && !extension.Equals("svg", StringComparison.OrdinalIgnoreCase)
    && !extension.Equals("gif", StringComparison.OrdinalIgnoreCase))
{
    return null;
}

```

Det hele filnavnet deles i to variabler; en for navnet og en for filendelsen. Så sjekkes det om filendelsen er av typen "png", "jpg", "jpeg", "svg", eller "gif". Hvis den ikke er det vil opprettelsen av `CustomStreamProvider` objektet feile og en `IOException` vil bli kastet.

## Home Controller

`HomeController.cs` håndterer navigasjon av sidene på dashboardet, samt kontroll over sesjoner. Når man navigerer til en side på dashboardet, for eksempel <https://webapi-app.99x.no/dashboard/NewArticle/>, vil en forespørsel sendes til Home Controller.

Kontrolleren inneholder en metode for hver side man kan navigere til i dashboardet, og vil returnere siden som blir forespurt.

```
public ActionResult NewArticle()
{
    if (IsLoggedIn()) return View();
    return RedirectToAction("Index");
}
```

I metoden gjøres en sjekk på om brukeren er logget inn. Hvis de er det får de tilgang til siden, hvis de ikke er det blir de omdirigert til innloggingssiden.

På innloggingssiden settes sesjoner hvis brukeren får logget inn. Da gjøres et kall til <https://webapi-app.99x.no/dashboard/SetUsernameSession?username=ola@eksempel.no>, hvor emailen sendes med som en parameter. Dette kaller på en metode i Home Controller:

```
public string SetUsernameSession(string username)
{
    System.Web.HttpContext.Current.Session["user_name"] = username;
    return username;
}
```

En sesjon settes med navn "user\_name", som blant annet kan brukes til å hente brukernavnet i dashboardet, som i innstillingssiden.

```
<p>Logged in as <b>@System.Web.HttpContext.Current.Session["user_name"]</b></p>
```

## 6 Oppsummering

I løpet av prosjektperioden har vi utviklet en mobilapplikasjon, med tilhørende serverløsning, for håndtering av henvendelser til servicedesken hos firmaet 99X. Våre fire fokusområder har vært:

- Prosjektstyring
- Sikkerhet
- Teknologier
- Brukervennlighet

Vi har hatt god nytte av inngående planlegging og detaljerte målbeskrivelser i oppstartsfasen av prosjektet. Investering av mye tid i planleggingsfasen dannet et solid grunnlag for et velfungerende samarbeid innad i gruppen, med de ansatte på 99X, og med veilederen hos OsloMet. Vi erfarte også at ikke alt kan planlegges, som for eksempel sykefravær, eller at vi ikke fikk tilgang til ressurser når vi hadde behov for det. Vi har lært at vi var nødt til å håndtere disse situasjonene og tilpasse planene underveis. I tillegg så vi hvor viktig god kommunikasjon og en strukturert arbeidsprosess var for at de planlagte målene skulle oppnås.

Underveis i utviklingen erfarte vi at prosessen med å autentisere sluttbrukere var mer innviklet enn først antatt. Vi la ned store ressurser knyttet til å hente inn informasjon relatert til autentisering, som medførte at vi fikk på plass vår autentiseringsmekanisme. I hele prosjektperioden har vi gjennomført sikkerhetssjekk av all kode ved å kontinuerlig gå gjennom og analysere kode med tanke på å avdekke sikkerhetshull og andre svakheter. Vi planla også å gjennomføre en ekstern sikkerhetsgjennomgang av prosjektet, men på grunn av at vår kontaktperson ble langtidssyk, ble dette dessverre ikke gjennomført.

Fra starten av prosjektet har vi hatt et fokus på å sette oss inn i nye teknologier, som blant annet Ionic, og verktøy brukt til autentisering. Vi har også ønsket å tilegne oss ny kunnskap knyttet til teknologier vi allerede hadde erfaring med, som ASP.NET MVC og ASP.NET Web API.

På grunn av prosjektets tidsramme viste Ionic seg å være et godt valg, ettersom applikasjonen som ble utviklet er kompatibel med både iOS og Android, som ga mulighet for å jobbe med samme kodebase for begge operativsystemene. Ionic tilfredstilte også vårt initielle krav til lisens, siden det er utgitt under en MIT-lisens. Bruken av ASP.NET Web API i samspill med smartphone-applikasjonen og løsningene for autorisering har også bidratt til å heve vårt kompetansenivå innenfor allerede kjente teknologier.

Det har vært viktig å teste brukervennligheten til produktet. En test internt hos 99X med et utvalg av de som kom til å benytte og administrere applikasjonen når den var ferdig utviklet ble derfor gjennomført. Vi ønsket at oppdragsgiveren skulle være fornøyd med produktet, og at de faktisk ville få bruk for det. Fra testen fikk vi konstruktive tilbakemeldinger og forbedringsforslag, noe som var et svært nyttig bidrag til utvikle en brukervennlig og intuitiv applikasjon.

I løpet av prosjektperioden har vi sett nytten av å ha en veldefinert kravspesifikasjon. Dette har hjulpet gruppen å holde fokus på hva som var mest essensielt for applikasjonen. I tillegg viste risikoanalysen seg å være verdifull, ettersom vi hadde en tiltaksplan når noen av risikoene inntraff. Gruppen har også sett verdien av det vi har lært oss i faget systemutvikling, da vi har hatt muligheten å benytte teknikkene vi lærte i en praktisk situasjon.

## 7 Referanser

- [1] "OWASP Top 10 - 2017", Owasp.org, 2017. [Internett]. Tilgjengelig: [https://www.owasp.org/images/7/72/OWASP\\_Top\\_10-2017\\_%28en%29.pdf.pdf](https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf). [Besøkt: 18- Mai- 2018].
- [2] "Core app quality | Android Developers", Android Developers, 2018. [Internett]. Tilgjengelig: <https://developer.android.com/docs/quality-guidelines/core-app-quality>. [Besøkt: 18- Mai- 2018].
- [3] "Security, Authentication, and Authorization with ASP.NET MVC", Docs.microsoft.com, 2018. [Internett]. Tilgjengelig: <https://docs.microsoft.com/en-us/aspnet/mvc/overview/security/>. [Besøkt: 18- Mai- 2018].
- [4] T. Joudeh, "ASP.NET Identity 2.1 with ASP.NET Web API 2.2 (Accounts Management) - Part 1 - Bit of Technology", Bit of Technology, 2015. [Internett]. Tilgjengelig: <http://bitoftech.net/2015/01/21/asp-net-identity-2-with-asp-net-web-api-2-accounts-management/>. [Besøkt: 18- Mai- 2018].
- [5] "The MIT License | Open Source Initiative", Opensource.org, 2018. [Internett]. Tilgjengelig: <https://opensource.org/licenses/MIT>. [Besøkt: 18- Mai- 2018].
- [6] "GNU General Public License v3.0", Choose a License, 2018. [Internett]. Tilgjengelig: <https://choosealicense.com/licenses/gpl-3.0/>. [Besøkt: 18- Mai- 2018].
- [7] "SAML Tokens and Claims", Docs.microsoft.com, 2018. [Internett]. Tilgjengelig: <https://docs.microsoft.com/>. [Besøkt: 18- Mai- 2018].

[8] "Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application (9 of 10)", Docs.microsoft.com, 2018. [Internett]. Tilgjengelig: <https://docs.microsoft.com/>. [Besøkt: 18- Mai- 2018].

[9] M. Rouse, L. Hoff, C. Kirsch and F. Norman, "What is MIME (Multi-Purpose Internet Mail Extensions)? - Definition from WhatIs.com", WhatIs.com, 2018. [Internett]. Tilgjengelig: <https://whatis.techtarget.com/definition/MIME-Multi-Purpose-Internet-Mail-Extensions>. [Besøkt: 18- Mai- 2018].

[10] M. Wasson, L. Latham, A. Pasic and T. Dykstra, "Sending HTML Form Data in ASP.NET Web API: File Upload and Multipart MIME", Docs.microsoft.com, 2018. [Internett]. Tilgjengelig: <https://docs.microsoft.com/en-us/aspnet/web-api/overview/advanced/sending-html-form-data-part-2>. [Besøkt: 18- Mai- 2018].



## 8 Vedlegg

### 8.1 Brukerhistorier

#### Smarttelefon-applikasjon

ID	Som <type bruker>	Vil jeg <Gjøre en oppgave>	Så jeg kan / slik at <nå et mål>
S0	Sluttbruker	Sende inn nye supporthenvendelser	Få support
S1	Sluttbruker	Få oversikt over pågående saker	Få med meg oppdateringer i saker
S2	Sluttbruker	Få oversikt over avsluttede saker	Gi beskjed hvis feilen gjenstår
S3	Sluttbruker	Sende inn svar på pågående saker	Svare eller oppdatere servicedesken
S4	Sluttbruker	Filtrere saker i lister	Enklere kunne finne saker
S5	Sluttbruker	Logge på med min firmabruker	Bruke applikasjonen
S6	Sluttbruker	Ringe til servicedesk med korrekt nummer	Få support
S7	Sluttbruker	Chatte med servicedesken	Få support
S8	Sluttbruker	Fjerne nyheter fra nyhetsstrøm	Holde applikasjonen oversiktlig
S9	Sluttbruker	Lese nyheter fra nyhetsstrøm	Få med meg oppdateringer om produkter og tjenester
S10	Sluttbruker	Tilbakestille passord	Jeg ikke trenger å kontakte servicedesken
S11	Sluttbruker	Lese notifikasjoner om kritiske hendelser	Få informasjon om kritiske hendelser
S12	Sluttbruker	Konfigurere applikasjonen	Velge instillinger som passer meg

## Dashboard-webapplikasjon

ID	Som <type bruker>	Vil jeg <Gjøre en oppgave>	Så jeg kan / slik at <nå et mål>
D0	Teknisk konsulent	Sende notifikasjoner om kritiske hendelser	Informere om kritiske hendelser og/eller vedlikehold
D1	Teknisk konsulent	Fjerne aktive notifikasjoner	Slik at brukere ikke trur at problemet gjenstår
D2	Teknisk konsulent	Se oversikt over aktive notifikasjoner	Få oversikt over aktive notifikasjoner
D3	Teknisk konsulent	Begrense mottakere av notifikasjoner	Meldinger kun går ut til relevante sluttbrukere
D4	Teknisk konsulent	Hente ut historikk og statistikk	Ta ut rapporter og analysere data
D5	Teknisk konsulent	Oppdatere pågående notifikasjoner	Brukere får oppdatert informasjon
D6	Teknisk konsulent	Bruke maler for vanlige hendelser	Spare tid og opprettholde standard
D8	Teknisk konsulent	Logge på utenfor firmanettverket	Utføre kritiske oppgaver når jeg ikke er på firmanett
D9	Markedsfører	Sende ut nyheter til sluttbrukere	Kunder får informasjon om nye produkter og tjenester
D10	Markedsfører	Editere sendte nyheter	Oppdateringer eller korrigeringer av feil kan utføres
D11	Markedsfører	Sette utløpstid på nyheter	Nyheter blir fjernet automatisk
D12	Markedsfører	Sette publiseringstid på nyheter	Nyheter blir publisert automatisk på riktig tid
D13	Markedsfører	Se oversikt over publiserte nyheter	Endre / sette innhold og datoer
D14	Markedsfører	Laste opp og/eller hente bilder fra galleri	Inkludere bilder i nyhetsutsendinger
D15	Administrator	Hente ut og se statistikk	Få oversikt over bruk av applikasjonen
D16	Administrator	Konfigurere standardmeldinger	Andre brukere kan bruke korrekte maler
D17	Administrator	Legge til og fjerne standardbilder	Kunden mottar standardiserte bilder

## 8.2 Møteforberedelse og referat

### Møtereferat fra oppstartsmøter:

# Forprosjektetmøte bachelor 2018

mandag 20. November 2017

#### Must have's

1. Newsfeed med push notifications (planerad service, incidenter osv)
2. Real time chat (jQuery / html)
3. Call servicedesk (baserat på kund)
4. Sända mail / saker till servicedesk
5. Rapporter för IT-brukere hos kunder
6. Beställa lösenord / låsa upp konton
7. Se status över egna saker
8. Knowledge-dokument-uploads (guider, etc.)
9. Logging av händelser (mail till Livetime)
10. Admin GUI (håndtering av informasjon i appen)
11. Språkändring

#### Nice to have's

1. RSS-feed från 99X
2. Div. schaner för beställning (bruker, utstyr, tillgångar, deaktivering, etc.)

#### Generellt

- Login via ADFS / Ad-user / email
- Låse kontoen til appen istedenfor i AD ved feil innlogging
- Möjligt att delegera roller / tillgångar via ADFS

#### Framtida möten

- Möte med Glenn ang. sikkerhet
- Möte med Sissel ang. GDPR / datalagring
- Möte med Line ang. Design

Oppstartsmøte 08.01.2018, 13:00 (Statement of Work / projekt-skiss osv.)

# Oppstartsmøte bachelor 2018

mandag 8. januar 2018

- Møte hver mandag med ukens arbeidsoppgaver
- Møte i slutten av uken på hvordan ukens arbeidsoppgaver har gått
- Base på 99X man-fre.
- Lage brukerguide på appen
- Norsk og engelsk

## Viktig i appen

- News feed
- Aktuelle saker
- Den aktuelle brukeren får kun opp info om sine arbeidsoppgaver
- Lage en demomodell for brukertest
- Teste demo modellen
- Fikse og sjekk lisenser
- 

## Dokumentasjon

- **Viktig og komme i gang tidlig**
- Sette av tid i slutten av uke for innspill om hva som skal legges til

## Møtereferat fra et vilkårlig møte:

# Møtereferat bachelor 2018

mandag 29. Januar 2018

### App autentisering

- Skal man logge inn i LiveTime etter man har logget inn i appen eller skal man kun sende brukeren rett inn i livetime ettersom de allerede er autentisering via ADFS
- Felles tjenestebruker per kunde for å autentisere / lese alle saker i livetime til tilhørende kunde?
  - Om bruker skal sende note kan de bruke mail
  - Gjemme alle andre saker enn de som tilhører autentisert bruker

### App

- Kan man se på sikkerheten på mobilen vs hvor lenge man er innlogget i appen?

### App -> LiveTime

- Må finne en metode å bruke ADFS mail / pålogging til å vise saker for den aktuelle innloggede
- filtrere søket på en smart måte ved bruk av den klonede databasen
- 

### Statistikk

- Integre så mye statistikk som mulig
- Filtrere meldinger/statuser som kan hentes til andre formål en appen basert på ADFS autentisering
- ADFS bruker som ikke finnes i LiveTime kan bli rapportert

### ToDo

- Porter
- OS / Windows
- bekrefte å få bruke DB til Marius / info og tilgang til server / rettigheter på bruker
- Finne ut av ADFS -> LT saker
- SKrive i SD at man ikke bruke livetime API, men i steden Dashboard replica DB
-

## Møteforberedelse til et vilkårlig møte:

# Møteforberedelse bachelor 2018

fredag 2. Mars 2018

### Webserver

- Hvordan skal vi autentisere mot app dashboardet internt?

### App pålogging

- Hvordan / hvor lenge skal man være pålogget appen?
- Skal vi basere oss på at alle logger seg på med email? Eller må vi ta hensyn til brukernavn og alle andre former for påloggingsid'er

### Browsersversjon

- Ønsker 99X at appen også skal være tilgjengelig i en "browser" versjon i tillegg til app? (brukere kan bruke den som en webpage)

### App->home

- Skal nyheter komme opp som egen side eller skal det vises i en modal / popup vindu?

### 99X app dashboard statistikk

- Skal statistikk bli sendt på mail til faste tider?

### Service desc

- Kan vi legge ved service desc i dokumentasjonen forutsatt at vi ikke tar med sensitiv data og får godkjenning

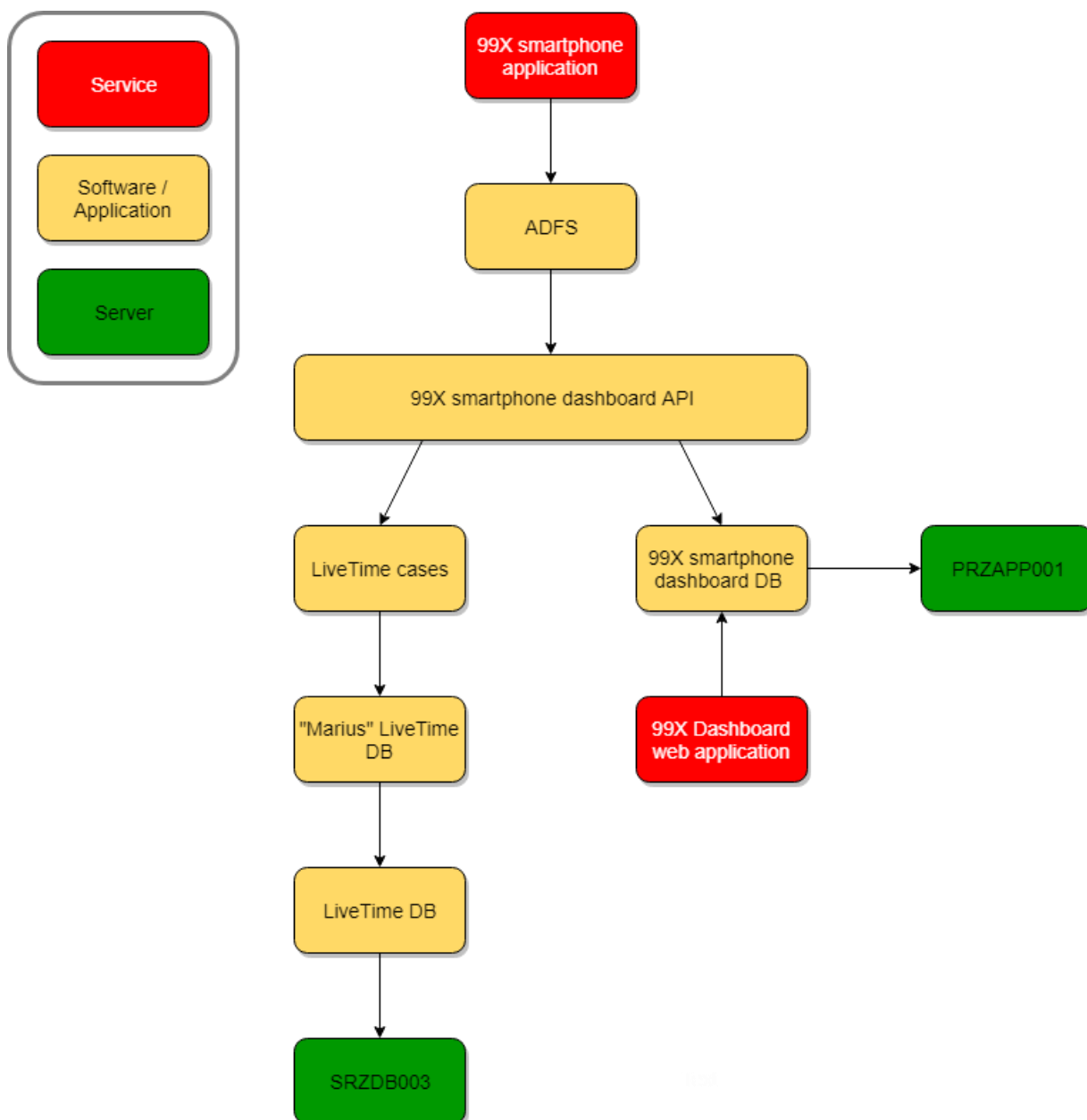
### Demonstrasjon

- Hvordan skal / kan vi demonstrer appen og dashboard? Bruke testmiljø?

### Publisering til butikker

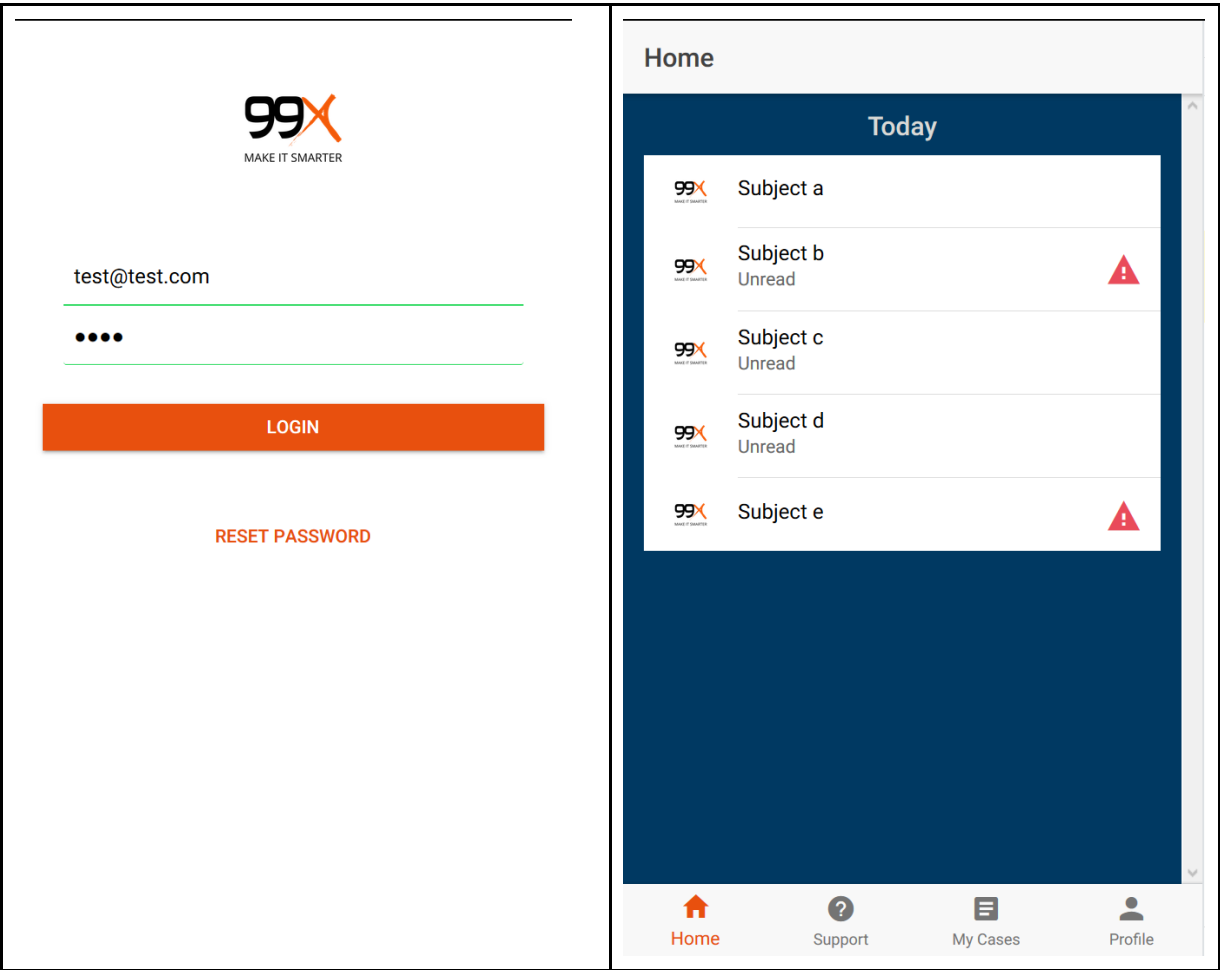
- Har 99X en Google-konto? evt lage en.
- Har 99X et d-u-n-s nummer? (AppStore)

Skisse etter møte med nettverk og sikkerhetsansvarlig:



# 8.3 Prototyper

## Mobilapplikasjon prototype:





## Support

Contact us

- Send us an email
- Give us a call
- Start a chat

Self service

- Reset my password
- Set email autoreply

## My Cases

Active Cases

- 544875** - I lost my nunchucks. I lost them som...
- 541876** - Where are my nunchucks????!?!?
- 546857** - Never mind - I found them. Close the ...
- 546879** - All these are added from array.

Closed Cases

- 546875** Something, something, nunchucks.

## Profile

About me

Name  
Bruce Lee

My contact info

Email  
bruce@lee.com

Phone  
+4744556677

My employment

Company  
Hollywood Inc.

Title  
Martial Artist

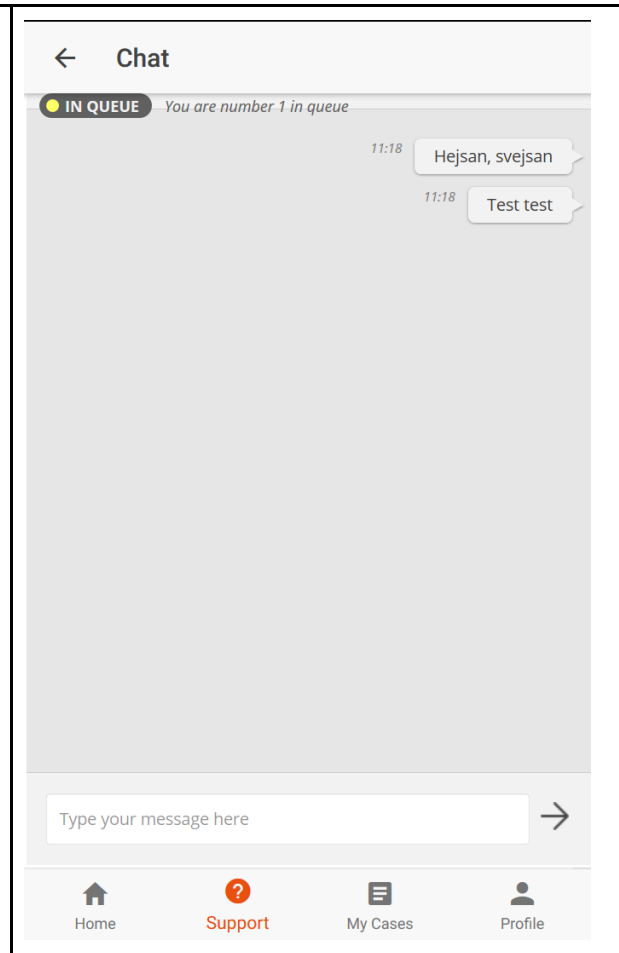
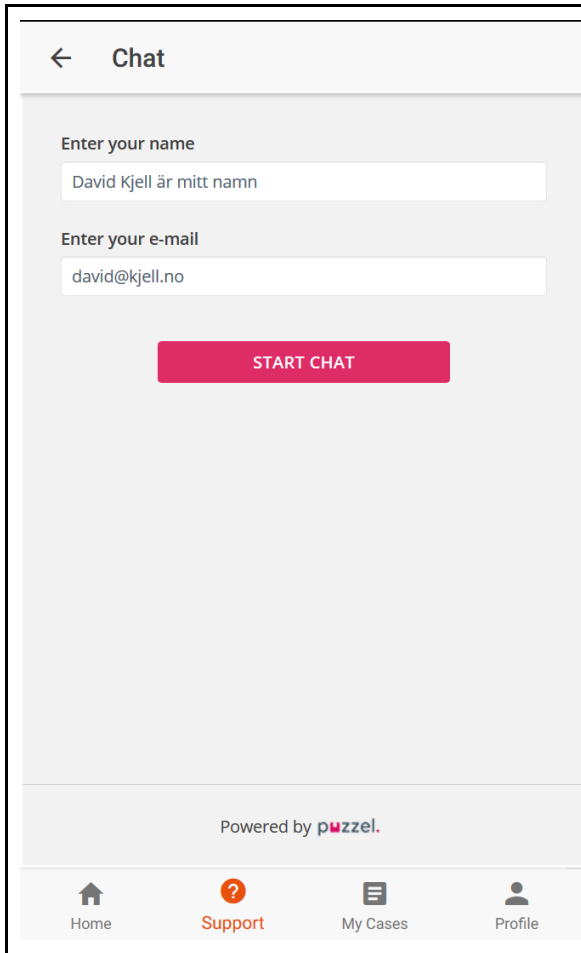
## ← Ticket 546857

Subject: Never mind - I found them. Close the ticket.

Created: 29.01.2018 11:16

Description

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Nisi quis eleifend quam adipiscing vitae proin sagittis nisl. Diam quis enim lobortis scelerisque fermentum dui faucibus in ornare. Ornare aenean euismod elementum nisi. Pellentesque massa placerat duis ultricies lacus sed. Magna sit amet purus gravida quis blandit turpis. Tincidunt arcu non sodales neque sodales ut etiam sit. Mollis nunc sed id semper risus in. Penatibus et magnis dis parturient. Egestas dui id ornare arcu odio ut sem. Ac tortor dignissim convallis aenean et tortor at. Posuere morbi leo urna molestie at elementum. Et netus et malesuada fames ac turpis. Ut morbi tincidunt augue interdum velit euismod in pellentesque. Pellentesque elit ullamcorper dignissim cras tincidunt lobortis.



Dashboard prototype:

**99X** **Dashboard for servicedesk app**

Logged in as **Elon Musk**

**New article to get published**

News title

Article text

Servers will go down for a short period because of maintenance.

Mark message as important

Choose a picture for the article (optional)

Browse...

Publish

Manage messages

## 8.4 Akseptansetest

### Mobilapplikasjon-scenarier:

Testpersonen skal:	Testpersonens tilbakemeldinger:
Resette passord på sin bruker via applikasjonen (fra påloggingskjermen)	<ul style="list-style-type: none"><li>• Logo på pålogging for liten</li><li>• Dårlig flyt i logoen og fargetema, ettersom adaxes ikke samsvarer med app logoen</li><li>• Hvilke passord skal resettes med reset knappen?</li><li>• Hvilke bruker og passord skal man logge på med? (tomy langengen)</li><li>• Pålogging: en bruker havnet inn på adaxes sin information screen gjennom å gå videre i web browseren</li></ul>
Logge på applikasjonen med sin AD bruker (fra påloggingskjermen)	<ul style="list-style-type: none"><li>• Knapp for å se passordet mens man taster inn</li></ul>
Finne frem til og få oversikt over pågående og avsluttede saker	<ul style="list-style-type: none"><li>• Ønsker ikke oversettelsealternativer når man kjører appen i "web view"</li><li>• Ønsker liste på aktive saker</li><li>• + for å lage ny sak har for mye fokus, heller en scrollbar på siden</li><li>• En mer intuitiv måte å legge til nye saker mener en bruker i testen, de andre brukerne er fornøyd med + for å legge til saker</li><li>• Saknummer har for mye fokus, kanskje bytte med subject på saken (mulig ha begge på samme linje)</li><li>• Fjerne # fra casenummer</li><li>• Closed case er bare lukket, status ikke viktig på lukkede saker (kanskje heller en dato for når den ble lukket)</li></ul>

<p>Finne følgende detaljert saksinformasjon:</p> <ul style="list-style-type: none"> <li>● Saksnummer</li> <li>● Emne</li> <li>● Spørsmålstekst</li> <li>● Notater</li> <li>● Nåværende status</li> <li>● Åpnet / lukket dato og tid</li> </ul>	<ul style="list-style-type: none"> <li>● Separere notes tydeligere per sak</li> <li>● Notes sortert etter dato</li> </ul>
<p>Filtrere saker, både åpne og avsluttede saker, ved hjelp av søkefeltet</p>	<ul style="list-style-type: none"> <li>● Finner ikke søkeknapp for filtrering (trykke på retur for å få bort keyboard)</li> <li>● Gi en følelse av å avslutte søket</li> <li>● Mer informasjon om hva den filtrerer (beskriver på en måte hva som kan søkes på)(søker ikke på innholdet, kunne vært mulig)</li> </ul>
<p>Sende inn en ny suppothenvendelse via applikasjonen</p>	<ul style="list-style-type: none"> <li>● Ta bilde i tillegg til attachment vedlegg</li> <li>● Bør fremgå / legges til informasjon om at signaturen følger med fra signaturen fra mailen på telefonen</li> </ul>
<p>Sende inn og legge til et nytt notat i en pågående sak</p>	
<p>Support siden</p>	<ul style="list-style-type: none"> <li>● Kan være link til 99x.no</li> <li>● Google maps for veibeskrivelse</li> <li>● Bør kanskje kalles kontakt istedenfor support</li> </ul>
<p>Send us an support email</p>	<ul style="list-style-type: none"> <li>● Bør fremgå om det det er support mail eller vanlig henvendelse</li> </ul>
<p>Ring til servicedesk via applikasjonen (åpne telefonens ringe-funksjonalitet)</p>	<ul style="list-style-type: none"> <li>● Give us a call (er det support eller sentralbordet?)</li> </ul>

med ferdig utfylt nummer)	
Åpne en chat sesjon og kommunisere med 99X sin servicedesk	<ul style="list-style-type: none"> <li>• Ta med navnet videre fra appen og inn i chat så den er automatisk utfylt</li> <li>• Iphone - ingen kryss for å avslutte chaten (må legge inn et tast avslutning)</li> <li>• Ønsker samtalelogg (det kommer vel opp etter man trykker avslutt)</li> </ul>
Homescreen	<ul style="list-style-type: none"> <li>• Link til 99X salg /</li> <li>• En setning</li> </ul>
Lese nyheter i fra nyhetsstrømmen, for så å fjerne nyheten fra nyhetsstrømmen	<ul style="list-style-type: none"> <li>• Bør være informasjon som gir kunden hint om at det kan leses mer en bare headingen i nyheter</li> <li>• Angreknapp for å angre sletting av nyheter</li> <li>• Legge til fjernet / slettet nyheter ved hjelp av vis slett artikkel</li> <li>• Bør fremgå tydeligere at man kan fjerne nyheter</li> <li>• Kunne se et arkiv over mottatte meldinger fra en viss periode tilbake i tid etter "expiry-date" har gått ut fra artikkelen</li> </ul>
Finne og lese kritiske hendelser	<ul style="list-style-type: none"> <li>• Får ikke trykket på push notifikasjonen og lest mer (urent message)</li> </ul>
Finne frem til innstillinger i applikasjonen	<ul style="list-style-type: none"> <li>•</li> </ul>
Logge ut av applikasjonen	<ul style="list-style-type: none"> <li>•</li> </ul>

#### Generelt

- Får ikke status bar når appen er åpnet (Dette fungerer i appen når det ikke er test på Ad sin tlf)

#### Introduksjon

- Lite tekst, my illustrasjon

### “Nice to haves”

#### My cases

- Felles bruker for store IT avdelinger var et ønske fra en deltager

### Dashboard-scenarioer:

Testpersonen skal:	Testpersonens tilbakemeldinger:
Logge på smartphone-dashboardet med sin AD bruker (fra påloggingsskjermen)	•
Publisere en artikkel med følgende: <ul style="list-style-type: none"> <li>• Tittel og innhold</li> <li>• “publish-date” og “expiry-date”</li> <li>• Laste opp et bilde eller link</li> </ul>	• Døpe om nyheter til ny artikkel og endre posisjon
Finne frem til og se en oversikt over allerede publiserte artikler / nyheter	•
Redigere “expiry-date” på en allerede publisert artikkel / nyhet	•
Slette en vilkårlig nyhet fra oversikten over publiserte artikkel / nyheter	•

Se statistikk over antall publiserte artikler og hvilke artikkel som sist ble publisert	•
---	---

#### Generelt

- Sorter kunder og innenfor kunden sortere på segment (grupper, som på dagen SMS)



Spørreundersøkelse til testing

## Smartphone App

1) Hva synes du om designet?

2) Syns du brukergrensesnitt ser bra ut og er lett å jobbe med?

3) Er det noe i applikasjonen som ikke fungerer slik du hadde sett for deg? Hvis ja, forklar.

4) Hvor lett syns du applikasjonen var å bruke? På en skala fra 1-10 (Der 1 er veldig vanskelig, og 10 er veldig lett i bruk).

5) Var det noe funksjonalitet du kunne ønsket å ha med?

6) Nevn tre ting du ville forbedret i applikasjonen.

7) Andre kommentarer til applikasjonen som du mener kan være nyttig til å forbedre produktet.

## Dashboard

1) Hva synes du om designen i dashboardet:

2) Var det noe du synes var lite intuitivt i dashboardet?

3) Var det noe funksjonalitet du kunne ønsket å ha med?

4) Andre kommentarer til dashboard som du mener kan være nyttig til å forbedre produktet.

## 8.5 Risikoer

Tabellene med risikoer er sortert etter prioritet. Prioriteten er et resultat av hvor stor sannsynlighet noe har for å inntreffe og hvor stor konsekvens det blir vurdert å ha for prosjektet og produktet.

Forkortelsen SKR står for sannsynlighet, konsekvens og risikofaktor.

Prioritet - Kategori	R01 - Sikkerhet
Beskrivelse	Applikasjonen er ikke sikker i produksjon
Årsak	Sikkerhetshull er ikke oppdaget under testing/utvikling
SKR	$5 \times 5 = 25$
Tiltak	Enkel og oversiktlig kode, sikkerhetsgjennomgang og testing, restriktiv tilgjengeliggjøring av ressurser

Prioritet - Kategori	R02 - Integrasjon
----------------------	-------------------

Beskrivelse	Vanskeligheter med å integrere app med dashboard/Livetime
Årsak	Dårlig dokumentert/konstruert API
SKR	$4 \times 4 = 16$
Tiltak	Kontakt med utvikleren av aktuelt system

Prioritet - Kategori	R03 - Tidsestimering
Beskrivelse	Estimere for mye eller for lite tid til prosjektet
Årsak	Uventede/tidkrevende arbeidsoppgaver som oppstår underveis
SKR	$5 \times 3 = 15$
Tiltak	Revurdering av allokert tid til oppgaver

Prioritet - Kategori	R04 - Utviklingsmiljø
Beskrivelse	Ressurser hos oppdragsgiver er ikke tilgjengelige
Årsak	Dårlig oppfølging av nøkkelpersoner hos oppdragsgiver
SKR	$3 \times 5 = 15$
Tiltak	Purringer, statusmøter, tidlig oversikt over nødvendige systemtilganger og ressurser

Prioritet - Kategori	R05 - Kravspesifikasjon
Beskrivelse	Mangler eller endringer i kravspesifikasjon
Årsak	Ikke tydelig definert kravspesifikasjon
SKR	$4 \times 3 = 12$
Tiltak	Tidlig og tydelig utarbeidelse av spesifikke krav

Prioritet - Kategori	R06 - Lisens
Beskrivelse	Verktøy har lisenser som resulterer i kostnader for bedrift
Årsak	Manglende støtte fra oppdragsgiver å tilby nødvendige lisenser

SKR	$2 \times 5 = 10$
Tiltak	Bruk av Open Source-verktøy, enighet med bruk av verktøy med oppdragsgiver

Prioritet - Kategori	R07 - Kommunikasjon
Beskrivelse	Mangelfull kommunikasjon med oppdragsgiver
Årsak	Dårlig oppmøte, prioritering av andre arbeidsoppgaver
SKR	$2 \times 4 = 8$
Tiltak	Rådgiving med veileder og/eller kontaktpersoner hos firmaet

Prioritet - Kategori	R08 - Kunnskap
Beskrivelse	Mangel på kunnskap om teknologi som skal benyttes
Årsak	Mangel på erfaring, erfaring, tid, dokumentasjon
SKR	$2 \times 3 = 6$
Tiltak	Begynne tidlig å se på teknologier vi skal ta i bruk, eksperthjelp

Prioritet - Kategori	R09 - Langvarig fravær
Beskrivelse	Gruppemedlem blir utilgjengelig over en lenger periode
Årsak	Sykdom eller død hos gruppemedlem eller nøkkelperson hos 99X
SKR	$1 \times 5 = 5$
Tiltak	Restrukturering av ansvarsområder

Prioritet - Kategori	R10 - Kommunikasjon
Beskrivelse	Mangelfull kommunikasjon internt i gruppen
Årsak	Dårlig oppmøte, useriøse møter, lite bruk av hjelpemidler/verktøy
SKR	$1 \times 5 = 5$
Tiltak	Dagbøter, ekskludering ur grupp, kontakt med veileder/administrasjon

Prioritet - Kategori	R11 - Teknologi
Beskrivelse	Utfasing (deprecation) av anvendt teknologi
Årsak	Komponenter er ikke lenger supporterte av operativsystemer
SKR	$1 \times 5 = 5$
Tiltak	Bruk av kjente, aktive og hyppig oppdaterte produkter

Prioritet - Kategori	R12 - Kommunikasjon
Beskrivelse	Mangelfull kommunikasjon med veileder
Årsak	Dårlig oppmøte og/eller oppfølging
SKR	$1 \times 4 = 4$
Tiltak	Kontakt med studieadministrasjon

Prioritet - Kategori	R13 - Motivasjon
Beskrivelse	Forskjellig arbeidsinnsats internt i gruppen
Årsak	Forskjellig bakgrunn, livssituasjon, innstilling og kompetanse
SKR	$1 \times 4 = 4$
Tiltak	Deling av kompetanse, hjelp ved vanskeligheter, sosiale aktiviteter