

**BACHELORPROSJEKT VÅR 2021**  
**Kunnskapsportalen Badminton**



**Norges Badmintonforbund**

Kunnskapsportal og kommunikasjonsplattform for  
badmintonspillere over hele landet

**OSLOMET****Institutt for Informasjonsteknologi**

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

PROSJEKT NR.

29

TILGJENGELIGHET

Åpen

Telefon: 22 45 32 00

# BACHELORPROSJEKT

HOVEDPROSJEKETS TITTEL  Kunnskapsportalen Badminton	DATO  25.05.2021
	ANTALL SIDER / BILAG  191
PROSJEKTDELTAKERE  Pia Karoline Aamodt (s333976) Henrik Nøkleby Hjellup (s308948) Erik Skrautvol Larsen (s325882) Tommy Pedersen (s306650) Sepideh Tajik (s329328)	INTERN VEILEDER  Roza Abolghasemi

OPPDRAKGIVER  Norges Badmintonforbund	KONTAKTPERSON  Charlotte Støelen
---	--

**SAMMENDRAG**

Bachelorprosjektet Kunnskapsportalen Badminton er gjennomført i samarbeid med Norges Badmintonforbund. Kunnskapsportalen Badminton er en responsiv webapplikasjon med mål om å formidle kunnskap om badminton og skape en felles platform for badmintoninteresserte i hele Norge.

**3 STIKKORD**

Webapplikasjon

React

Azure DevOps

## Forord

Dette dokumentet består av en samling dokumenter produsert under bachelorprosjektet Kunnskapsportalen Badminton vårsemesteret 2021 ved Oslo Metropolitan University. I samarbeid med Norges Badmintonforbund har denne gruppa på 5 studenter fra studiene anvendt datateknologi og dataingeniør skapt en webapplikasjon som kan bidra til å etablere kontakt mellom badmintonspillere og badmintoninteresserte over hele landet.

Vi vil takke Charlotte Støelen for å ha vært en god veileder. Selv om hun ikke hadde den tekniske kompetansen til å dirigere oss gjennom prosjektet, har hun alltid vært flink til å kommunisere og kommet med gode innspill til hvordan webapplikasjonen skulle fungere. Vi vil også takke Norges Badmintonforbund for å gi oss muligheten til å bruke det vi har lært gjennom tre års utdanning i en virkelighetsnær situasjon.

**Kapittel 1 Presentasjon**

**Kapittel 2 Prosessdokumentasjon**

**Kapittel 3 Kravspesifikasjon**

**Kapittel 4 Produktdokumentasjon**

**Kapittel 5 Resultat av brukertesting**

**Kapittel 6 Brukermanual**

**Kapittel 7 Referanser**

**Kapittel 8 Ordliste**

Kapittel 1

# Presentasjon

# Innhold

<b>Presentasjon .....</b>	<b>1</b>
<b>1.1 Presentasjon.....</b>	<b>3</b>
1.1.1 Bakgrunn for prosjekt.....	3
1.1.2 Om oss .....	5
1.1.3 Om oppdragsgiver .....	6
1.1.4 Faglige forutsetninger .....	7
1.1.5 Kort beskrivelse av løsning .....	9
1.1.6 Konklusjon .....	9

## 1.1 Presentasjon

I dette dokumentet presenteres bakgrunnen for prosjektet, oss som gruppe, oppdragsgiver og mer.

### 1.1.1 Bakgrunn for prosjekt

---

Norges Badmintonforbund hadde et ønske om å samle kommunikasjon blant deres medlemmer på én plattform - i motsetning til dagens situasjon, hvor medlemmer kommuniserer med hverandre på mange forskjellige plattformer.

#### 1.1.1.1 Bakgrunn

Norges Badmintonforbund (NBF) hadde et ønske om å skape en ny plattform for badmintonspillere og entusiaster for badmintonporten. Dermed ble det aktuelt å sende et prosjektforslag til bachelorstudenter ved Oslo Metropolitan University, se figur 1.1.1.1a. Dette prosjektforslaget skapte en unik sjanse for studenter til å teste egenskapene de har tilegnet seg gjennom tre år på bachelorstudiet i diverse datateknologifag.

#### Kunnskapsportal Badminton

Målet er å utvikle en plattform på web og app som skal kunne bidra til et kunnskapsløft og erfaringsutveksling hos badmintonmedlemmer i Norge. Plattformen skal kunne brukes til å opprette nettverk samt dele/ta i bruk nye/eksisterende treningsprogram og øvelser. Det skal også være mulig å dele inn informasjon etter nivå, kategori og lignende.

Følgende spesifikasjoner er å foretrekke, men kan endres i samarbeid med studenten:

- Brukervennlig
- Flere brukere
- Mulighet for å bygge nettverk/samtaler (inspirasjon: Slack)
- Støtter video, bilde og tekst (inspirasjon: skadefri)

Dette er altså ideen vår, og vi håper naturligvis at dette er noe studenten vil være med å utvikle til et produkt i samarbeid med aktuelle styremedlemmer og meg som prosjekteier.

Kontaktinformasjon:  
Charlotte Støelen  
Mail: [Charlotte.stoelen@badminton.no](mailto:Charlotte.stoelen@badminton.no)  
Tlf: 97180074

*Figur 1.1.1.1a: Utklipp av NBF sitt prosjektforslag på bachelorprosjektets nettside*

Plattformen NBF ønsker seg skal være med på å etablere kommunikasjon mellom badmintoninteresserte over hele landet gjennom et diskusjonsforum. Men den skal også inneholde en kunnskapsdel der administrator kan poste videoer og dokumenter som brukere skal kunne laste ned og se på. Disse videoene og dokumentene skal ta for seg innhold som for eksempel korrekt treningsteknikk eller oversikter over badmintonkamper.

### **1.1.1.2 Problemstilling**

Dagens situasjon er at NBF betaler flere titalls tusen kroner i året til en tredjepart for kun videotjenesten nevnt ovenfor. Hensikten med dette prosjektet er å gi NBF en felles plattform som vil samle all kommunikasjon og visning av videoer uten involvering av en tredjepart. Dette er ikke den eneste grunnen til at dette prosjektet ble aktuelt. Flere personer fra Norges Badmintonforbund har uttalt seg rundt hvorfor dette prosjektet ble relevant. Oppsummert er det et generelt ønske om å skape en plattform for samhandling, erfaringsutveksling og inspirasjon for badmintonsporten. Det var også et håp om å motivere flere til å melde seg på trenerkurs, blant annet fordi mindre sportsklubber ofte mangler frivillige eller i det hele tatt gode trenere, noe som fører til at man ”blir stående alene på trening uten øvelser/kompetanse til å utføre øvelser [sic]”. Foreløpig har klubbene i Norges Badmintonforbund kun frivillige i administrasjonen, og de tror at en god delingskultur kan gi et bedre utgangspunkt for at flere frivillige trenere skal bli inspirert til å lære bort kunnskapen sin.

## 1.1.2 Om oss

---

Gruppen består av tre studenter fra dataingeniør og to studenter fra Anvendt datateknologi. Ulike faglige bakgrunner gir oss et bredt kunnskapsområde som vi har brukt gjennom hele prosjektet.

### **Pia Karoline Aamodt (s333976)**

Pia studerer Anvendt Datateknologi, og hun er glad i web- og interaksjonsdesign. I tillegg ønsker hun at alle brukere skal ha en tilfredsstillende opplevelse gjennom bruk av et system. Dermed ble det naturlig at Pias rolle i prosjektet ble som en av frontend-utviklerne.

### **Henrik Nøkleby Hjellup (s308948)**

Henrik er dataingeniørstudent. Han begynte helt i begynnelsen å sette seg inn i backend, men ble etter hvert med på å bistå frontend-utviklingen pga. arbeidsmengden der.

### **Erik Skrautvol Larsen (s325882)**

Erik er dataingeniørstudent. Han har jobbet som frontend-utvikler i dette prosjektet, hovedsakelig med samhandling mellom frontend og backend.

### **Tommy Pedersen (s306650)**

Tommy er dataingeniørstudent, og er den eneste i gruppen som hadde webapplikasjoner som valgfag. Det ble derfor naturlig at han påtok seg ansvaret for utviklingen av backend-systemet.

### **Sepideh Tajik (s329328)**

Sepideh studerer Anvendt Datateknologi og er frontend-utvikler i prosjektet. Hun har stor lidenskap for programmering, og trives med å utvikle løsninger som er universelt utformet.

## 1.1.3 Om oppdragsgiver

---

Norges Badmintonforbund er gruppas oppdragsgiver for bachelorprosjektet. De er høyeste myndighet på badmintonsportens område i Norge og er medlem i Badminton World Federation (BWF), Badminton Europe, samt Norges Idrettsforbund.

### **Norges Badmintonforbund**

Gruppas oppdragsgiver er Norges Badmintonforbund ble etablert i 1938 og har i dag registrert rundt 5500 aktive medlemmer fordelt på 115 klubber i Norge.

### **Charlotte Støelen**

Charlotte Støelen er gruppas eksterne veileder. Hennes ansvar i Norges Badmintonforbund er utviklingskonsulent med ansvar for utviklingsprosjekter for å tilrettelegge enda mer badmintonaktivitet i Norge. I tillegg er hun ansvarlig for IKT i forbundet.

## 1.1.4 Faglige forutsetninger

---

Gjennom 3 år har gruppa opparbeidet en bred, generell kunnskap om IT og datateknologi som gjør oss i stand til å produsere det produktet Norges Badmintonforbund ønsker seg. Én av oss har tatt faget ITPE3200 Webapplikasjoner, som er nærmest knyttet til oppdraget vi har fått tildelt. Andre fag, som for eksempel Systemutvikling, har også vært avgjørende for vår utvikling av produktet.

### **DAFE1200: Webutvikling og inkluderende design**

“Web og internett har i dag en viktig og sentral samfunnsmessig rolle og er av fundamental betydning for de som skal ha databehandling og IT som profesjon. I dette emnet vil web være en plattform for å etablere kunnskap om og ferdigheter i ideer, teknologi og metodikk som er sentrale for yrkesområdet data og IT.” (Oslo Metropolitan University, 2020a)

DAFE1200 var et introduksjonsfag til webutvikling høsten 2018. Dette studiet ga en innføring i webutvikling og HTML, CSS samt JavaScript, noe som naturligvis har vært svært viktig og noe av det mest grunnleggende for utviklingen av webapplikasjonen i samarbeid med Norges Badmintonforbund.

### **DAFE2200: Systemutvikling**

“I dette emnet skal studenten utvikle kunnskap, ferdigheter og generell kompetanse knyttet til utvikling av programvaresystemer. Studenten skal få innsikt i hvordan systemenes egenskaper defineres, hvilke rammer som gjelder for utviklingen, og hvordan utviklingsprosessen ledes. Videre skal studenten kunne forstå noe av kompleksiteten i samspillet mellom programvaresystemer og ulike bruker- og interessegrupper. Studenten skal forstå essensen i og utvikle en kritisk sans for vurdering av både moderne (inkludert smidige) og tradisjonelle metoder og teknologier for systemutvikling.” (Oslo Metropolitan University, 2020b)

DAFE2200 har vært helt avgjørende for bachelorprosjektet; det ga en dypere innsikt i hvordan et system bygges opp gjennom en prosess, noe som uten tvil har vært en viktig faktor gjennom hele prosjektet. Kunnskap om forskjellige utviklingsmetoder som Scrum og Kanban, samt forskjellige diagram som klassediagram, use case-diagram med mer var en del av dette emnet, og disse ble svært nyttige gjennom utviklingsprosessen.

## DATS1500: Databaser

"Studentene skal tilegne seg kunnskaper om databasedesign og ferdigheter i bruk av relasjonsdatabaser. Videre skal de utvikle ferdigheter i konstruksjon og vedlikehold av databaser samt innsikt i flerbruker og flerlags databasearkitektur. De skal få kjennskap til XML og innsikt i programmering mot databaser." (Oslo Metropolitan University, 2016)

Databaser har vært en naturlig del av prosjektet for å blant annet oppbevare informasjon om brukere, dokumenter, forumposter og lignende. Kunnskapen tilegnet i emnet DATS1500 har derfor vært av stor viktighet for bachelorprosjektet.

## DATA1700: Webprogrammering

"Studentene skal tilegne seg grunnleggende kunnskaper om webservere og utvikle ferdigheter og innsikt i full-stack programmering for web. Videre skal de utvikle ferdigheter og innsikt i programmering av dynamisk websider som kommuniserer med database. De skal også få kjennskap til informasjonssikkerhet i webdesign." (Oslo Metropolitan University, 2020c)

I DATA1700 var et av målene å få frontend til å kommunisere med backend, samt skape dynamiske websider og gi en dypere forståelse av JavaScript og diverse tilhørende bibliotek. For bachelorprosjektet har det å få backend og frontend til å fungere sammen vært en grunnleggende faktor for å få en fungerende webapplikasjon.

## ITPE3200: Webapplikasjoner

"Studentene skal tilegne seg kunnskap og innsikt i aktuelle teknologier og teknikker som benyttes i næringslivet til utvikling av avanserte web-applikasjoner. Emnet vil være et nyttig fundament for studenter som ønsker å arbeide med web-applikasjoner i forbindelse med bacheloroppgaver." (Oslo Metropolitan University, 2020d)

ITPE3200 har vært sentralt for utviklingen av backend-systemet, da dette faget ga en dypere innsikt i hvordan en webapplikasjon er strukturert og hvordan databaser håndteres i systemet. Faget var også en god introduksjon til C# kodespråket.

## 1.1.5 Kort beskrivelse av løsning

---

Oppsummert er løsningen en webapplikasjon som er et forum og kunnskapsportal i én - badmintonspillere og badmintonentusiaster kan diskutere relevante temaer i badmintonmiljøet med hverandre, og Norges Badmintonforbund kan dele kunnskap om badmintonsporten, som for eksempel korrekt utførelse av diverse øvelser og god teknikk.

## 1.1.6 Konklusjon

---

Vår gruppe består av en gjeng mennesker med forskjellige egenskaper og utgangspunkt som gjør at vi egner oss godt til å gjennomføre dette prosjektet. Gjennom utviklingen av webapplikasjonen har vi lært mye vi alle kan ta med oss videre i livet, både yrkesmessig og på et mer personlig plan.

Kapittel 2

# **Prosessdokumentasjon**

# Innhold

<b>Prosessdokumentasjon .....</b>	<b>1</b>
<b>2.1 Planlegging og metode .....</b>	<b>3</b>
2.1.1 Planleggingsprosess .....	3
2.1.2 Metode .....	29
2.1.3 Verktøy og språk.....	33
<b>2.2 Utviklingsprosess .....</b>	<b>36</b>
2.2.1 Prosjektet gjennom fasene .....	36
<b>2.3 Om kravspesifikasjonen.....</b>	<b>43</b>
2.3.1 Kravspesifikasjonens rolle .....	43
2.3.2 Endringer i kravspesifikasjonen .....	44
<b>2.4 Testing .....</b>	<b>46</b>
2.4.1 Hvorfor teste? .....	46
2.4.2 Enhetstesting. ....	46
2.4.3 Tilgjengelighetstesting. ....	51
<b>2.5 Resultat av prosjekt .....</b>	<b>55</b>
2.5.1 Endelig produkt.....	55
<b>2.6 Avslutning.....</b>	<b>60</b>
2.6.1 Refleksjon og konklusjon .....	60

## 2.1 Planlegging og metode

I et så viktig og avgjørende prosjekt som bachelorprosjektet, er det høyst nødvendig med en plan som beskriver hvordan prosjektet skal gjennomføres. I denne seksjonen går vi dypere inn i hvorfor vi valgte akkurat dette prosjektet, hvordan vi planla det, samt hvilke verktøy vi brukte.

### 2.1.1 Planleggingsprosess

---

Planleggingsfasen av prosjektet startet i praksis allerede da vi fikk første kontakt med Norges Badmintonforbund i oktober 2020. På grunn av Covid-19 måtte hele prosjektet skje over nett, noe som kompliserte situasjonen, men gjennom godt samarbeid og kommunikasjon klarte gruppen å lage en oversikt for hvordan prosjektet skulle gjennomføres.

#### 2.1.1.1 Oppstartsfasen

Oppstartsfasen begynte allerede i midten av høstsemesteret 2020. Etter å ha etablert oss som gruppe, sendte vi ut e-poster til mange potensielle oppdragsgivere, både de som hadde lagt ut prosjektforslag på bachelorprosjektets nettside og andre firmaer som vi tenkte kunne være aktuelle. Norges Badmintonforbund var en av mange gruppa kontaktet, og en av de som hadde lagt ut et prosjektforslag. Den første responsen kom kort tid etter at vi sendte den første e-posten 23. oktober 2020, se figur 2.1.1.1a.

Pia Karoline Aamodt  
fr. 23.10.2020 12:55  
Til: Charlotte.stoelen@badminton.no <Charlotte.Stoelen@badminton.no>

Heisann!

Vi er en gruppe på 5 studenter fra OsloMet som er interesserte i "Kunnskapsportal Badminton" til vårt endelige bachelorprosjekt våren 2021. Gruppa vår består av 2 anvendt datateknologistudenter og 3 dataingeniørstudenter, med andre ord har vi et bredt kunnskapsfelt innen web- og datateknologi. Vi håper at dere vil vurdere oss til denne oppgaven, og vi venter spent på tilbakemelding!

Med vennlig hilsen  
 Pia Karoline Aamodt - [s333976@oslomet.no](mailto:s333976@oslomet.no)  
 Henrik Nøkleby Hjellup - [s308948@oslomet.no](mailto:s308948@oslomet.no)  
 Erik Skrautvol Larsen - [s325882@oslomet.no](mailto:s325882@oslomet.no)  
 Tommy Pedersen - [s306650@oslomet.no](mailto:s306650@oslomet.no)  
 Sepideh Tajik - [s329328@oslomet.no](mailto:s329328@oslomet.no)

Figur 2.1.1.1a: Utklipp av den første e-posten til Norges Badmintonforbund

Etter flere utvekslinger med Norges Badmintonforbund sin representant, Charlotte Støelen, hadde gruppa klart å sikre et bachelorprosjekt. Ikke lenge etter satte gruppa opp et møte med Charlotte via Microsoft Teams, slik at vi kunne bli kjent med hverandre og få en nærmere idé av hva oppdragsgiveren faktisk ønsket seg. Ut fra dette kunne gruppa lage prosjektskissa, det dokumentet som definerer prosjektets idé nærmere.

### 2.1.1.2 Arbeids- og fremdriftsplan

For å kunne planlegge prosjektet nærmere, satte vi opp en arbeids- og fremdriftsplan slik at vi kunne få en generell oversikt over hvordan prosjektet skulle utføres i praksis, se figur 2.1.1.2a.

	04.01	Uke												25.05		
Aktivitet	1	2	3	4	5	6-8	9 *	10-12	13 *	14-16	17 *	18	19	20	21	22
Forprosjektrapport																
Prosjektplanlegging																
Design og utkast																
Utvikling og implementering																
Testing og forbedring																

### *Figur 2.1.1.2a: Prosjektets arbeids- og fremdriftsplan*

Planen tok utgangspunkt i prosjektets offisielle begynnelse, altså tidlig januar 2021 og til prosjektets innlevering, 26. mai 2021. Prosjektet hadde 3 definerte milepæler, disse utdypes nærmere i kapittel 2.2.

### 2.1.1.3 Risikoanalyse

Det er forskjellige typer risiko assosiert ved all systemutvikling. Det kan være risikoer som har mulighet til å føre til mindre eller større problemer. Hvis man skal klare å gjennomføre en hel utviklingsprosess, er det derfor viktig at man er klar over ulike risikoer, samt at man planlegger for å eventuelt møte på noen av disse.

For å identifisere mulige trusler ved systemutviklingen, kan man bruke en risikomatrise som vist under. I tillegg til de typiske kolonnene «Risiko», «Sannsynlighet», og «Trussel» som beskriver omfanget av de forskjellige risikoene, har vi lagt til en kolonne, «Løsning», som tar for seg mulige løsninger for de forskjellige problemene som kan oppstå av de ulike risikoene.

Risiko	Sannsynlighet	Trussel	Løsning
Forandringer av systemkrav	Høy	Middels	Be kunden om at endringer av systemkrav bør komme tidligst mulig i utviklingsprosessen.
Dårlig tidsestimering	Middels	Middels	«Timeboxing»: allokerer en spesifikk mengde tid for en oppgave.

Feilestimering av oppgavenes omfang	Høy	Middels	Nøye planlegging av gjennomføringen av de ulike oppgavene.
Misforståelse av systemkrav	Middels	Høy	Mye og god kommunikasjon med kunden om hva de ønsker av produktet.
Kode av dårlig kvalitet	Middels	Middels	Gjøre seg kjent med «best practices» og planlegge tidlig hvordan koden skal skrives.
Dårlig samarbeid	Middels	Middels	God kommunikasjon innad.
Alvorlig sykdom	Lav	Høy	God livsstil.

### 2.1.1.4 Personvern

#### Personvernforordningen / GDPR

GDPR (General Data Protection Regulation), også kjent som personvernsforordningen på norsk, er den største og viktigste personvernsforordningen (forordning = bindende lover for hvert enkelt medlemsland) i EU og EØS. GDPR legger mange føringer på blant annet lagring og deling av data fra EU og EØS-landene.

Noen relevante kulepunkter for webutvikling er:

1. Retten til å se lagrede opplysninger om egen person.
2. Retten til sletting ("Retten til å bli glemt").
3. Retten til samtykke av lagring av personlig data.

Ved brudd på disse, eller andre, punktene ved GDPR kan man bli utsatt for ulike sanksjoner avhengig av type brudd og alvorlighetsgraden:

1. Skriftlig advarsel ved førstegangs og ikke-tilsiktede brudd.
2. Krav om jevnlig oppsyn.
3. Bøter på opptil 10/20 millioner EUR eller 2%/4% av total omsetning. Største beløp gjelder.

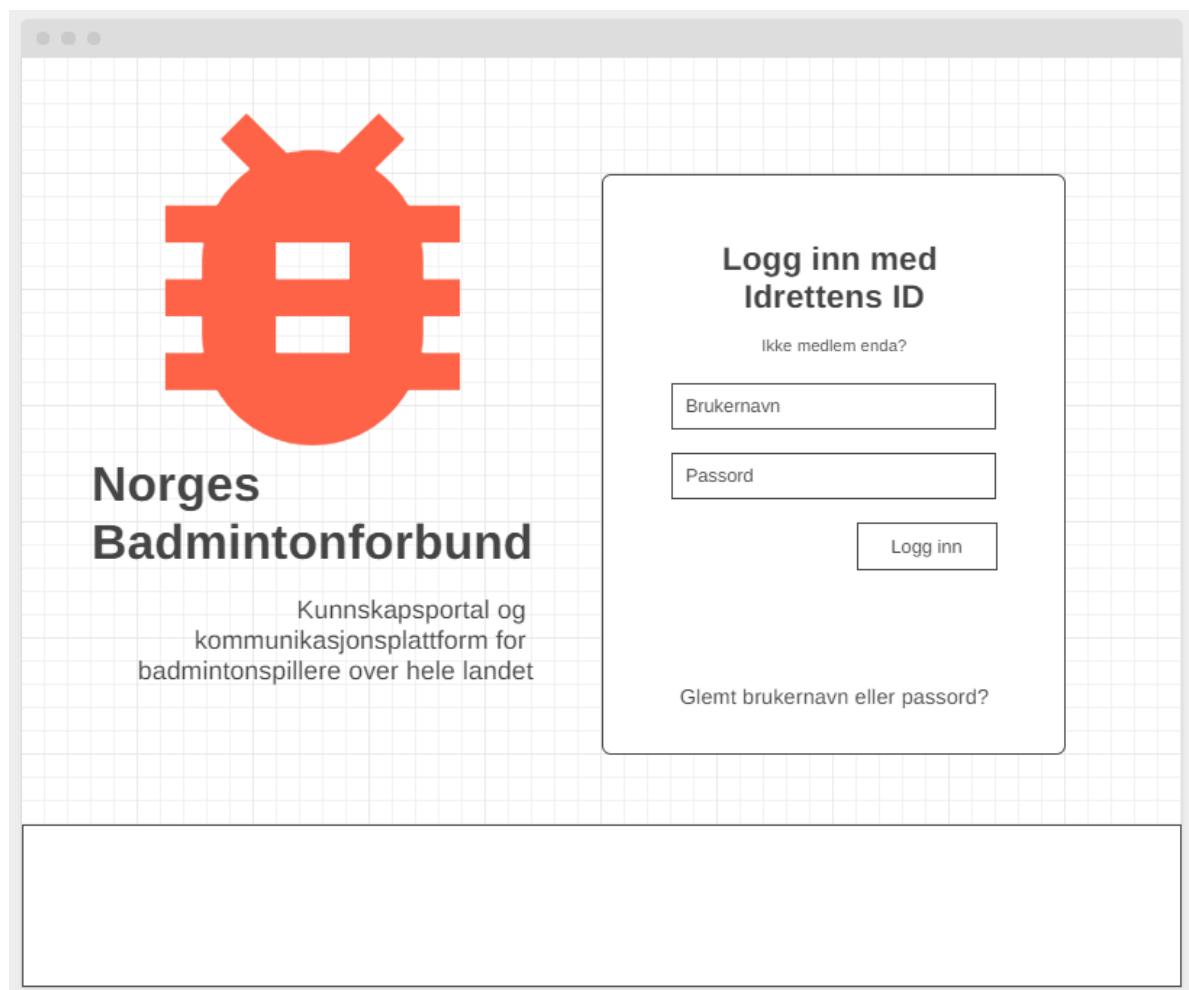
Gjennom utviklingen av webapplikasjonen, har vi tenkt på og tatt hensyn til ulike aspekter ved GDPR og personvern generelt:

1. Brukere kan se all informasjonen webapplikasjonen lagrer om dem direkte i applikasjonen.
2. Brukere tillates å slette egne brukerkontoer direkte i applikasjonen.

### 2.1.1.5 Skissering

For å få et klarere mål på hva oppdragsgiveren faktisk ønsket, lagde vi skisser for hvordan webapplikasjonens utséende kunne se ut. Disse viste vi deretter fram til Charlotte for å få tilbakemeldinger på om vi var på riktig vei. På dette tidspunktet var det klart at de ønsket en innloggingsside, en forsiden, et forum og en kunnskapsdel. Basert på dette produserte vi skisser for dette, der det oransje ikonet er en plassholder for NBF sin logo.

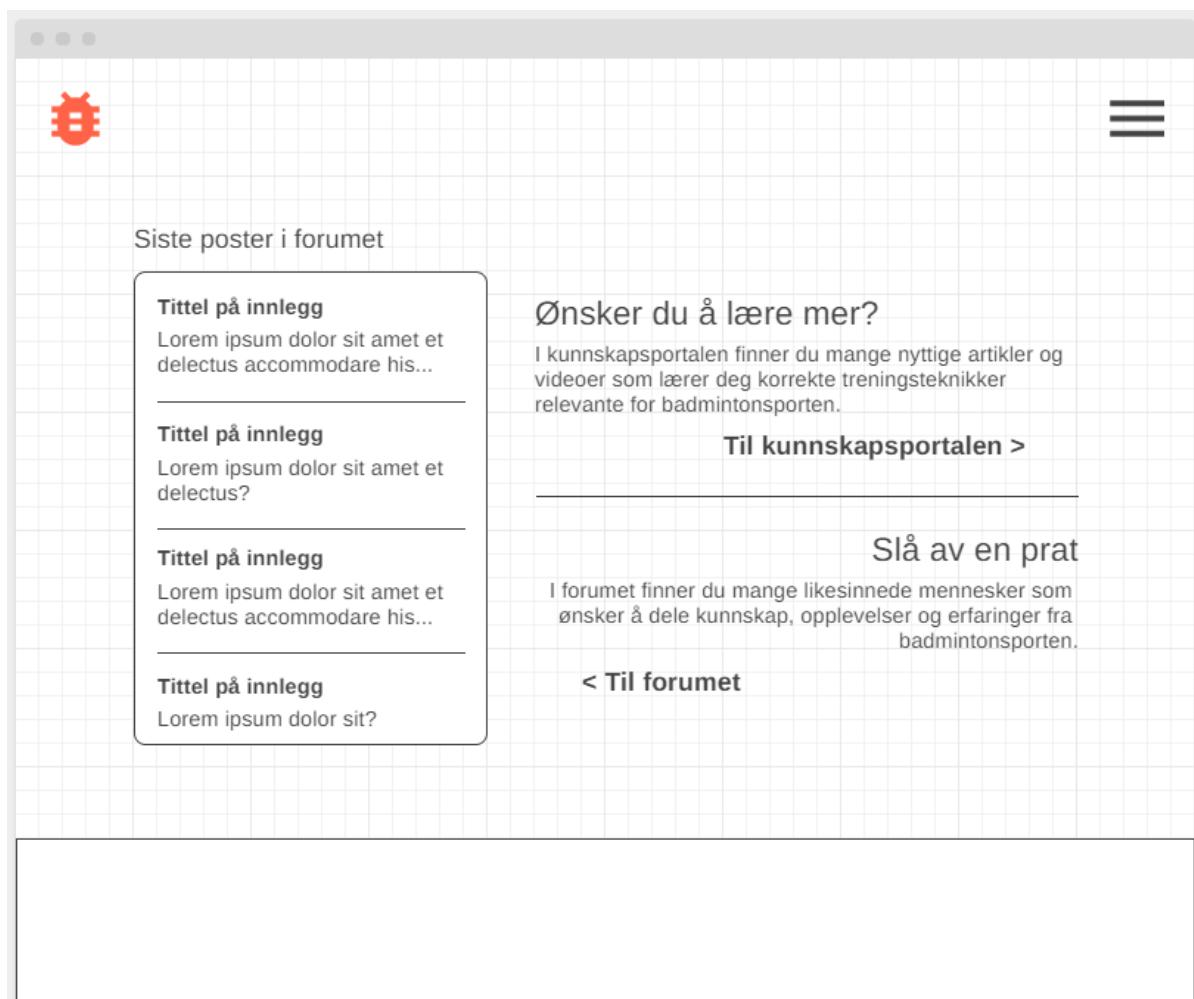
#### Innlogging



Figur 2.1.1.5a: Skisse av innloggingsside

Figur 2.1.1.5a viser en skisse av webapplikasjonens innlogginsside. Her er det tiltenkt at NBF sin logo og en introduserende tekst skal være på venstre side, med innlogging på høyre side. I tillegg er det tiltenkt at det skal være en footer på nedre del av siden.

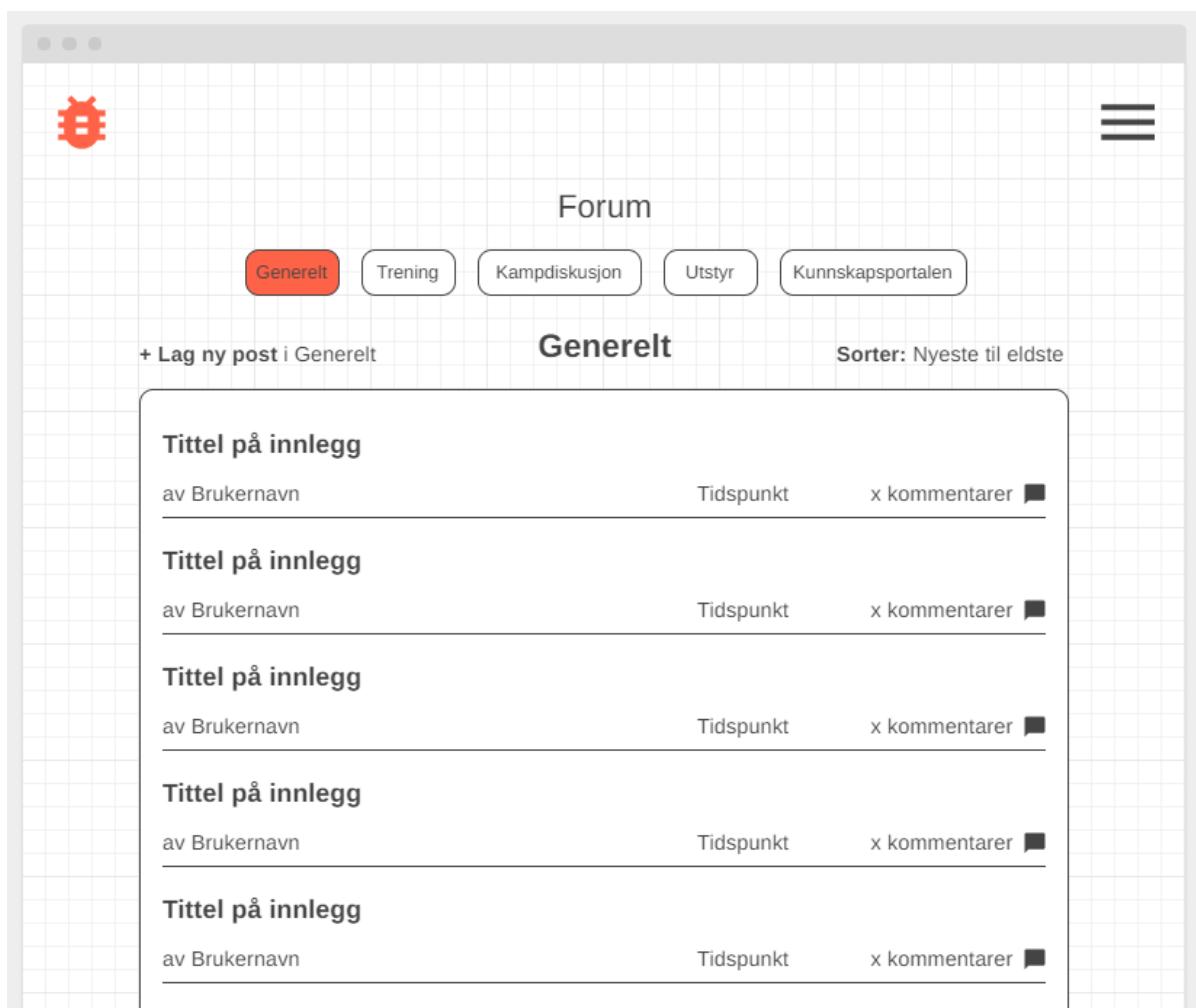
## Forside



Figur 2.1.1.5b: Skisse av webapplikasjonens forside

Figur 2.1.1.5b viser webapplikasjonsens forside, altså det en bruker ser etter å ha logget inn. Her er det tiltenkt at en feed på venstre side skal vise brukeren de siste postene i forumet, dette for å gjøre brukeren oppmerksom på disse og forhåpentligvis øke engasjementet i nye poster. På høyre side finner man linker til kunnskapsportalen og forumet med innledende tekster. Det er også tiltenkt å ha en navigasjonsbar med NBF sin logo, men denne ble ikke skissert.

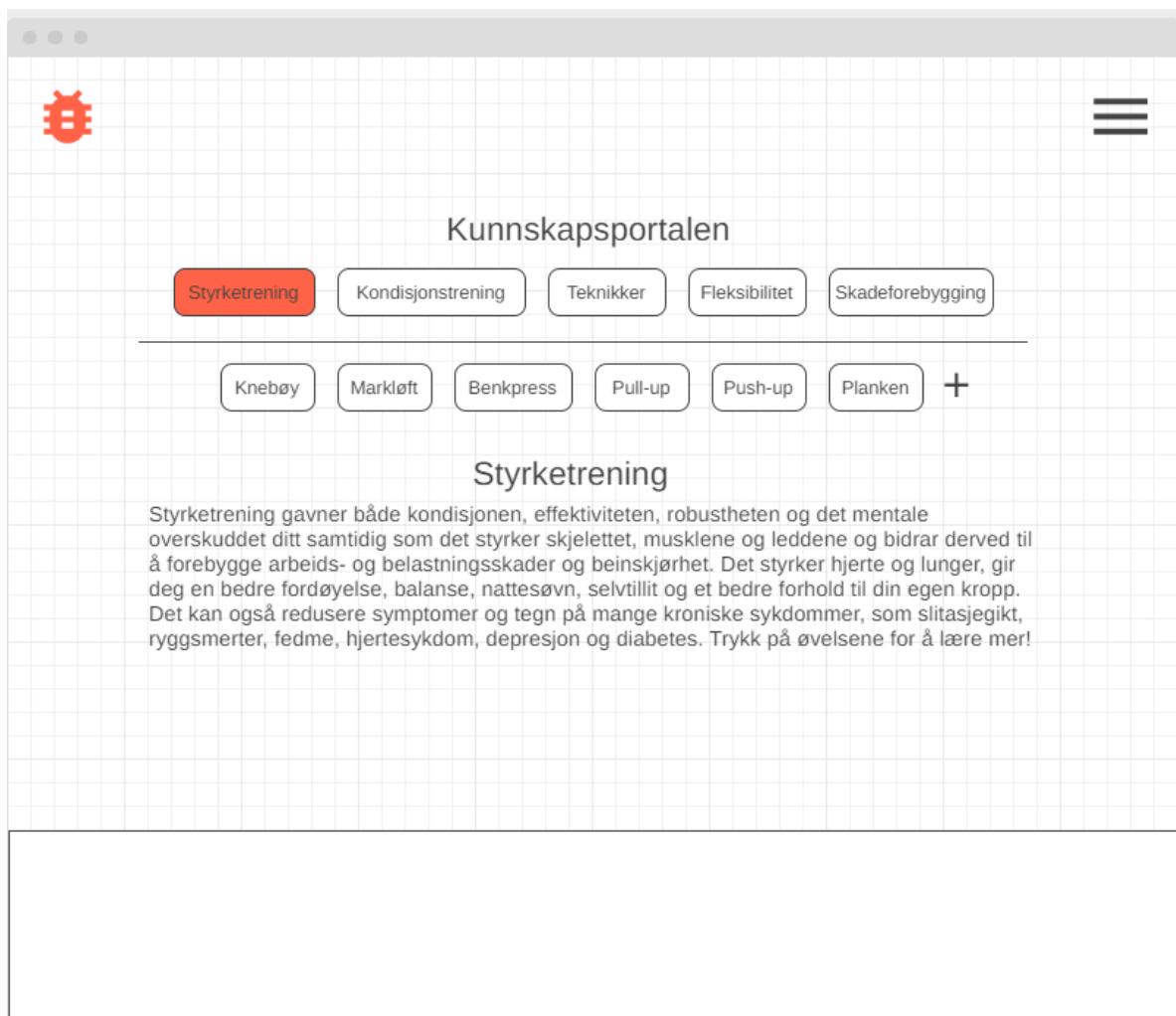
## Forum



Figur 2.1.1.5c: Skisse av forumet

Forumets initielle skisse var svært enkel, se figur 2.1.1.5c. Her skal brukeren velge mellom forskjellige kategorier i forumet, samt lage ny post og sortere postene etter ønske. Feeden, altså samlingen av webapplikasjonens poster, skal vise brukeren titlene på postene, brukernavn på de som opprettet postene, når de ble opprettet og antall kommentarer for hver post. Herfra skal brukeren kunne klikke på en post for å lese innholdet og eventuelt kommentere.

## Kunnskapsportalen



*Figur 2.1.1.3d: Skisse av kunnskapsportalen*

I prosjektets begynnelse var det ikke like klart hva kunnskapsportalens kravspesifikasjon skulle innebære, så dermed skisserte vi et forslag for hvordan denne kunne se ut, se figur 2.1.1.3d. Her er det tiltenkt at brukeren skal kunne sortere etter kategorier og underkategorier og se innholdet i disse. Senere ble det bestemt at kunnskapsportalen hovedsakelig skulle støtte opplasting av videoer og dokumenter, slik at NBF selv kunne enkelt publisere nytt innhold ved behov.

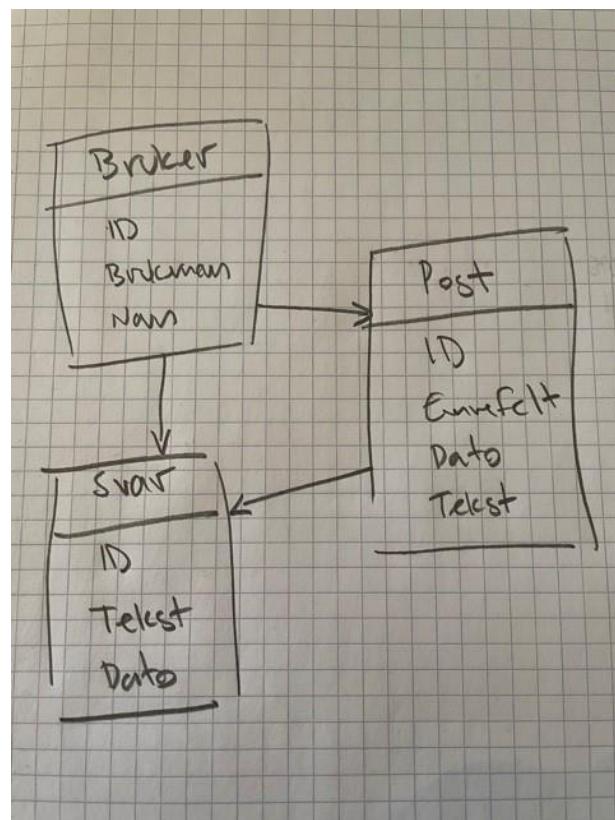
## 2.1.1.6 Modellering

### Databasen

Da vi skulle modellere databasen, begynte vi først med det aller enkleste og deretter la vi til flere attributter og entiteter etter hvert som systemet vokste. Det første utkastet av databasen hadde derfor en meget simpel ER-modell. Vi tok forum-delen av oppgaven som utgangspunkt for denne modellen. Som en del av et diskusjonsforum vil det være nødvendig å dele opp tabellene til databasen på en måte som gjør det logisk å lagre unik informasjon i hver tabell og bruke primærnøkler/fremmednøkler for å lage en relasjon mellom disse. Vi startet med en kladd av databasen på første normalform (heretter 1NF).

### Første normalform

En tabell på 1NF skal kun inneholde atomære verdier (Kristoffersen, 2009, s. 169), det vil si at den ikke inneholder verdier som kan deles opp. Et eksempel på dette vil være en tabell med poster som også inneholder informasjon om brukeren som laget denne posten. Selve skissen over databasen på 1NF ble tegnet som kladd på papir som vist på figur 2.1.1.6a.



Figur 2.1.1.6a: Første kladd på papir

Her bestemte vi oss for at vi trengte tre entiteter til å begynne med for et fungerende forum. Disse tre entitetene ble da en bruker, en post og et svar. Attributtene til disse entitetene ble også lagt til, men justert noe når vi skulle normalisere tabellen til andre normalform (heretter 2NF).

### **Andre normalform**

For at en tabell skal oppfylle kravene for 2NF må den allerede være på 1NF og ikke inneholde noen partielle avhengigheter (Kristoffersen, 2009, s. 169). Dette kan vi oppnå ved å dele opp en tabell slik at alle feltene i tabellen vil være avhengig av en primærnøkkel. Brudd på 2NF forekommer som regel når attributter blir feilplassert i en entitet, og gjerne steder hvor en tabell kunne vært delt opp i to mindre tabeller. Når en skal normalisere en tabell er dette for å dele den opp i enklere tabeller og fjerne muligheten for redundans (Kristoffersen, 2009, s. 173). Redundans er det samme som partiell lagring og kan sees på som dobbeltlagring. Slik vi har gått frem med denne modellen har vi ikke hatt så stort behov for å skille mellom en tabell på 2NF og tredje normalform (heretter 3NF), da vi etter en skisse på papiret allerede hadde delt inn alle tabellene ganske nøyne.

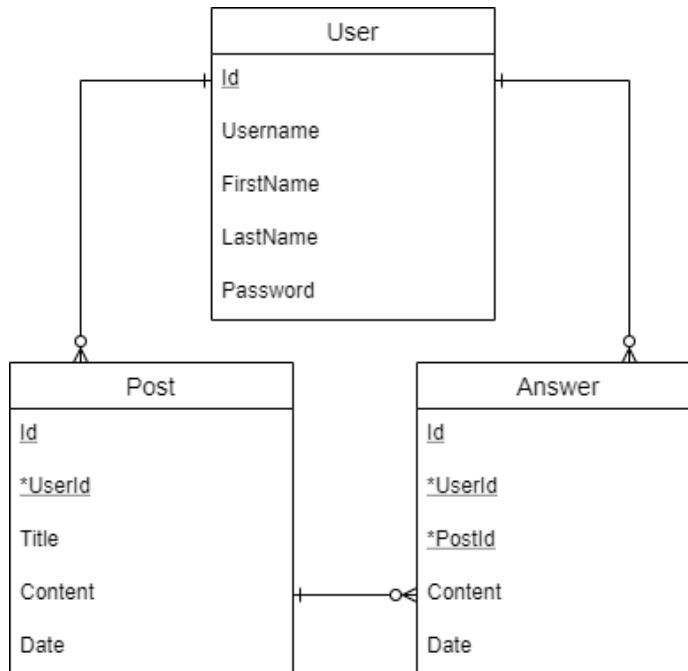
### **Tredje normalform**

Når en skal normalisere en tabell til 3NF må tabellen først være på 2NF, og den kan ikke inneholde noen transitive avhengigheter (Kristoffersen, 2009, s. 170). Et eksempel på dette kan vi se på den siste ER-modellen. En transitiv avhengighet her ville vært om et emne (topic) hadde vært en attributt til entiteten for et underemne (subtopic). Dette hadde vært et brudd på 3NF og tabellen ville derfor fortsatt ha vært på 2NF. Har vi et underemne, så er også emne gitt og skal derfor ligge i sin egen tabell. På den første ER-modellen vi laget hadde vi inkludert egne primærnøkler og fremmednøkler i de forskjellige tabellene som hadde relasjoner til hverandre. Da nærmer vi oss allerede en tabell på Boyce-Codd normalform (heretter BCNF).

### **Boyce-Codd normalform**

En tabell er på BCNF hvis enhver determinant er en supernøkkel (Kristoffersen, 2009, s. 171). Dette betyr at enhver minimal determinant er en kandidatnøkkel. En determinant er når vi har en kolonne i en tabell som er lik en kolonne i en annen tabell slik at det blir en relasjon mellom disse to. Det er ingen forskjell på 3NF og BCNF i tabeller som kun har en eneste kandidatnøkkel siden dette da blir primærnøkkelen, forskjellen oppstår først når en har tabeller med overlappende kandidatnøkler (Kristoffersen, 2009, s. 174). Et eksempel på dette

er når vi ser på tabellen til brukere (user). Her har alle rader en unik primærnøkkel (Id). Vi ser også på tabellene til poster (post) og svar (answer) at alle radene har en unik primærnøkkel (Id), men også en unik fremmednøkkel (UserId) som peker direkte på bruker sin primærnøkkel. Vi har derfor en funksjonell avhengighet mellom alle tabellene, men ingen overlappende kandidatnøkler og har fjernet all redundans, dermed oppfyller tabellen på figur 2.1.1.6b kravet til BCNF.



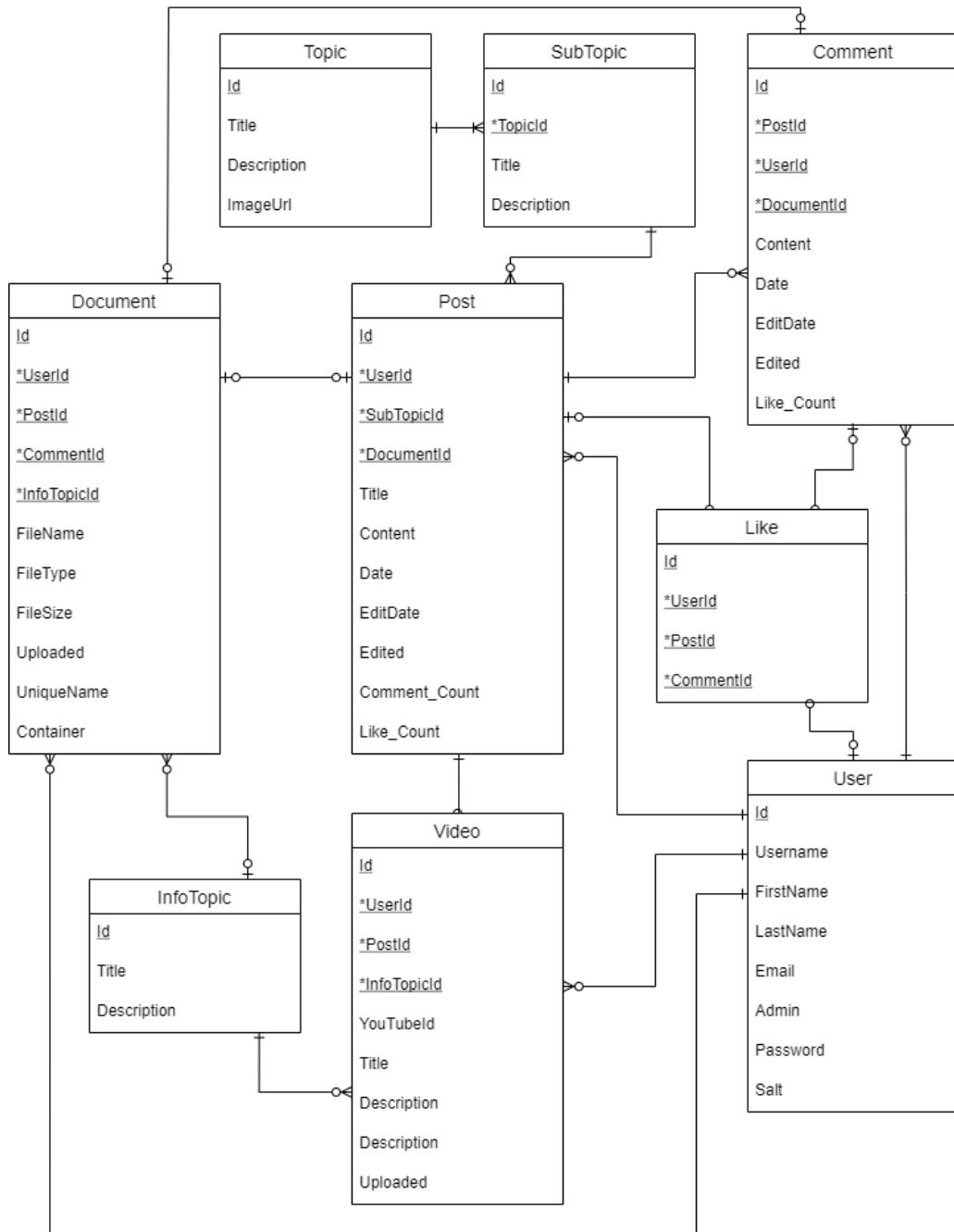
Figur 2.1.1.6b: Utgangspunktet for databasen

## Sluttresultatet

Siden vi har tilnærmet oss denne oppgaven på en slik måte at vi kun starter med det som er nødvendig og bygge videre på dette, har vi ikke hatt store modeller å forholde oss til i starten og normalisering av tabellene ble kun gjort på den første modellen. Når vi bygget videre på databasen, la vi til entiteter og attributter etter hvert som vi fikk brukt for det i systemet.

Utviklingen av databasen har derfor gått parallelt med utviklingen av systemet. Det første utkastet av databasen ble utgangspunktet for hele systemet. På figuren til ER-modellen for det første utkastet har vi kun med de nødvendige entitetene (user, post og answer). Her kan vi se primærnøkler som det øverste attributtet med strek under navnet. Fremmednøkler er listet under primærnøkkel, med stjerne foran navnet og understrek under hele navnet. Resten av attributtene er listet til slutt. Det går også streker mellom de forskjellige entitetene hvor et symbol representerer relasjonen mellom disse. Et eksempel på disse relasjonene er mellom user og post. Her ser vi at user peker på post med symbolet for ingen eller mange, og post

peker tilbake på user med symbolet for en. Dette betyr at mellom user og post er det en en-til-mange relasjon. User kan derfor ha null, en eller flere poster, men en post kan kun ha en user. Figur 2.1.1.6c er den endelige ER-modellen og denne representerer hele det ferdige databasesystemet.



Figur 2.1.1.6c: Den ferdige databasen

Her kan vi se at vi har fulgt denne fremgangsmåten hele veien, og alle entiteter har korrekte relasjoner og primærnøkler/fremmednøkler i forhold til slik det blir brukt i applikasjonen.

### 2.1.1.7 UML

UML (Unified Modeling Language) er det standardspråket for modellering av programvaresystemer. UML er hovedsakelig basert på diagrammer som forklarer aspekter ved handlingsmønstre og strukturen til det gjeldende systemet. Det vil si at UML-diagrammer kan deles inn i to grupper basert på dette. Som i mange andre programvaresystemer hadde vi før utviklingsstart modellert hvordan vi hadde forestilt oss systemets funksjonalitet og oppsett, basert på kravspesifikasjonene. Dette gjøres for å planlegge hvordan man skal jobbe med selve utviklingen. Modellering av programvaresystemer kan sammenlignes med arkitekttegning; i begge er det nødvendig å lage modeller og planer for å ha en ledestjerne mot det ferdige produktet.

"We build models so that we can better understand the system we are developing." (Booch et al., 2005, s. 17).

*"Gjennom modellering oppnår vi fire mål:*

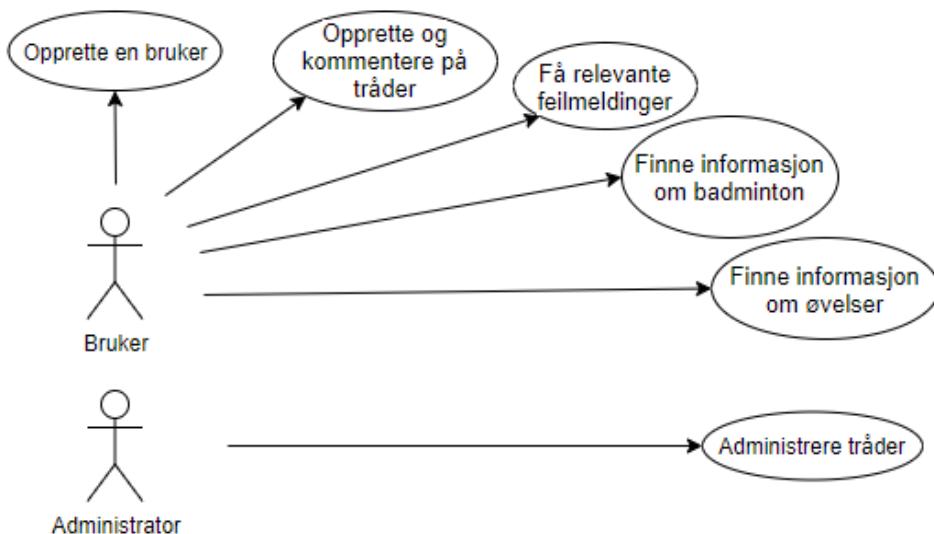
- *Modeller hjelper oss å visualisere et system slik det er eller slik vi vil ha det.*
- *Modeller tillater oss å spesifisere strukturen eller oppførselen til et system.*
- *Modeller gir oss en mal som veileder oss i å konstruere et system.*
- *Modeller dokumenterer beslutningene vi har tatt."*

(Booch et al., 2005, s. 18)

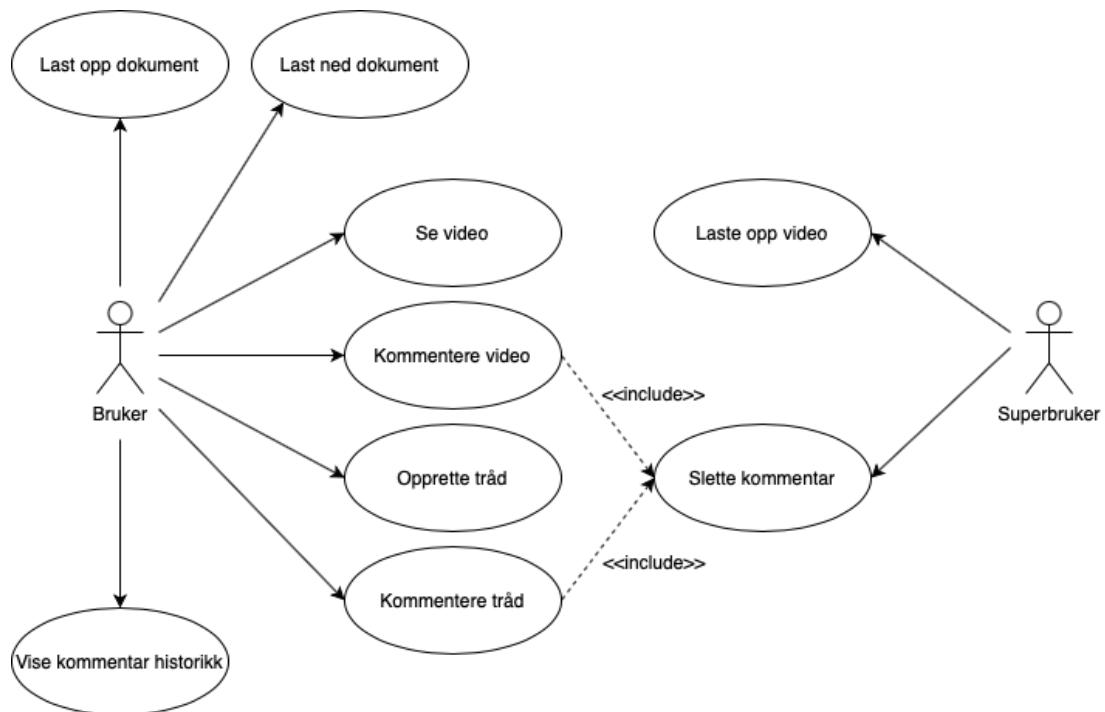
Vi begynte tidlig med å sette opp use case-diagrammer basert på de første kravspesifikasjonene gitt av oppdragsgiver. Deretter gikk vi videre til å produsere ER-, klasse- og sekvensdiagrammer.

#### Use case-diagram

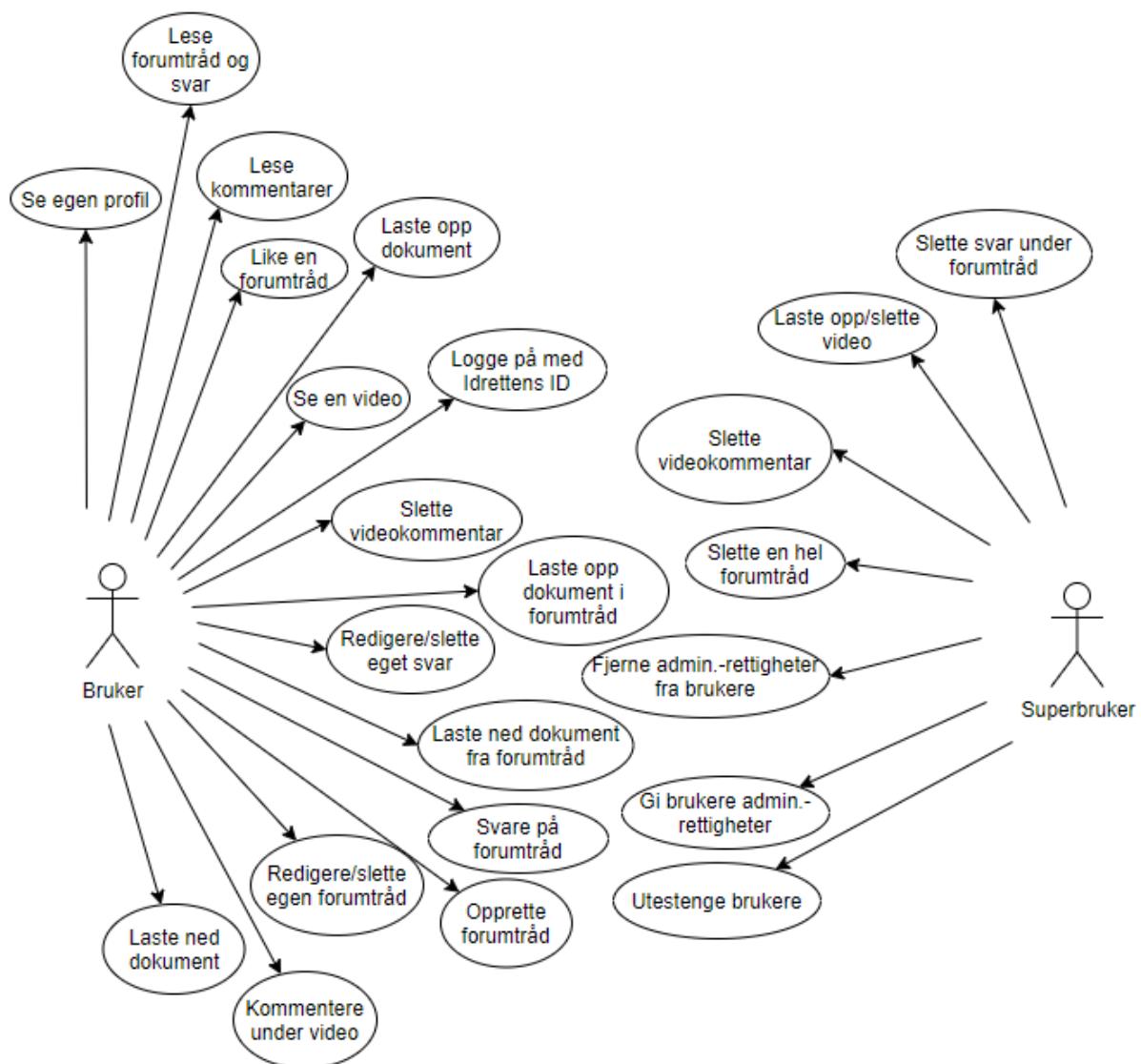
Et use case-diagram er en grafisk beskrivelse eller oversikt av en forskjellige interaksjoner en bruker kan ha med et system. Et use case-diagram viser ulike bruksområder og forskjellige typer brukere systemet har og vil ofte også bli ledet av andre typer diagrammer. Use casene er representert som sirkler eller ellipser med tilhørende tekst. Skuespillerne er representert som strekfigurer. "Use case diagrammer viser systemets funksjonalitet og samspillet mellom systemet og omgivelsene (brukere, andre systemer, komponenter)" - (Lindsjørn, 2020, s. 18). Figur 2.1.1.7a-d viser utviklingen av webapplikasjonens use case-diagram.



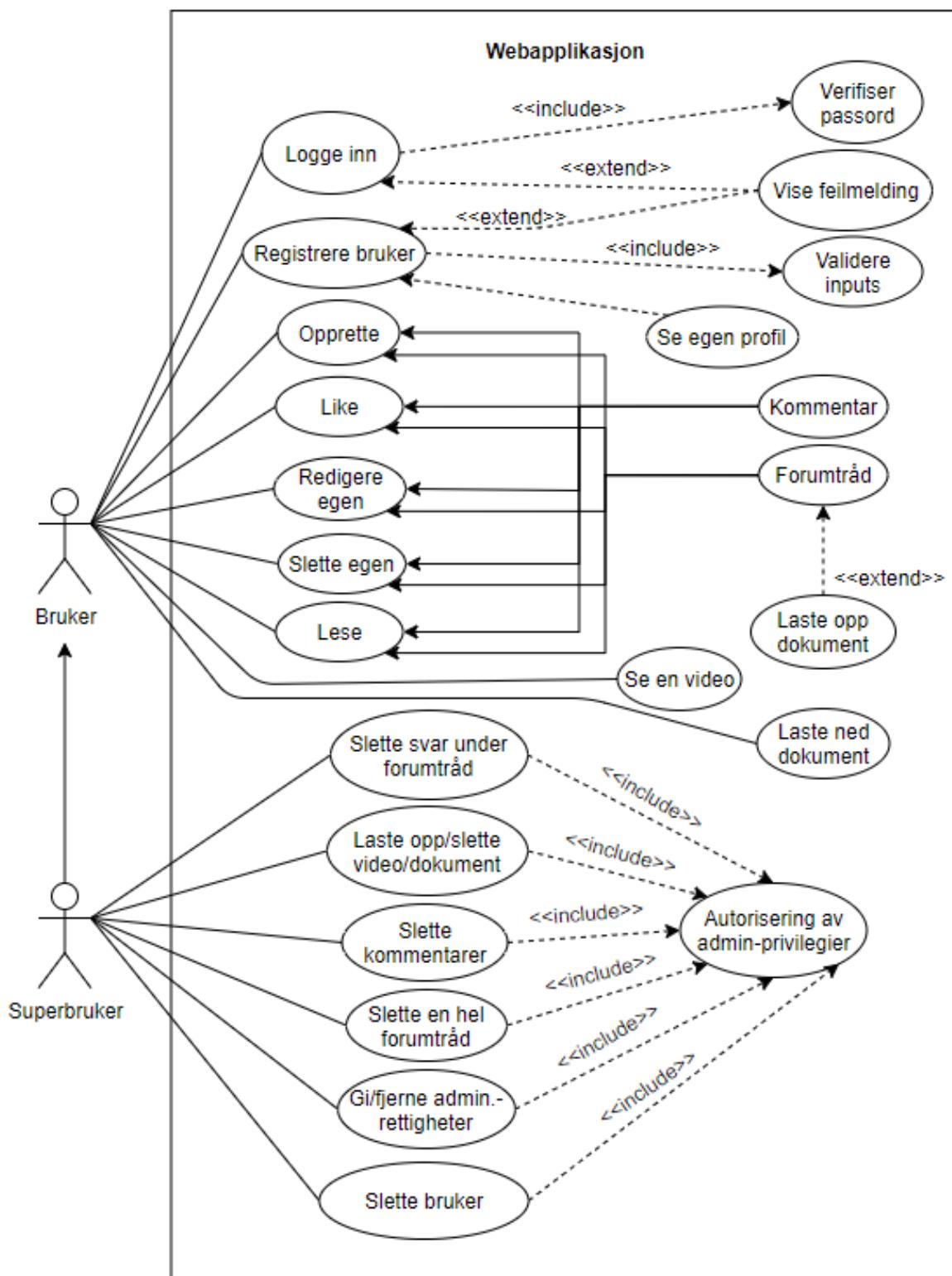
Figur 2.1.1.7a: Førsteutkast av use case-diagram



Figur 2.1.1.7b: Andreutkast av use case-diagram



Figur 2.1.1.7c: Tredjeutkast av use case diagram



Figur 2.1.1.7d: Det ferdige use case-diagrammet

## Tekstlig beskrivelse

En tekstlig beskrivelse, i UML-sammenheng, er en utvidelse av et use case-diagram. I stedet for å modellere brukeres samhandling med et system som et diagram, går tekstlige beskrivelser ut på å forklare brukernes systeminteraksjoner tekstlig.

### Use case: «Logge inn»

**Aktør:** Bruker, Superbruker

**Prebetingelser:** Bruker må ha registrert brukerprofil.

**Postbetingelser:** Bruker skal være logget inn.

#### Hovedflyt:

1. Bruker navigerer til internettdomenet som hoster webapplikasjonen.
2. Bruker skriver inn brukernavn eller epostadresse, og passord.
3. Bruker trykker «enter» på tastaturet, eller klikker på «Logg inn»-knappen.
4. Webapplikasjonen validerer feltene.
5. Webapplikasjonen sender HTTP-forespørsel til backend og ber om brukerdetaljer.
6. HTTP-forespørselen ble godkjent og brukeren logges inn og blir sendt til forsiden.

#### Alternativ flyt 1, steg 4:

A1.1 Feltene blir ikke godkjent under validering.

A1.2 Bruker må skrive inn brukernavn eller epostadresse, og passord igjen.

#### Alternativ flyt 2, steg 5:

A2.1 Feltene ble validert, men innloggingsdetaljene er ikke riktige.

A2.2 Bruker må skrive inn brukernavn eller epostadresse, og passord igjen.

**Use case: «Registrere bruker»**

**Aktør:** Bruker (Bruker kan kun registreres uten administratorprivilegier)

**Prebetingelser:** Brukernavn kan ikke være registrert fra tidligere.

**Postbetingelser:** Brukernavn skal være registrert og bruker skal være logget inn.

**Hovedflyt:**

1. Bruker nавигerer til internettdomenet som hoster webapplikasjonen.
2. Bruker trykker på «Registrer»-fanen i innloggings- /registreringssiden.
3. Bruker skriver inn ønsket brukernavn, fornavn, etternavn, epost-adresse, samt passord i to ulike felter.
4. Bruker trykker «enter» på tastaturet, eller klikker på «Registrer»-knappen.
5. Webapplikasjonen validerer feltene.
6. Webapplikasjonen sender HTTP-forespørsel til backend og ber om brukerdetaljer.
7. HTTP-forespørselen ble godkjent og brukeren logges inn og blir sendt til forsiden.

**Alternativ flyt 1, steg 5:**

A1.1 Feltene blir ikke godkjent under validering.

A1.2 Bruker må skrive inn brukernavn eller epostadresse, og passord igjen.

**Alternativ flyt 2, steg 6:**

A2.1 Feltene ble validert, men innloggingsdetaljene er ikke riktige.

A2.2 Bruker må skrive inn brukernavn eller epostadresse, og passord igjen.

**Use case: «Opprette»****Aktør:** Bruker, Superbruker**Prebetingelser:** Bruker må være logget inn. Server må ha nok plass til objektet.**Postbetingelser:** Objektet skal være opprettet.**Hovedflyt:**

1. Bruker trykker på Forum-linken på toolbaren.
2. A) Hvis bruker skal opprette post, trykker bruker på relevant kategori og underkategori.
3. A) Bruker trykker på «Ny post»-knappen.
4. A) Bruker skriver inn tittel og innhold til post, legger eventuelt ved fil.
5. A) Bruker trykker «Send inn»-knappen.
6. B) Hvis bruker skal opprette kommentar eller legge ved fil, trykker bruker på relevant post.
7. B) Bruker skriver kommentar i kommentar-taben eller legger ved fil i vedlegg-taben.
8. B) Bruker trykker «Send inn»-knappen.

**Use case: «Like»****Aktør:** Bruker, Superbruker**Prebetingelser:** Bruker må være logget inn. Objekt må eksistere.**Postbetingelser:** Objekt skal være "liked".**Hovedflyt:**

1. Bruker trykker på Forum-linken på toolbaren.
2. Bruker trykker på relevant post.
3. Bruker trykker «» på den relevante posten eller kommentaren.

**Use case: «Redigere egen»****Aktør:** Bruker, Superbruker**Prebetingelser:** Bruker må være logget inn. Objekt må eksistere.**Hovedflyt:**

1. Superbruker trykker på Kunnskapsportalen-linken på toolbaren.
2. Superbruker trykker på «Videoer»- eller «Dokumenter»-knappen, avhengig av hva superbruker ønsker å behandle.
3. Superbruker trykker «Slett»-knappen under de relevante objektene, eller «Last opp fil/dokument»-knappen
4. A) Hvis superbruker trykket «Last opp fil»-knappen, må vedkommende trykke på «Velg fil»-feltet og bla gjennom datamaskinen sin for å finne relevant fil.
5. A) Deretter må superbruker velge kategori som passer med filen.
6. A) Superbruker trykker «Send»-knappen.
7. B) Hvis superbruker trykker «Last opp video»-knappen, må vedkommende skrive inn gyldig YouTube-URL, en tittel, og en beskrivelse til videoen.
8. B) Deretter må superbruker velge kategori som passer med videoen.
9. B) Superbruker trykker «Send»-knappen

**Use case: «Slette egen»****Aktør:** Bruker, Superbruker**Prebetingelser:** Bruker må være logget inn. Objekt må eksistere.**Hovedflyt:**

1. Bruker trykker på Forum-linken på toolbaren.
2. Superbruker trykker på den relevante posten.
3. Bruker trykker «Slett»-knappen ved posten eller relevant kommentar.

### **Use case «Lese»**

**Aktør:** Bruker, Superbruker

**Prebetingelser:** Bruker må være logget inn. Objekt må eksistere.

#### **Hovedflyt:**

1. Bruker trykker på Forum-linken på toolbaren.
2. Superbruker trykker på den relevante posten.
3. Bruker leser relevant post eller kommentar.

### **Use case: «Slette svar under forumtråd»**

**Aktør:** Superbruker

**Prebetingelser:** Superbruker må være logget inn. Serveren må ha nok plass til objektet.

**Postbetingelser:** Objektet skal være lastet opp.

#### **Hovedflyt:**

1. Superbruker trykker på Kunnskapsportalen-linken på toolbaren.
2. Superbruker trykker på «Videoer»- eller «Dokumenter»-knappen, avhengig av hva superbruker ønsker å behandle.
3. Superbruker trykker «Slett»-knappen under de relevante objektene, eller «Last opp fil/dokument»-knappen
4. A) Hvis superbruker trykket «Last opp fil»-knappen, må vedkommende trykke på «Velg fil»-feltet og bla gjennom datamaskinen sin for finne relevant fil.
5. A) Deretter må superbruker velge kategori som passer med filen.
6. A) Superbruker trykker «Send»-knappen.
7. B) Hvis superbruker trykker «Last opp video»-knappen, må vedkommende skrive inn gyldig YouTube-URL, en tittel, og en beskrivelse til videoen.

8. B) Deretter må superbruker velge kategori som passer med videoen.
9. B) Superbruker trykker «Send»-knappen

**Use case: «Se en video»****Aktør:** Bruker, superbruker**Prebetingelser:** Bruker må være logget inn.**Hovedflyt:**

1. Bruker trykker på Kunnskapsportalen-linken på toolbaren.
2. Bruker trykker på «Videoer»-knappen.
3. Bruker trykker på relevant video og «Spill av»-knappen.

**Use case: «Laste ned dokument»****Aktør:** Bruker, superbruker**Prebetingelser:** Bruker må være logget inn.**Hovedflyt:**

1. Bruker trykker på Kunnskapsportalen-linken på toolbaren.
2. Bruker trykker på «Dokumenter»-knappen.
3. Bruker trykker på relevant dokument.

**Use case: «Laste opp/slette video/dokument»****Aktør:** Superbruker**Prebetingelser:** Superbruker må være logget inn. Serveren må ha nok plass til objektet.**Postbetingelser:** Objektet skal være lastet opp.**Hovedflyt:**

10. Superbruker trykker på Kunnskapsportalen-linken på toolbaren.

11. Superbruker trykker på «Videoer»- eller «Dokumenter»-knappen, avhengig av hva superbruker ønsker å behandle.
12. Superbruker trykker «Slett»-knappen under de relevante objektene, eller «Last opp fil/dokument»-knappen
13. A) Hvis superbruker trykket «Last opp fil»-knappen, må vedkommende trykke på «Velg fil»-feltet og bla gjennom datamaskinen sin for finne relevant fil.
14. A) Deretter må superbruker velge kategori som passer med filen.
15. A) Superbruker trykker «Send»-knappen.
16. B) Hvis superbruker trykker «Last opp video»-knappen, må vedkommende skrive inn gyldig YouTube-URL, en tittel, og en beskrivelse til videoen.
17. B) Deretter må superbruker velge kategori som passer med videoen.
18. B) Superbruker trykker «Send»-knappen

#### **Use case: «Slette en hel forumtråd»**

**Aktør:** Superbruker

**Prebetingelser:** Superbruker må være logget inn. Én eller flere forumtråder må eksistere.

**Postbetingelser:** Forumtråden(e) skal være slettet.

#### **Hovedflyt:**

1. Superbruker trykker på Forum-linken på toolbaren.
2. Superbruker blar seg inn på en forumtråd som vedkommende ønsker å slette.
3. Superbruker trykker den øverste «Slett»-knappen.

#### **Use case: «Gi/fjerne admin-rettigheter»**

**Aktør:** Superbruker

**Prebetingelser:** Superbruker må være logget inn.

**Postbetingelser:** Admin-rettigheter skal være forandret.

**Hovedflyt:**

1. Superbruker trykker på toolbaren og administrerer brukere.
2. Superbruker velger true eller false i «Admin»-kolonnen for de relevante brukerne.

**Use case: «Slette bruker(e)»****Aktør:** Superbruker**Prebetingelser:** Superbruker må være logget inn. Bruker(e) må eksistere.**Postbetingelser:** Bruker(e) skal være slettet.**Hovedflyt:**

1. Superbruker trykker på toolbaren og administrerer brukere.
2. Superbruker huker av ved siden av bruker(e) som skal slettes.
3. Superbruker trykker «Slett bruker(e)»-knappen

**User Stories (Brukerhistorier)**

Brukerhistorier beskriver hva ulike aktører ønsker å gjøre i eller med et programvaresystem på en naturlig, standardisert og enklest mulig måte. Brukerhistorier brukes som en utvidelse av use cases, som dermed komplementerer hverandre. Brukerhistorier er skrevet på en fast mal, typisk denne:

Som «aktør» ønsker jeg å «utføre handling» slik at «en begrunnelse».

Våre brukerhistorier for webapplikasjonen er som følger:

1. Som bruker ønsker jeg å logge meg inn slik at jeg kan få tilgang til systemet.
2. Som bruker ønsker jeg å registrere en bruker slik at jeg kan bruke den mot systemet.
3. Som bruker ønsker jeg å opprette kommentarer og forumtråder slik at jeg kan interagere med andre brukere via systemet.
4. Som bruker ønsker jeg å «like» kommentarer og forumtråder slik at jeg kan vise støtte til innhold jeg liker eller synes er nyttig.

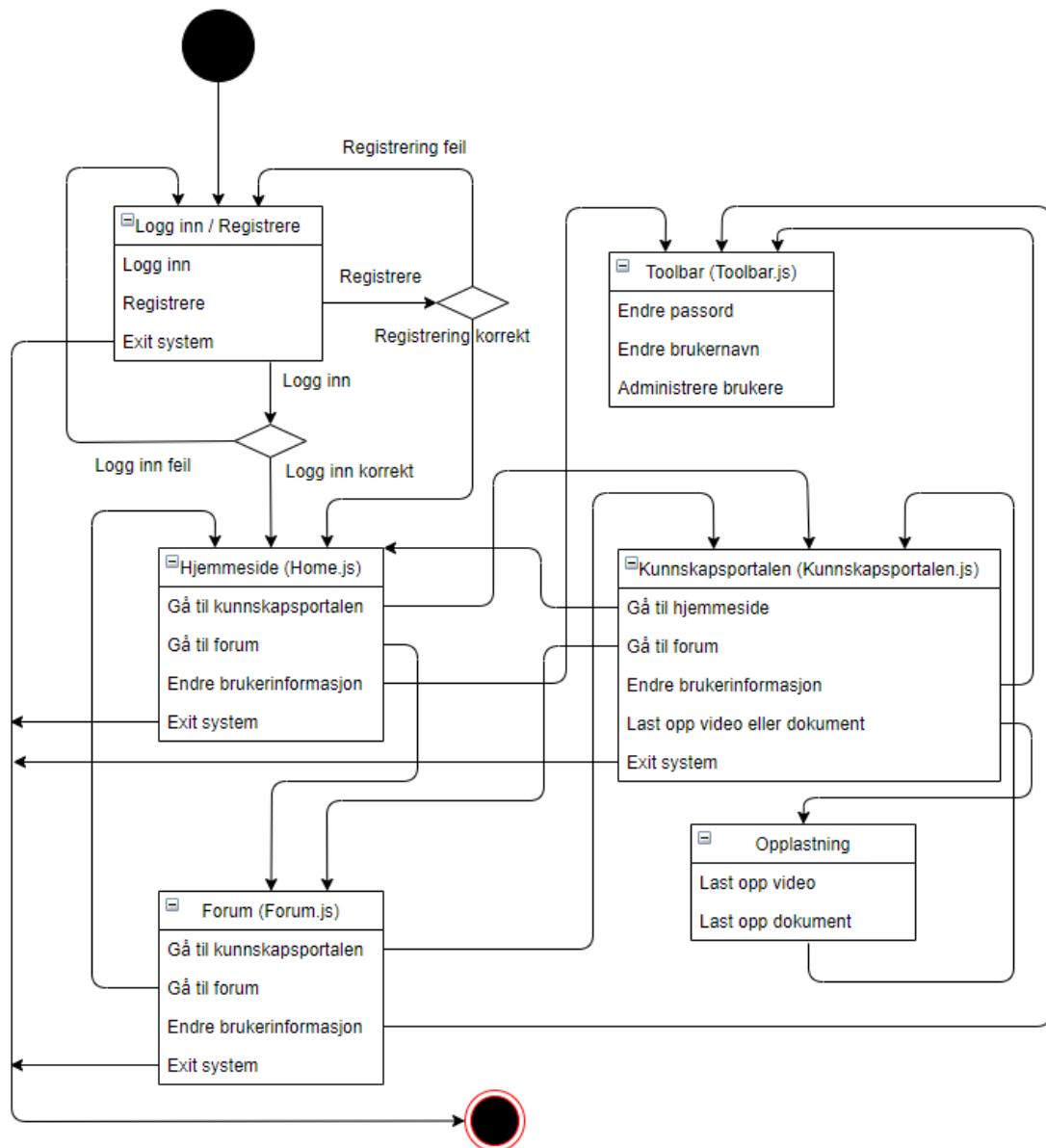
5. Som bruker ønsker jeg å ha muligheten til å redigere egne kommentarer og forumtråder slik at jeg kan skrive mer utfyllende om, eller gi oppklaring i misforståelser i teksten.
6. Som bruker ønsker jeg å ha muligheten til å slette egne kommentarer og forumtråder slik at jeg kan fjerne unødvendig informasjon fra forumet.
7. Som bruker ønsker jeg å lese kommentarer og forumtråder i forumet slik at jeg kan oppdatere meg om hva som skjer i organisasjonen.
8. Som superbruker ønsker jeg å slette kommentarer under forumtråder slik at uønsket innhold, for eksempel banneord og hets, forsvinner fra forumet.
9. Som superbruker ønsker jeg å laste opp og ha muligheten til å slette videoer og andre dokumenter i kunnskapsportalen slik at brukerne kan bli oppdatert på hva som skjer i organisasjonen eller bli informert om andre relevante saker.
10. Som superbruker ønsker jeg å ha muligheten til å slette en hel forumtråd slik at unødvendig eller irrelevant informasjon fjernes fra forumet.
11. Som superbruker ønsker jeg å kunne gi og fjerne administratorrettigheter til ulike brukere slik at flere i ledelsen kan delta i moderering og innholdsproduksjon, og slik at brukere som ikke skal ha den tilgangen, mister den.
12. Som superbruker ønsker jeg å kunne slette brukerkontoer slik at brukere som misbruker systemet kan uteslenges.

## Tilstandsdiagram

Et tilstandsdiagram er et annet av de mye brukte UML-diagrammene (figur 2.1.1.7.e).

Tilstandsdiagram brukes til å forklare hvordan et systems tilstand forandres ved bruk.

Tilstandsdiagram begynner med en svart sirkel (startnode), og avslutter med en svart sirkel med en rød ring rundt (sluttnode). Innimellom er systemets mange tilstander beskrevet ved navn og funksjonalitet. Tilstandene er koblet sammen med piler og valg-diamanter som beskriver hvordan systemet går fra én tilstand til en annen. (Lindsjørn, 2020, s. 34).



Figur 2.1.1.7e: Tilstandsdiagram

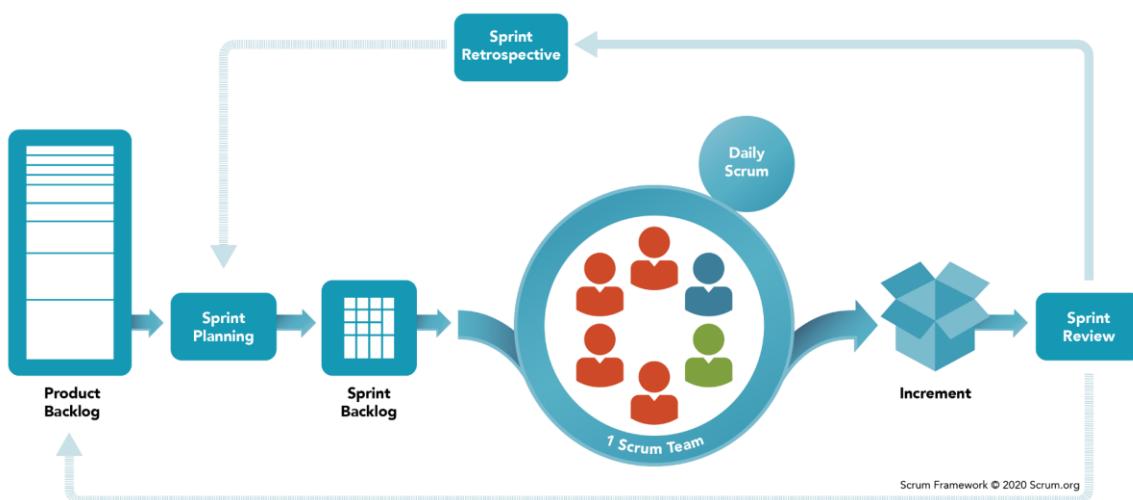
## 2.1.2 Metode

Prosjektet hadde ingen klart definert arbeidsmetodikk, men vi visste at vi ville ta i bruk grunnelementer fra smidige metoder som Scrum og Kanban. Begge disse metodene egner seg godt for utvikling av webapplikasjoner da de tillater prosjektet å ta et steg tilbake hvis det er behov for det.

### 2.1.2.1 Smidig metode

En vanlig tilnærming i smidige prosesser er inkrementelt arbeid, altså at man bygger opp et system litt etter litt. Scrum og Kanban eksempler på smidige utviklingsmetoder, og vi brukte elementer fra begge disse tilnærmingene i prosjektet.

#### Scrum



Figur 2.1.2.1a: What is Scrum?, 2021, (<https://www.scrum.org/resources/what-is-scrum>)

Scrum (figur 2.1.2.1a) er et rammeverk brukt til å administrere produktutvikling. Det lar deltakerne i teamet etablere hypoteser for hvordan noe kan fungere, prøve det ut i praksis, reflektere over erfaringen og deretter gjøre forandringer på produktet ved behov. I Scrum opererer man med en "product backlog", altså alt som skal være med i produktet i løpet av en "sprint". En sprint er en fase på 1-4 uker der man har satt et mål om å utvikle en del av systemet tilnærmet ferdig og deretter levere den til kunden. Denne potensielt ferdige delen av systemet omtales som et inkrement (Agile Alliance, u.å.). I vårt prosjekt har vi operert

med lignende faser som ender i en milepæl der vi presenterer inkrementet for oppdragsgiveren. Ved å gjøre dette sikret vi oss en tilbakemelding på produktet slik at vi kunne tilpasse det etter NBF sine nye ønsker og krav. I tillegg hadde gruppa møte mandag, onsdag og fredag hver uke, sammenlignbart med stand-up-praksisen i Scrum. Ideelt sett hadde dette foregått daglig, men på grunn av Covid-19-situasjonen og forskjellige livssituasjoner med tanke på andre studier og jobb, ble ikke dette aktuelt for oss.

## Kanban



Figur 2.1.2.1b: What is Kanban?, u.å., (<https://www.digitel.com/kanban/what-is-kanban/>)

Kanban (figur 2.1.2.1b) er også et rammeverk for produktutvikling og sammenlignes ofte med Scrum. I Kanban er det vanlig å visualisere "work items" i et "Kanban Board". Dette boardet gir et team muligheten til å se hva som må gjøres, hva som arbeides med og hva som er ferdig. (Max Rehkopf, u.å.). Azure DevOps har en funksjon som lar brukere opprette work items, se figur 2.1.2.1c. Her kan man også velge hvordan man skal visualisere disse, og vi har valgt boards til dette, lignende et Kanban Board. Dersom en på gruppa så en feil i webapplikasjonen, kunne hun eller han opprette work items i boardet som beskrev denne feilen, slik at de andre på gruppa kunne se dette og eventuelt rette opp i feilen. På denne måten kunne gruppa sikre seg at ingen jobbet med det samme problemet samtidig.

Frontend Team ▾ ⚡ 8

Board Analytics View as Backlog

To Do	Doing	5/5	Done
<a href="#">New item</a>			
<a href="#">91 Sorteringsfunksjon for kommentarer i tråder</a> State To Do	<a href="#">64 InfoTopics i Kunnskapsportalen</a> PA Pia Karoline Aamodt State Doing		<a href="#">70 Fikse rekkefølge på kommentarer i poster</a> PA Pia Karoline Aamodt State Done
<a href="#">90 Sidefunksjon for kommentarer i tråder</a> State To Do	<a href="#">66 Stilisere Homejs</a> PA Pia Karoline Aamodt State Doing		<a href="#">68 fix likes</a> EL Erik Skrautvol Larsen State Done
<a href="#">85 Topic-bilder fra DB til mappe i frontend</a> State To Do	<a href="#">78 Appen kræsjer på Azure Portal etter en stund. Kaster en 502 feilmelding.</a> EL Erik Skrautvol Larsen State Doing		<a href="#">69 Søkefunksjon i Forum.js (SearchPosts.js)</a> EL Erik Skrautvol Larsen State Done
<a href="#">83 Fjerne DateTime ved nye poster og kommentarer (gjelder også ved redigering). Dette skjer i backend nå (W. Europe Standard Time)</a> State To Do	<a href="#">39 Lage valideringsmetoder til alle inputfelt</a> ST Sepideh Tajik		<a href="#">84 Nyeste burde være øverst som default på sortering</a> HH Henrik Nøkleby Hjellup
<a href="#">80 Knappen for subplot må forblí valgt etter å ha blitt trykket på (ikkt som bildet for topic forblí)</a>			

Figur 2.1.2.1c: Skjermbilde av work items i Azure DevOps

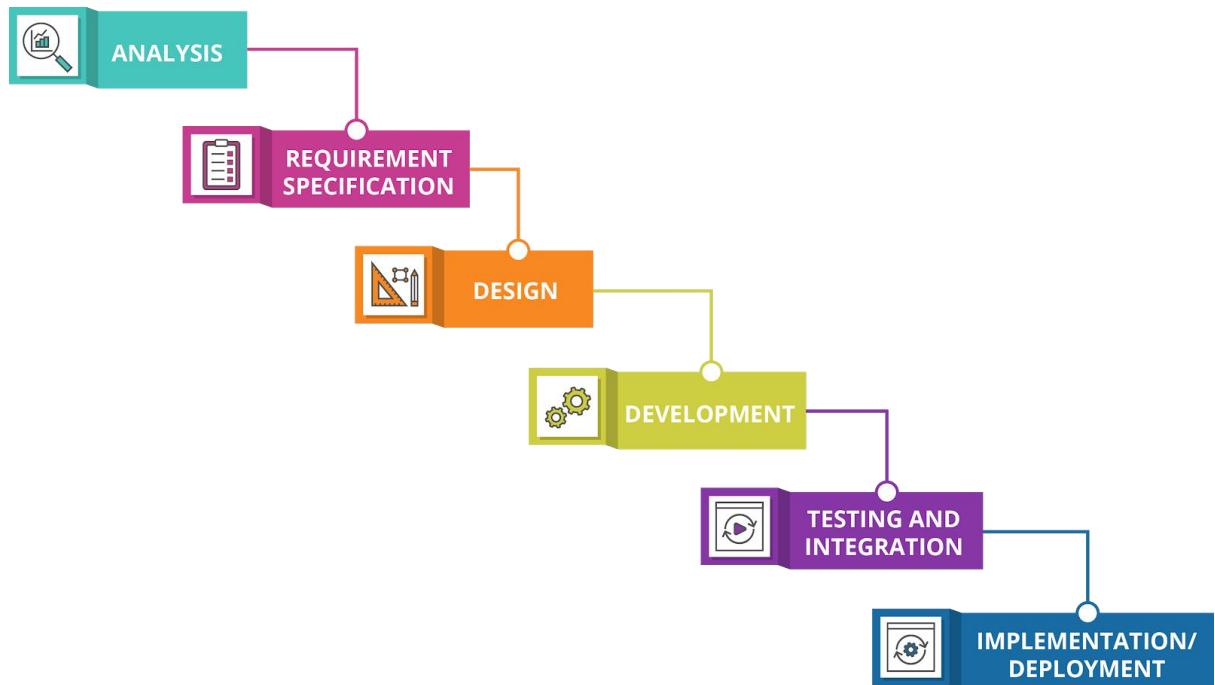
### 2.1.2.2 Hvorfor smidig metode?

I Agile Manifesto (Beck et al., 2001) beskrives 4 grunnverdier for smidige utviklingsprosesser. Disse lyder som følger:

1. **Individer og interaksjoner** fremfor prosesser og verktøy
2. **Fungerende programvare** fremfor omfattende dokumentasjon
3. **Kundesamarbeid fremfor** kontraktsforhandlinger
4. **Respons til forandring** fremfor å følge en plan

Vår gruppe følte at disse verdiene samsvarte best med hvordan vi ønsket å gå fram med prosjektet. Punkt nr. 1 belyser viktigheten av å respondere til hvordan individer opplever produktet da det er disse som driver utviklingsprosessen framover gjennom krav. Gruppa ønsket alltid å lytte til kundens endrede ønsker underveis fremfor å følge en planlagt prosess til punkt og prikke. Punkt nr. 2 formidler at selv om dokumentasjon og planlegging er viktig, er det enda viktigere at produktet fungerer. Gruppa har selvfølgelig dokumentert prosjektet fra start til slutt, men ikke på en måte som virker overflødig eller forhindrende for utviklingen av produktet. Nr. 3 promoterer kontinuerlig samarbeid med kunden, og nr. 4 oppsummerer ved å belyse at det er viktig å være åpen til forandring ved behov. Etter å ha vist fram et inkrement i prosjektet, har NBF endret krav og ønsker underveis i prosjektet. Gruppa har

alltid vært åpen for dette, og disse har vært mye enklere å implementere på grunn av vårt valg av metode.



*Figur 2.1.2.2a: The Cascading Costs of Waterfall, 2019, (<https://medium.com/@joneswaddell/the-cascading-costs-of-waterfall-5c3b1b8beaec>)*

Dersom vi hadde valgt en tyngre metode, som for eksempel fossefallsmodellen (figur 2.1.2.2a), ville det for eksempel ha vært mye vanskeligere å gå et steg tilbake i prosessen. Fossefallsmodellen krever mye planlegging før prosjektets oppstart og opererer med mye strengere faser enn smidige metoder, og man bør være ferdig med én fase før man går over til den neste. I tillegg jobbet gruppa i en begrenset tidsperiode, noe som smidige metoder egner seg bedre til.

### 2.1.2.3 Roller

Da prosjektet består av frontend og backend, ble det naturlig å fordele rollene på disse områdene. Likevel var det viktig for oss at alle hadde en generell overordnet forståelse av de forskjellige områdene, så vi passet alltid på å forklare hverandre hva vi drev med og hvordan ting fungerte i møtene. Dokumentskriving var også en viktig del av prosjektet, om ikke den viktigste. Her bidro alle etter evne og behov.

På frontend var det hovedsakelig Pia, Erik og Sepideh som hadde ansvaret. Pia arbeidet mest med applikasjonens utséende og mindre funksjonaliteter som for eksempel synlig respons ved trykk på en knapp. Da Erik allerede hadde litt erfaring med React fra før, samt noe backend-erfaring, jobbet han med de større funksjonalitetene, som for eksempel å få backend og frontend til å kommunisere ved innsendelse av nye kommentarer i poster og lignende. Sepideh jobbet blant annet med applikasjonens utséende, inputvalidering og noen funksjonaliteter i webapplikasjonen, som for eksempel registrering og navigasjon mellom de forskjellige hovedkomponentene.

På backend var det hovedsakelig Tommy og Henrik som hadde ansvaret. Tommy jobbet mye med å lage prosjektets backend-metoder og underliggende arkitektur. Henrik startet med å jobbe sammen med Tommy på backend, men hoppet over til frontend etter hvert da det var svært mye å gjøre.

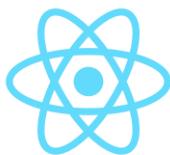
## 2.1.3 Verktøy og språk

---

I prosjektet brukte vi en rekke forskjellige verktøy og språk, både for å utvikle, ha en overordnet oversikt over hvordan vi lå an, kommunisere med hverandre, med mer. I denne delen går vi igjennom alle disse og forklarer hvordan og hvorfor vi valgte akkurat disse hjelpemiddlene.

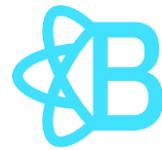
### 2.1.3.1 Frontend

For å utvikle systemets utséende og overfladiske funksjoner, brukte vi hovedsakelig **Visual Studio Code** (VSC) av Microsoft. VSC er et gratis utviklingsmiljø som er spesialiserer seg på å bygge og debugge moderne webapplikasjoner. I tillegg har det et mangfoldig bibliotek med mange populære tilleggsverktøy man kan installere for å optimalisere utviklingsopplevelsen.



Kodespråket vi brukte til frontend-delen av prosjektet var **React**, et JavaScript-bibliotek utviklet av Facebook. React spesialiserer seg på dynamiske brukergrensesnitt og komponentene til disse. **Cascading Style Sheets** (CSS) ble brukt for å stilisere disse komponentene.

Vi brukte også **React-Bootstrap**, et rammeverk for React som gjør det adskillig enklere å lage responsive løsninger. Dette brukte vi da vi ønsket å gjøre webapplikasjonen så mobilvennlig som mulig.



### 2.1.3.2 Backend

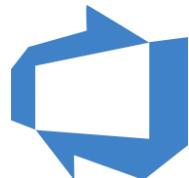
Backend-systemet er utviklet i **Visual Studio**. Oppdragsgiver ønsket at applikasjonen skulle publiseres på **Azure**, og Visual Studio har innebygget støtte for publisering av webapplikasjoner til **Azure Cloud**, så derfor ble dette et naturlig valg. Det finnes også mange nedlastbare pakker (disse heter NuGet packages) som forenkler integrering med blant annet database og filesystem. Her brukes da henholdsvis **Entity Framework** og **Azure Blob Storage**.



Rammeverket vi har brukt i backend-systemet er **ASP.NET Core**, og dette er en del av Microsoft sin egen **.NET-plattform**. Vi har brukt **C#** som er Microsoft sitt eget programmeringsspråk, og som er en del av denne plattformen. Selve kodespråket er basert på Java og Microsoft sitt programmeringsspråk **C++**. Siden vi har brukt Visual Studio som verktøy, ble dette kodespråket det naturlige valget. Det eksisterer også mye dokumentasjon og kodeeksempler vi har kunne benytte oss av for å lære mer om diverse metoder og teknikker.

### 2.1.3.3 Administrative verktøy

For å holde en generell oversikt over fordeling av oppgaver og kode for både backend og frontend brukte vi **Azure DevOps**, også et Microsoft-produkt. Det støtter versjonskontroll, rapportering, kravspesifisering, release-management, oppgavefordeling og mye mer.



Til versjonskontroll har vi brukt **Git**, et versjonskontrollsysten skrevet med åpen kildekode basert på Linux-prosjektet.



**Google Drive** var også et verktøy vi benyttet oss mye av da det lot oss opprette og dele dokumenter i sanntid, og vi brukte det til å produsere styrings- og sluttdokumenter med mer.

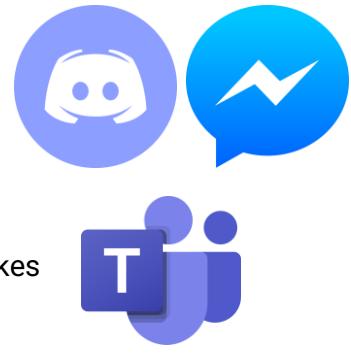
**Discord** og **Facebook Messenger** ble brukt som kommunikasjonsmidler i gruppa der førstnevnte ble brukt til virtuelle møter i gruppa og annen generell kommunikasjon innad i

gruppa. **Microsoft Teams** ble brukt for direkte kommunikasjon og deling av dokumenter og lignende med oppdragsgiver.



#### 2.1.3.4 Andre verktøy

Å lage skisser for å visualisere nettsidens ytre ble en overraskende viktig del av prosjektet. Da Norges Badmintonforbund ikke hadde noen spesifikke ønsker rundt dette, kunne vi skissere idéene våre og vise fram disse. Til dette brukte vi **wireframe.cc**, et verktøy som lar brukeren designe layouts gjennom klassisk drag-and-drop.



**Diagrams.net (draw.io)** er et diagramprogram som kan brukes til å produsere blant annet flowcharts, databaseskjema, nettverksdiagrammer og UML-diagrammer på nett.



Diagrams.net (draw.io) ble brukt for å lage diverse diagrammer som klassediagram, aktivitetsdiagram og andre UML-diagrammer.

**wireframe.cc**

**Postman** ble brukt for utvikling og testing av API-et. Postman er en «collaboration platform» som forenkler API-utvikling ved å tilby en rekke nytte utviklingsfunksjoner for hvert trinn i utviklingsprosessen. En av hovedfunksjonalitetene til Postman er at man kan bruke det som en API-klient og enkelt sende REST-, SOAP- og GraphQL-forespørsler rett fra programmet. Mange andre nyttige funksjoner finnes også, som: automatisert testing, API-sluttpunktdesign, automatisert dokumentasjon, monitorer, og arbeidsrom.



**Wave Evaluation Tool**, utviklet av WebAIM, er et utvidelsesverktøy til Google Chrome som evaluerer hvorvidt en nettside følger de grunnleggende kravene til universell utforming. I prosjektet ble Wave Evaluation Tool brukt for å teste og passe på at elementer som for eksempel alternativ bildetekst o.l. ble implementert.

## 2.2 Utviklingsprosess

I denne delen går vi nærmere inn på hvordan gjennomføringen og oppbyggingen av prosjektet foregikk, samt hvilke utfordringer vi fikk som gruppe underveis.

### 2.2.1 Prosjektet gjennom fasene

---

Prosjektet hadde tre faser dedikert til hver store del av webapplikasjonen; forumet, kunnskapsportalen og resterende funksjoner. Disse fasene endte gruppa å bruke mest som en ramme til hvordan utviklingen skulle foregå, da det å følge fasene etter punkt og prikke viste seg å bli vanskelig i praksis. I denne delen oppsummeres disse fasene, samt hva som skjedde før og etter disse.

#### 2.2.1.1 Før fasene

Tidlig i januar gjorde gruppa forberedelser til utviklingen av webapplikasjonen. Forprosjektrapporten skulle ferdigstilles, og denne definerer mye av grunnlaget for resten av prosjektet. I denne fasen bestemte vi oss for hvilke verktøy og lignende vi skulle bruke, se kapittel 2.1.3. Da Norges Badmintonforbund allerede hadde god integrasjon med Microsoft Teams for kommunikasjon, ble det naturlig at vi ble inkludert her, se figur 2.2.1.1a. Dialog med Charlotte var hyppig i starten for å få på plass krav til design, funksjoner og lignende. Mange skisser (se kapittel 2.1.1.3) for hvordan webapplikasjonen kunne se ut ble sendt for å få en idé og et grunnlag for systemet å ta utgangspunkt i.



Figur 2.2.1.1a: Første dialog på Microsoft Teams med Charlotte Støelen

Da flere av oss var ukjent med noen av teknologiene vi skulle bruke, brukte vi tid på å lære oss det grunnleggende før vi offisielt startet utviklingen. Vi valgte tidlig i prosessen å splitte backend og frontend. Dette gjorde vi fordi det var et ønske fra forbundet om å utvikle både en webapplikasjon og en mobilapplikasjon. Vi endte opp med å ikke utvikle en separat mobilapplikasjon, men besluttet at webapplikasjonen skulle være egnet for skjermer i alle størrelser.

### 2.2.1.2 Første fase

I prosjektets første fase skulle den vanskeligste og trolig mest omfattende delen av webapplikasjonen utvikles, nemlig forumet. Dette inkluderte også enkelte poster med kommentarfelt under. Ut fra kravspesifikasjonen skulle den første ordentlige fasen foregå fra uke 6-8, men i praksis ble dette fram til og med uke 10. I tillegg gjorde vi også små endringer på dette underveis gjennom hele prosjektet.



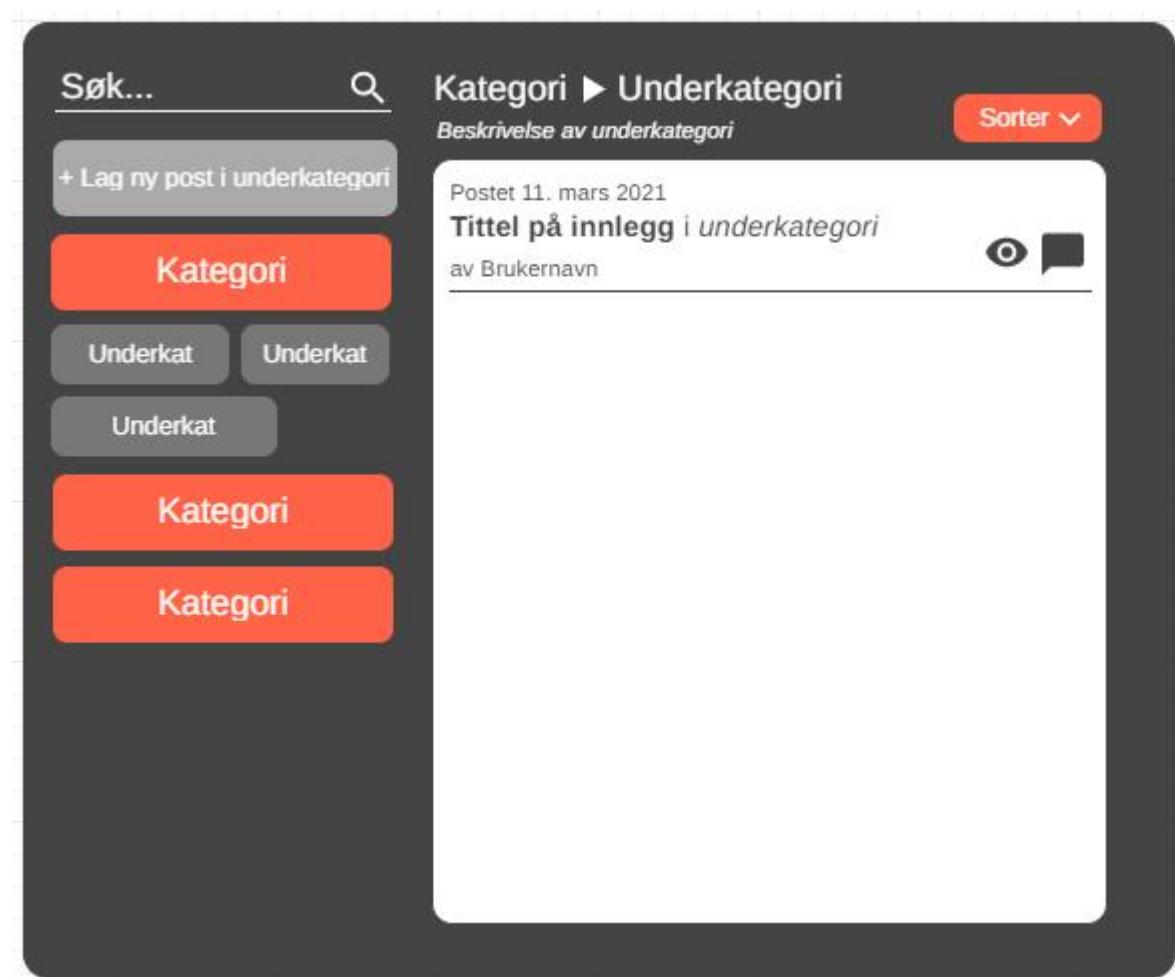
The screenshot shows a forum interface with a navigation bar at the top containing categories: Konkurranse, Kompetanse, Utvikling, and Toppidrett. Below this is a secondary navigation bar with sub-categories: Dommer / oppmann, Seriespill, Lokale turneringer, Rankingturneringer, and Mesterskap. The main content area is titled "Konkurranse" and "Dommer / oppmann". It displays a post by "sepita" with the text "fgg i Dommer / oppmann" and "i går kl. 11:55 av sepita", which has 2 comments. Another post by "henrik" with the text "script i Dommer / oppmann" and "forrige tirsdag kl. 02:21 av henrik" is also shown, with 0 comments. At the bottom right of the interface, there is a "Sorter:" dropdown menu.

*Figur 2.2.1.2a: Forumets utséende etter første fase*

Forumet skulle i dette steget av prosessen, per våre og oppdragsgivers krav og ønsker, ha mulighet for valg av kategorier og underkategorier, sortering av poster, og mulighet for bruker å opprette en ny post og kommentere på andres poster. Det skulle også være en søkefunksjon og sidetallsfunksjon, men disse funksjonene ble ikke ferdige i denne fasen. I tillegg til funksjonene nevnt ovenfor, ble også enkel navigasjon, innlogging og en forside implementert. Figur 2.2.1.2a viser hvordan forumet så ut etter første fase.

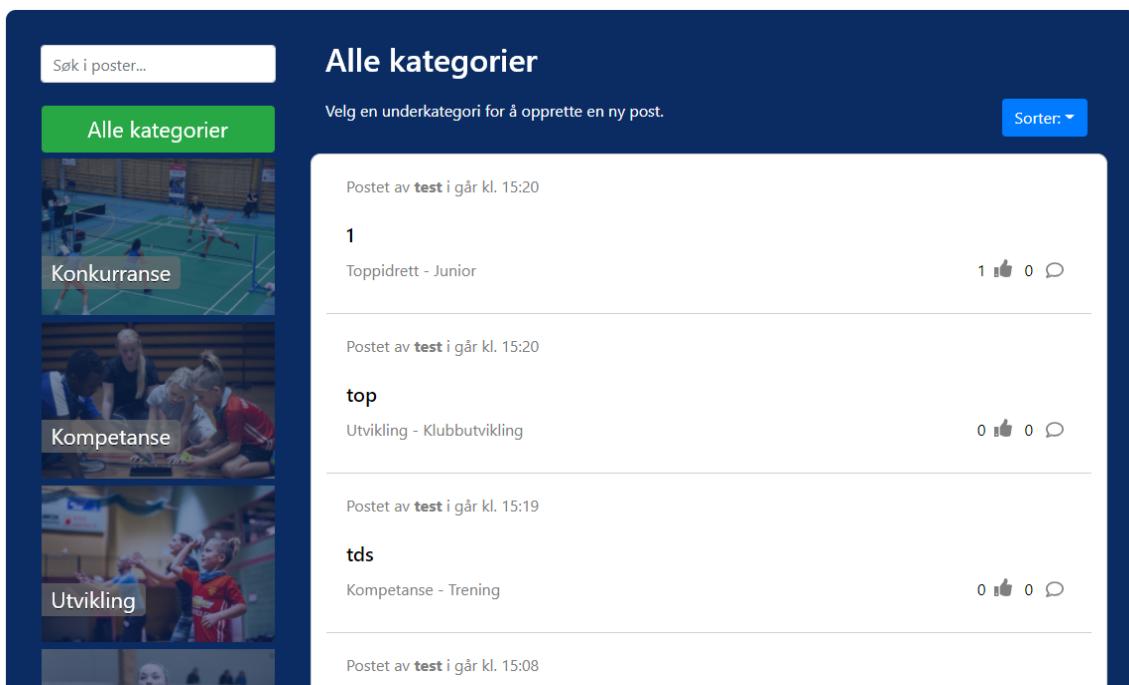
### 2.2.1.3. Andre fase

I prosjektets andre fase var det planlagt at Kunnskapsportalen skulle påbegynnes, men dette gikk ikke eksakt etter planen. Forumet trengte fremdeles mer oppmerksomhet, og oppdragsgiver hadde kommet med flere ønsker etter å ha sett forumets utséende etter første fase (se figur 2.2.1.2a). Altså måtte prosjektets fase i praksis utvides ytterligere. Basert på oppdragsgivers tilbakemeldinger ble en oppdatert skisse for forumet produsert, se figur 2.2.1.3a.



Figur 2.2.1.3a: Skisse til forumets oppdaterte utseende

For å få et bedre samspill mellom Norges Badmintonforbunds eksisterende sider og webapplikasjonen, ble de karakteristiske blåfargen inkorporert som en bakgrunn til blant annet forumet og dets innhold og funksjoner. I tillegg ble det også lagt til bilder til de forskjellige kategoriene i forumet. Søkefunksjonen kom også på plass, og brukerne fikk nå muligheten til å "like" en post. Figur 2.2.1.3b viser hvordan forumet så ut etter andre fase.



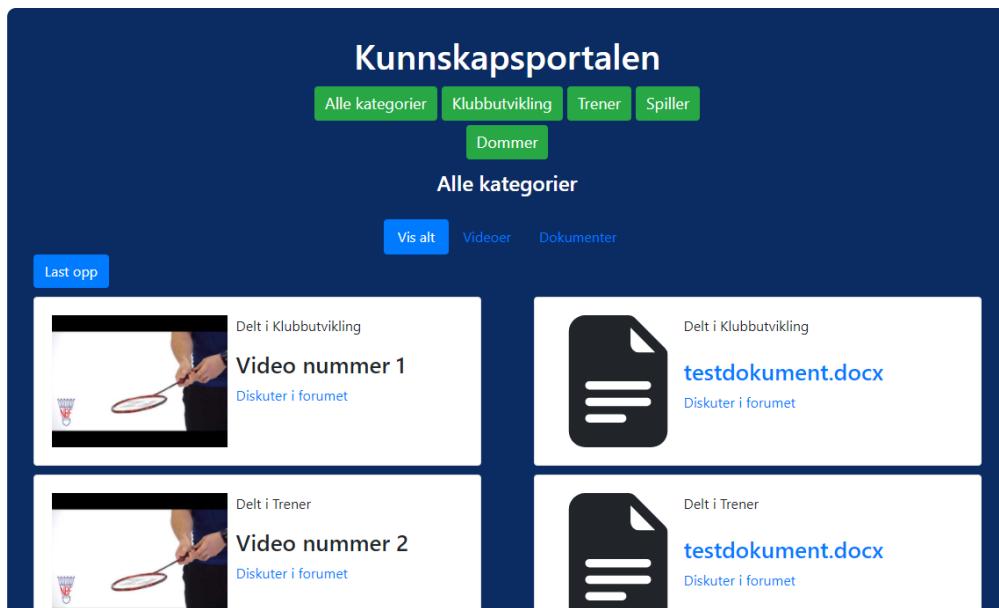
The screenshot shows a forum interface with a sidebar on the left containing three categories: 'Konkurranse' (with a badminton match video thumbnail), 'Kompetanse' (with a competition thumbnail), and 'Utvikling' (with a development thumbnail). The main area is titled 'Alle kategorier' and displays three posts:

- Postet av test i går kl. 15:20**  
1  
Toppidrett - Junior  
1 like 0 comments
- Postet av test i går kl. 15:20**  
top  
Utvikling - Klubbutvikling  
0 likes 0 comments
- Postet av test i går kl. 15:19**  
tds  
Kompetanse - Trening  
0 likes 0 comments

Figur 2.2.1.3b: Forumets utséende etter andre fase

#### 2.2.1.4 Tredje fase

I siste tredjedel av prosjektet hadde vi ikke lenger harde skiller mellom faser, men vi omtaler det som en fase for enkelthetens skyld. I denne fasen ble hovedsakelig kunnskapsportalen utviklet, men også de gjenstående komponentene skulle fullføres. Kunnskapsportalen skulle kategoriseres på lignende måte som forumet, med to overordnede kategorier, videoer og dokumenter, og en rekke underkategorier. For å gjøre det enklere for både oss og Norges Badmintonforbund, skapte vi en løsning som lar administrator laste opp videoer til kunnskapsportalen fra YouTube. Tidligere brukte NBF flere titalls tusen kroner i året på en egen løsning, men ved å bruke YouTube, kunne dette gjøres mye enklere og i tillegg kostnadsfritt. Figur 2.2.1.4a viser hvordan kunnskapsportalen så ut på dette tidspunktet i prosjektet.

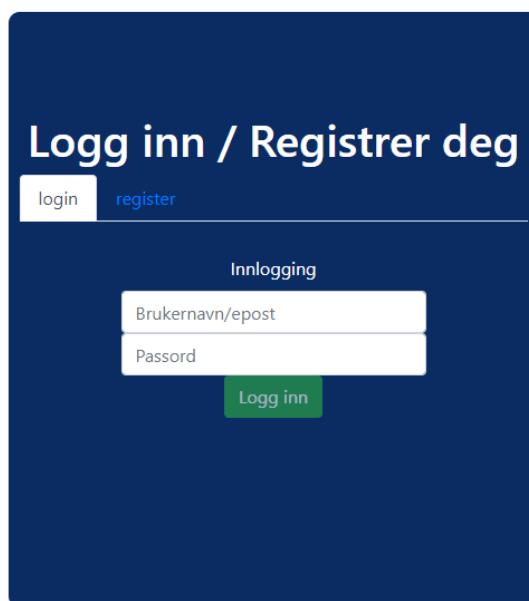


The screenshot shows the 'Kunnskapsportalen' interface. At the top, there are green buttons for 'Alle kategorier', 'Klubbutvikling', 'Trener', 'Spiller', and 'Dommer'. Below them is a blue button for 'Alle kategorier'. Underneath are three more buttons: 'Vis alt' (highlighted in blue), 'Videor', and 'Dokumenter'. A 'Last opp' button is located at the top left. The main area displays four items in a grid:

- Video nummer 1**: Delt i Klubbutvikling. Includes a thumbnail of a person playing badminton.
- testdokument.docx**: Delt i Klubbutvikling. Includes a document icon.
- Video nummer 2**: Delt i Trener. Includes a thumbnail of a person playing badminton.
- testdokument.docx**: Delt i Trener. Includes a document icon.

Figur 2.2.1.4a: Kunnskapsportalen utséende underveis i tredje fase

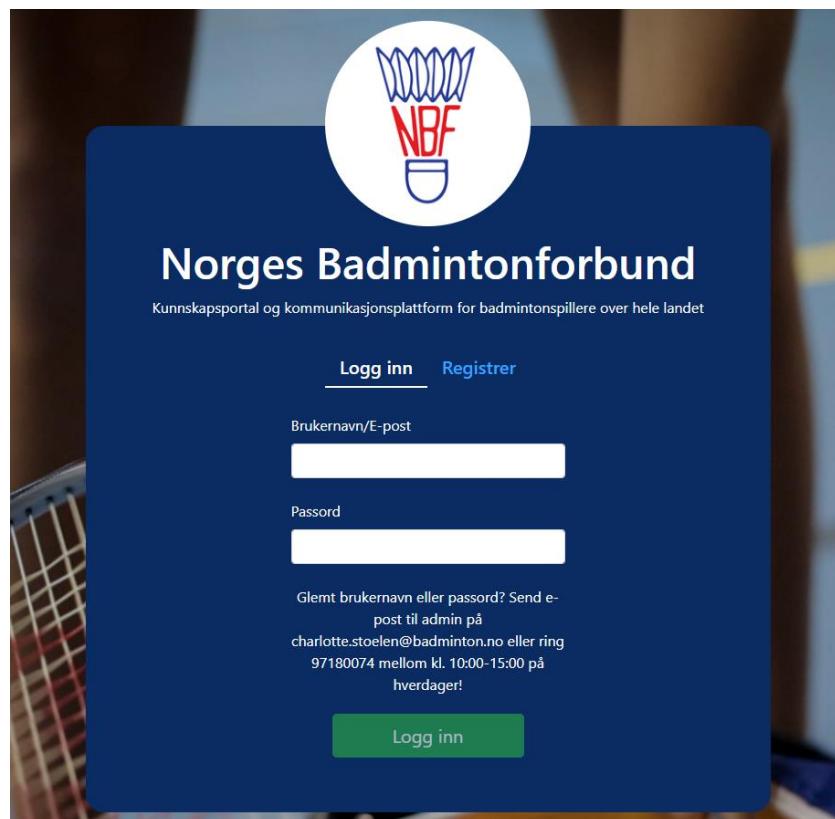
I løpet av denne fasen ble det også vedtatt at vi ikke skulle integrere Idrettens Id med webapplikasjonen. På grunn av mangelfull informasjon og kommunikasjon med Norges Badmintonforbund sin IT-avdeling, følte vi ikke som gruppe at vi hadde evne eller tid til å integrere noe nytt og ukjent som Idrettens Id med webapplikasjonen. Dermed måtte vi skape en egen innlogging- og registreringsmulighet. I starten var denne svært enkel, se figur 2.2.1.4b. Her hadde vi få krav til validering, autentisering og lignende, det som var viktigst i denne perioden var at den fungerte.



The screenshot shows a dark blue 'Logg inn / Registrer deg' (Login / Register) page. It has two tabs: 'login' (highlighted in white) and 'register' (in blue). Below the tabs is a horizontal line. The 'Innlogging' (Logging in) section contains two input fields: 'Brukernavn/epost' (Username/email) and 'Passord' (Password). A green 'Logg inn' (Log in) button is positioned below the password field.

Figur 2.2.1.4b: Loginsidens utséende i starten av tredje fase

Etter hvert ble også innlogging- og registreringssiden ferdig implementert og stilisert, se figur 2.2.1.4c. Vi satte strenge krav for registrering; brukernavn kunne ikke være mindre enn 5 eller mer enn 30 karakterer, passord må være minst 8 karakterer langt med minst ett tall, én stor bokstav og én liten bokstav og lignende. Brukernes passord ble oppbevart trygt og kryptert i systemets databaser. For å autentisere brukere i webapplikasjonen, besluttet vi å bruke JSON Web Token, les mer om dette i webapplikasjonens produktdokumentasjon (kapittel 4).



Figur 2.2.1.4c: Loginsidens endelige utséende

### 2.2.1.5 Refleksjon av utviklingsprosess

Tiden etter fasene ble brukt på å skrive ferdig sluttdokumentasjonen og finpusse prosjektet basert på tilgjengelighetstesting, brukertesting og lignende, samt eventuelle estetiske endringer. I denne tiden kunne vi også se tilbake på utviklingsprosessen og gjøre oss noen formeninger om hvordan det gikk og hva vi kunne ha gjort annerledes.

Til tross for at vi hadde en definert arbeids- og fremdriftsplan, var det vanskelig å følge denne nøyaktig. Forskjellige livssituasjoner innad i gruppa og Covid-19 gjorde det ikke alltid

like lett å bli ferdig med komponenter til et satt tidspunkt. I tillegg finner man alltid forbedringspotensiale i slike prosjekter, og å gå tilbake til noe og forbedre det er fristende etter hvert som man tilegner seg mer kunnskap og kompetanse. Likevel klarte gruppa å gjennomføre prosjektet i sin helhet uten store problemer.

Ideelt sett burde vi ha hatt et bedre skille mellom fasene, men vi er likevel fornøyde med det vi klarte å produsere i løpet av utviklingsprosessen. De fleste av oss hadde ikke særlig kjennskap til React fra før, og med det som grunnlag er vi imponert over at vi klarte å lage et fungerende produkt som også falt i god smak hos Norges Badmintonforbund.

## 2.3 Om kravspesifikasjonen

Kravspesifikasjonen er et av de viktigste dokumentene for utviklingen av et prosjekt. I dette delkapitlet tar vi for oss kravspesifikasjonens rolle i prosjektet og hvordan det eventuelt endret seg gjennom prosjektets løp.

### 2.3.1 Kravspesifikasjonens rolle

---

Kravspesifikasjonen er det dokumentet som gir en detaljert oversikt over de ønskede egenskapene til et system. Disse egenskapene kan omfatte for eksempel brukernes krav til systemet, krav til universell utforming, ytelseskrav med mer.

#### 2.3.1.1 Fra grunnleggende krav til kravspesifikasjon

I samarbeid med Norges Badmintonforbund utarbeidet vi en slik kravspesifikasjon. Denne startet først med noen få enkle krav fra NBF sin side; løsningen skulle være brukervennlig, støtte flere brukere, tilrettelegge for muligheten til å bygge nettverk og ha samtaler, samt støtte publisering av dokumenter, videoer og bilder. Ut fra disse grunnleggende kravene kunne vi sammen med Charlotte Støelen lage enda flere krav som bygget grunnmuren til et fungerende system. Når det kom til å gi brukere muligheten til å danne nettverk og ha samtaler, ble det for eksempel stilt spørsmål rundt hvordan dette skulle fungere, som for eksempel om det skal være som en slags chat, eller skal det være mer som et forum? Skal brukere kunne laste opp filer selv, og skal det i så fall være en grense på hva slags filer? Skal brukere kunne ”reagere” på andre sine poster? Deretter kunne vi skape det endelige dokumentet som definerer kravspesifikasjonen (se kapittel 3). Dette dokumentet brukte gruppa gjennom hele utviklingsprosessen også som en slags sjekkliste og definerte gjøremål ut fra denne, slik at vi hadde oversikt over hva som skulle implementeres og hva som hadde blitt implementert jamfør kravspesifikasjonen. Brukertester ble også utformet mye basert på kravspesifikasjonen. Dermed ble kravspesifikasjonens rolle svært viktig gjennom hele utviklingsprosessen.

#### 2.3.1.2 Kravspesifikasjonens oppbygging

Kravspesifikasjonen vi utarbeidet ble delt inn i to hovedgrupper; funksjonelle krav og ikke-funksjonelle krav. Funksjonelle krav beskriver i praksis *hva* systemet skal gjøre, altså dets

bestemte atferd og funksjoner, mens ikke-funksjonelle krav beskriver *hvordan* systemet skal implementere de funksjonelle kravene.

De funksjonelle kravene ble igjen delt opp i mindre grupper:

- Brukerkrav, hva en bruker ønsker at systemet skal gjøre. Dette kan for eksempel være "Som bruker vil jeg opprette en post"
- Forretningskrav, hva Norges Badmintonforbund ønsker at systemet skal gjøre. Et eksempel på dette er at administrator ønsker å ha muligheten til å slette innlegg
- Systemkrav, hvilke krav systemet skal oppfylle sett fra sluttbrukernes perspektiv. Dette kan være at systemet skal på en eller annen måte opplyse dersom en bruker er slettet.

De ikke-funksjonelle kravene ble også delt opp i mindre grupper:

- Ytelseskrav, hvilke krav systemet skal oppfylle med tanke på ytelse. Dette kan for eksempel være med tanke på hvor mange brukere det håndterer samtidig o.l.
- Sikkerhetskrav, hvilke krav systemet skal oppfylle med tanke på sikkerhet. Å kryptere passord kan være et eksempel på dette.
- Brukervennlighet, hvor lett systemet skal være å ta i bruk.
- Estetiske krav, krav til systemets utséende.
- Generelle krav.

## 2.3.2 Endringer i kravspesifikasjonen

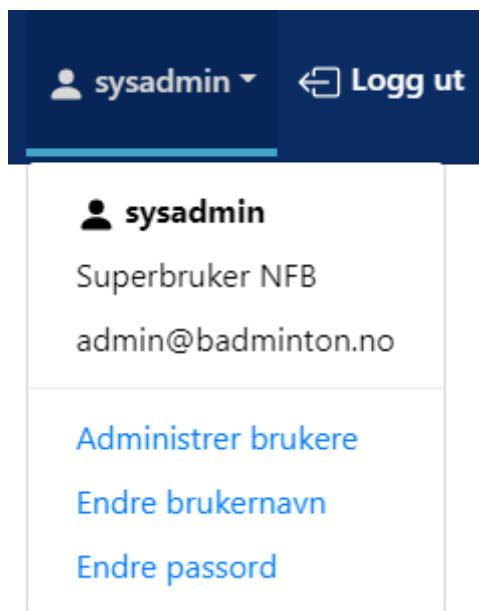
---

Underveis i prosjektet ble det nødvendig å gjøre flere endringer i kravspesifikasjonen. Dette fordi oppdragsgiverens krav endret seg, men også fordi ikke alt gikk etter planen.

### 2.3.2.1 Sluttresultat vs. kravspesifikasjon

I løpet av utviklingen av et hvilket som helst produkt er det gitt at kravene som det ferdige produktet skal oppfylle, endrer seg. Denne webapplikasjonen var intet unntak. Som nevnt tidligere ble ikke krav "F5: Bruker skal kunne logge inn med Idrettens ID" en del av sluttproduktet. Da tiden ble knapp til slutt, og vi ikke mottok den hjelpen vi trengte for å implementere dette, måtte vi skape en egen løsning for innlogging i systemet. Krav "F8:

Brukere skal kunne reagere på videoer” ble ikke som vi hadde sett for oss da det i starten var tiltenkt at hver video skulle ha et eget tilhørende kommentarfelt, lignende YouTube. Vi konkluderte med å heller bruke forumet til dette. Idet administrator laster opp en video i kunnskapsportalen, opprettes det automatisk en ny tråd for denne videoen i forumet. Her kan brukerne kommentere og “like” videoen.



*Figur 2.3.2.1a: Nedtrekksliste for visning av brukerinformasjon*

Krav ”F15: Brukere skal kunne sende meldinger til hverandre” ble heller ikke inkludert i sluttproduktet da det ikke var av høy prioritet. ”F16: Brukere skal kunne se oversikt over sin egen profil” ble ikke til en egen profilsiden, men en nedtrekksliste som viser brukeren informasjonen han har oppgitt om seg selv, se figur 2.3.1.2a. Krav ”F21: Systemet må kunne gi en oversikt over alle poster og kommentarer fra bruker” ble nedprioritert og dermed heller ikke implementert.

### **2.3.2.2 Konklusjon av kravspesifikasjon**

For å konkludere kravspesifikasjonen ble det meste implementert, og sluttproduktet samsvarer i stor grad med kravspesifikasjonen. Dersom vi hadde hatt mer tid og erfaring ville kanskje alt ha blitt implementert, om ikke mer, men sluttproduktet ble et fullverdig og fungerende system til slutt. Norges Badmintonforbund ble tilfredsstilt, og det var noe av det viktigste for oss gjennom hele prosjektet.

## 2.4 Testing

Testing er en grunnleggende prosess for å lage pålitelige og brukbare programvareprodukter som kan foregå på forskjellige stadier av et prosjekt. Det er viktig å velge riktig testtyper og begynne så tidlig som mulig for å kunne avdekke feil i systemet og finne ut av om det oppfyller de kravene som er stilt, samt spare mye penger.

### 2.4.1 Hvorfor teste?

---

Gjennom hele utviklingsprosessen testet vi at webapplikasjonen fungerte som den skulle. Da vi gjorde endringer, kompilerte vi webapplikasjonen på nytt flere titalls ganger om dagen for å sjekke hvordan endringene vi gjorde påvirket resten av systemet. Likevel bør systemet også testes på andre måter. Som utviklere har vi god kjennskap til systemet, men det er ikke mulig for oss for å oppdage alle feil i systemet uten å teste på forskjellige måter. Derfor utførte vi de testene vi tenkte var nødvendige for å avdekke eventuelle feil og forsikre oss om at Norges Badmintonforbunds krav ble møtt.

### 2.4.2 Enhetstesting

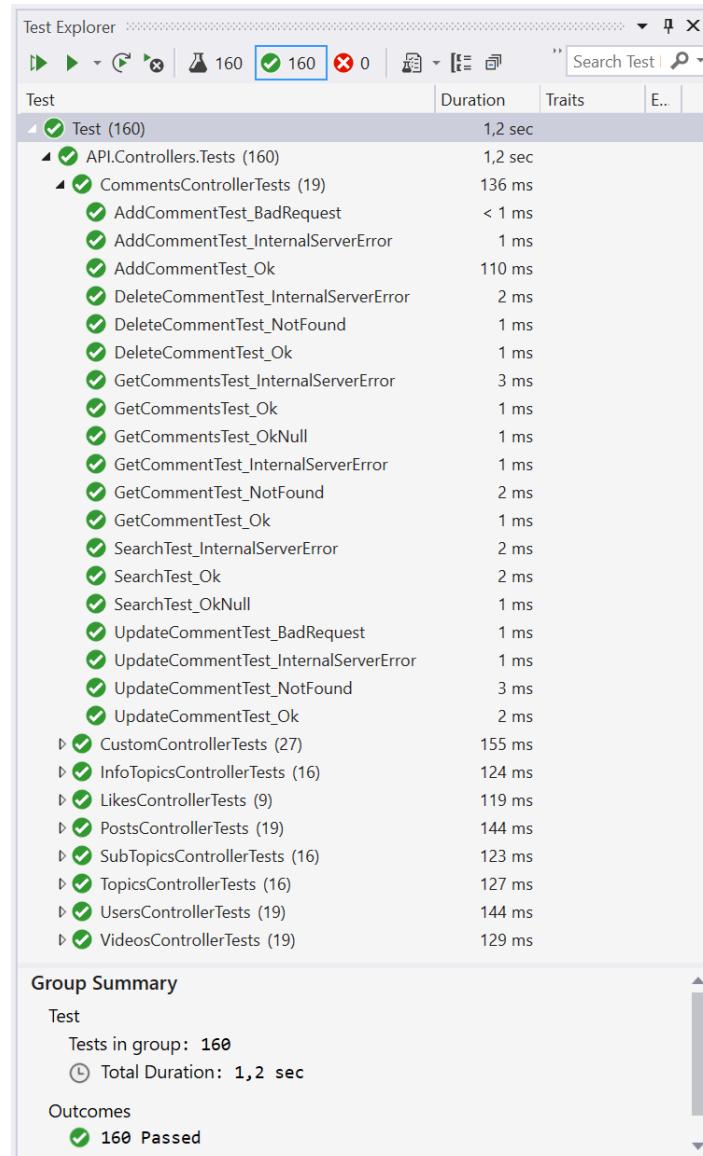
---

Enhetstesting brukes til å teste funksjonaliteten til en applikasjon og sikre at disse funksjonene oppfører seg som forventet (Guru99, u.å.). I backend-delen av applikasjonen utførte vi enhetstester for å teste nettopp dette.

#### Innledning

Til enhetstesting av backend-systemet tok vi i bruk xUnit rammeverket. Dette er gratis og er basert på åpen kildekode. Et annet rammeverk vi har benyttet oss av Moq. Med dette rammeverket kan vi lage objekter som simulerer oppførselen til virkelige objekter (Schadler, 2020). Vi har skrevet enhetstester for alle forskjellige HTTP-responskoder til alle Controllere i hele systemet. Det er en tidkrevende prosess, og vi endte opp med totalt 160 forskjellige

tester. En typisk test tar kun noen millisekunder å kjøre, så selv om vi har mange tester, tar ikke hele enhetstesten mer enn i overkant av ett sekund å fullføre. Som vist på figur 2.4.2a ser vi en liste over alle Controllere som er testet.



Figur 2.4.2a: Alle enhetstestene som er kjørt

Vi har utvidet CommentsControllerTests så vi kan se alle de individuelle testene til denne. Alle de forskjellige testene har funksjoner som ligner mye på hverandre siden hele systemet er basert på et REST API.

## Oppsett

En typisk enhetstest slik vi har skrevet de er basert på et AAA-mønster. Arrange/Act/Assert (AAA) er et mønster for å arrangere og formtere kode i en enhetstest (Telerik JustMock,

u.å.). Fordeler ved å ta i bruk dette er at vi klart og tydelig separer de forskjellige stegene for hver enkelt test. I figur 2.4.2b ser vi en test fra CommentsControllerTests.

```
[Fact]
● 0 references
public async void GetCommentTest_Ok()
{
    // Arrange
    int id = 1;
    var commentDTO = CommentObject.TestCommentDTO();
    var mockRepo = new Mock<ICommentBLL>();
    mockRepo.Setup(repo => repo.GetComment(id)).ReturnsAsync(commentDTO);
    var controller = new CommentsController(mockRepo.Object);

    // Act
    var result = await controller.GetComment(id);

    // Assert
    var actionResult = Assert.IsType<ActionResult<CommentDTO>>(result);
    var okResult = Assert.IsType<OkObjectResult>(actionResult.Result);
    var returnValue = Assert.IsType<CommentDTO>(okResult.Value);
    Assert.Equal(id, returnValue.Id);
}
```

Figur 2.4.2b: Her testes funksjonen for å hente ut en kommentar, og at alt stemmer

Her viser vi en av testene for å hente en kommentar. Denne testen sjekker om vi får tilbake korrekt svar hvis kommentaren vi har etterspurt eksisterer i systemet.

## Arrange

Dette er det første steget. Her setter vi opp variabler, testobjekter og forbereder de nødvendige forutsetningene for testen (Telerik JustMock, u.å.). Ser vi på figur 2.4.2c, kan vi se «Arrange» som en kommentar helt øverst i koden.

```
// Arrange
int id = 1;
var commentDTO = CommentObject.TestCommentDTO();
var mockRepo = new Mock<ICommentBLL>();
mockRepo.Setup(repo => repo.GetComment(id)).ReturnsAsync(commentDTO);
var controller = new CommentsController(mockRepo.Object);
```

Figur 2.4.2c: Arrange for metoden GetCommentTest\_OK

For å hente en kommentar trenger vi en variabelt til ID. Vi trenger også et testobjekt vi skal få tilbake. Dette testobjektet, som vi kan se på figur 2.4.2d, er et eget objekt vi har laget kun for enhetstester og det hentes ikke ut fra den virkelige databasen.

```
6 references | 6/6 passing
public static CommentDTO TestCommentDTO()
{
    var comment = new Comment()
    {
        Id = 1,
        Content = "testkommentar1",
        Date = DateTime.UtcNow,
        UserId = 1,
        PostId = 1
    };
    return new CommentDTO(comment, "sysadmin");
}
```

*Figur 2.4.2d: En testkommentar vi kan bruke som et testobjekt til enhetstesting*

Med Mock simulerer vi interfacet til kommentarer fra Business Logic Layer-klassene, slik at vi kan benytte oss av tilhørende metode videre i testen. Når vi setter opp dette Mock-objektet, sender vi ID-variabelen til metoden og sier at den skal returnere testobjektet vi har laget. Til slutt lager vi en variabel for Controlleren som tar i bruk dette Mock-objektet.

### Act

Dette steget skal utføre testens faktiske arbeid (Telerik JustMock, u.å.). På figur 2.4.2e ser vi «Act» som en kommentar over kodden.

```
// Act
var result = await controller.GetComment(id);
```

*Figur 2.4.2e: Tar i bruk metoden vi skal teste fra Controlleren*

Her lages det en variabel vi kaller «result» som bruker metoden vi skal teste fra Controlleren vi instansierte med et Mock-object fra Arrange-delen. Vi sender også med ID-variabelen til denne metoden.

### Assert

Dette er det siste steget i denne prosessen. Her skal vi bekrefte at resultatene stemmer (Telerik JustMock, u.å.). Figur 2.4.2f viser hva som gjøres her, og vi kan se «Assert» som en kommentar over disse kodene.

```
// Assert
var actionResult = Assert.IsType<ActionResult<CommentDTO>>(result);
var okResult = Assert.IsType<OkObjectResult>(actionResult.Result);
var returnValue = Assert.IsType<CommentDTO>(okResult.Value);
Assert.Equal(id, returnValue.Id);
```

*Figur 2.5.2f: Til slutt må vi bekrefte at alle resultatene stemmer*

Vi bruker «result» fra Act-delen og bekrefter først at denne metoden er av typen «ActionResult». Med denne bekreftelsen kontrollerer vi at dette resultatet gir oss tilbake et «OkObjectResult» som tilsvarer en «200 OK» HTTP-responskode. Med ok-resultatet lager vi en «returnValue»-variabel som sjekker at vi får returnert et kommentar-objekt slik vi etterspurte. Helt til slutt utføres også en test på dette objektet som sjekker at ID-en vi sendte med metoden tidligere tilsvarer ID-en til objektet vi får tilbake.

### Enhetstester i bruk

Vi har nå sett på hvordan en typisk test er bygget opp i backend-systemet vi har utviklet. Disse testene har vi brukt til å bekrefte at alle metoder fungerer som de skal. Vi har også kjørt testene på nytt når vi har gjort endringer. Dette har hjulpet oss med å plukke opp forskjellige feil vi har gjort, men noen ganger har endringene vært så store at vi faktisk måtte endre testkoden også. Typiske feil som ble oppdaget av tester etter endringer var som regel skrivefeil i tilbakemeldinger fra metoden. På figur 2.4.2g ser vi en tilbakemelding vi får hvis en kommentar ikke blir funnet.

```
var comment = await _commentBLL.GetComment(id);
if (comment != null)
{
    return Ok(comment);
}
else
{
    return NotFound($"Kommentar med ID {id} ble ikke funnet");
}
```

*Figur 2.4.2g: Nederst kan vi se at det returneres en feilmelding om objektet ikke ble funnet*

Siden Controllere i forskjellige metoder ligner mye på hverandre, har det gjerne blitt kopiert litt fra andre metoder og limt inn der vi har hatt brukt for det. Glemte vi å endre teksten på denne tilbakemeldingen, kan vi som vist på figur 2.4.2h se at vi fikk en beskjed fra enhetstesten at denne teksten ikke stemte.

```

Test Detail Summary
✖ API.Controllers.Tests.CommentsControllerTests.GetCommentTest_NotFound
  └ Source: CommentsControllerTests.cs line 99
    └ Duration: 87 ms

  Message:
    Assert.Equal() Failure
    Expected: Kommentar med ID 1 ble ikke funnet
    Actual:   Post med ID 1 ble ikke funnet

  └ Stack Trace:
    CommentsControllerTests.GetCommentTest\_NotFound\(\) line 113
      <>c.<ThrowAsync>b__140_0(Object state)
  
```

*Figur 2.4.2h: Her har vi glemt å endre teksten for tilbakemeldingen i metoden vi har testet*

## 2.4.3 Tilgjengelighetstesting

---

For å finne ut hvorvidt frontend-delen av webapplikasjonen møtte de grunnleggende kravene for universell utforming, ble testing med verktøyet Wave Evaluation Tool gjennomført. Dette er ikke en formell test, men det var likevel et svært viktig moment i utviklingen av webapplikasjonen.

### 2.4.3.1 Universell utforming

At webapplikasjonen følger de grunnleggende reglene for universell utforming betyr at alle brukere, uavhengig av funksjonsnedsettelse kan bruke den. Web Content Accessibility Guidelines (WCAG) har utviklet en gjennomgående liste som tar for seg retningslinjer en nettside bør møte for å definere seg som tilgjengelig for et mangfold av brukere. Web Accessibility In Min (WebAIM) har laget en egen sjekkliste basert på WCAG som tar for seg de grunnleggende retningslinjene en nettside bør møte (WebAIM's WCAG 2 Checklist, u.å.). Legg merke til at disse er retningslinjer og ikke krav.

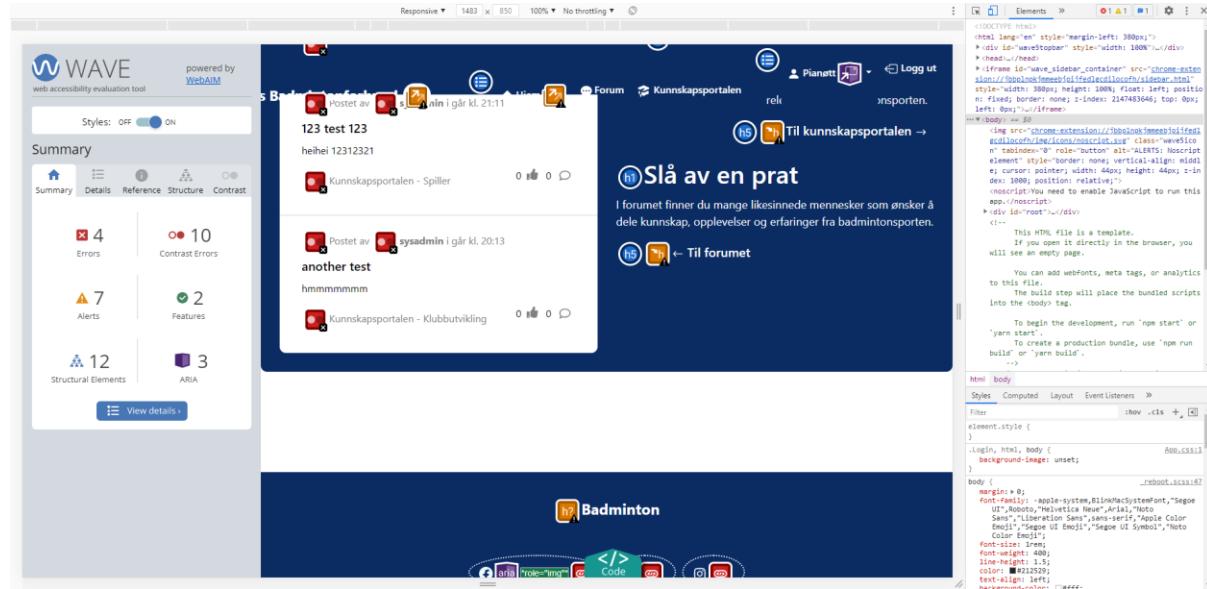
WebAIM har samlet sine retningslinjer i 4 forskjellige hovedgrupper; Perceivable, Operable, Understandable og Robust. Perceivable tar for seg hvorvidt webinnholdet er tilgjengelig for de forskjellige sansene syn, lyd og/eller berøring. Dette kan for eksempel være teksting av videoer for døve eller tekst til tale for blinde. Operable tar for seg i hvilken grad grensesnittet er manøvrerbart med for eksempel bruken av kun et tastatur eller hvor intuitivt det er å navigere rundt på en nettside. Understandable er nesten selvforklarende; det tar for seg hvorvidt en nettside er lett å forstå. Å passe på at tekst er lesbar og forståelig kan være et eksempel her, men også passe på at den ikke har uforutsigbare animasjoner og irriterende "pop ups" og lignende. Til slutt har vi Robust, som passer på om en nettside er kompatibel

med diverse assisterende verktøy. Noen av disse punktene avdekkes gjennom brukertesting, men for å være på den sikre siden, måtte også testing til.

#### 2.4.3.2 Testing

For å sjekke at webapplikasjonen følger de grunnleggende retningslinjene for universell utforming, brukte vi verktøyet Wave Evaluation Tool. Wave Evaluation Tool er henholdsvis utviklet av WebAIM, og figur 2.4.3.2a viser et utklipp av testing med Wave Evaluation Tool i webapplikasjonen. Her er det også viktig å være klar over forekomsten av falske negative; Wave Evaluation Tool er ikke et perfekt program, så dette kunne ikke være det eneste vi lente oss på for å teste webapplikasjonens tilgjengelighet. Manuelle tester av webapplikasjonen ble også gjennomført, som hvor enkelt det var å manøvrere seg rundt med kun tastaturet. Fordi vi brukte Bootstrap til det meste av utviklingen i Frontend, var vi allerede godt på vei med å få nettsiden til å følge retningslinjene da Bootstrap-elementer allerede har mye innebygd støtte for dette. Se mer dokumentasjon på

(<https://getbootstrap.com/docs/4.0/getting-started/accessibility/>).



The screenshot shows the Wave Evaluation Tool interface. On the left, there's a summary panel with counts for Errors (4), Contrast Errors (10), Alerts (7), Features (2), Structural Elements (12), and ARIA (3). The main area displays a webpage from the Badminton website. The page has a dark header with navigation links like 'Forum', 'Kunnskapsportalen', 'Logg ut', and 'insporten.'. Below the header, there's a post by '123 test 123' with the text 'heilhei 12312321'. Another post by 'Kunnskapsportalen - Spiller' with the text 'another test' and 'hmhummmmm' is also visible. At the bottom of the page, there's a footer with the Badminton logo and links for 'Brill', 'Role', 'Img', 'Code', and 'Social'. On the right side of the interface, there's a 'Elements' panel showing the HTML and CSS code for the current page. The code includes styles for various elements like 'wave\_sidebar\_container', 'wave\_sidebar\_header', and 'wave\_sidebar\_content'.

Figur 2.4.3.2a: Utklipp av testing med Wave Evaluation Tool

Figur 2.4.3.2a viser et utklipp av feil, advarsler med mer på forsiden av webapplikasjonen. Mange av feilene var på grunn av manglende "form labels", altså manglende overskrift til inputfelt. Alle inputfeltene i webapplikasjonen har tilhørende overskrifter, men de var ikke koblet til inputfeltene. Dermed måtte disse kobles sammen med en "htmlFor"-attributt i overskriften som henviste til inputfeltets id, se figur 2.4.3.2b.

```
<Form.Label htmlFor="username">Brukernavn/E-post</Form.Label>
<Form.Control
  id="username">
```

*Figur 2.4.3.2b: Demonstrasjon av "htmlFor"-attributten.*

I tillegg ble "role"-attributten for de forskjellige elementene lagt til der det var mest hensiktsmessig. For eksempel fikk påloggingsknappen rollen "submitloginbutton". Role-attributten gjør det for eksempel lettere for skjermlesere å lese av en knapp som en knapp istedenfor en link. Mange steder i webapplikasjonen var det også problemer med tekst med for lav kontrast i forhold til bakgrunnen, se et eksempel for dette i figur 2.4.3.2c. Når det ikke er stor forskjell på tekst og bakgrunn, kan det bli vanskelig å lese da det lett kan blende inn i bakgrunnen for de med synsvansker. Dette ble enkelt løst ved å gjøre teksten mørkere der det gjaldt. Manglende alternativ tekst for bilder o.l var også et problem, men noe som var enkelt å løse ved å legge til "alt"-attributten der det passet.

Postet av **sysadmin** i dag kl. 15:10

Postet av **sysadmin** i dag kl. 15:10

*Figur 2.4.3.2c: Før og etter kontrastendringer*

Det største problemet vi møtte på under tilgjengelighetstesting var bruk av kun tastatur til navigasjon. Enkelte brukere sliter med å bruke mus til å navigere seg rundt i et brukergrensesnitt, og dermed kan navigasjon med tastatur være et godt alternativ. Bootstrap sin dropdown-løsning (se figur 2.3.2.1a) lar brukeren definere dropdown-items, altså listeelementer. Disse kan kun defineres som linker og ikke knapper; dette fungerer med bruk av musepeker, men ikke med tastaturnavigasjon. Tidligere hadde vi definert knapper som åpner opp et modal-vindu i dropdown-funksjonen til å endre både brukernavn og passord, men gjennom testing fant vi ut at dette ikke fungerte med tastaturnavigasjon. Dermed ble vi nødt til å endre på dette. Figur 2.4.3.2d og 2.4.3.2e viser før og etter at vi har rettet opp enkelte tilgjengelighetsfeil.

 12 Errors

- 1 X Missing alternative text  
- 8 X Missing form label             
- 3 X Empty link    

 10 Features

- 1 X Alternative text  
- 8 X Form label       
- 1 X Language  

Figur 2.4.3.2d: Utklipp av feil; Figur 2.4.3.2e: Utklipp etter å ha rettet opp i feil

Oppsummert har tilgjengelighetstesting vært svært nødvendig for å avdekke eventuelle feil ved systemet som kunne ha utfall på brukernes opplevelse av webapplikasjonen. Dersom vi hadde levert produktet uten å teste det og passe på at det følger de grunnleggende retningslinjene for universell utforming, hadde sluttproduktet stridet mot kravspesifikasjonen.

## 2.5 Resultat av prosjekt

I denne delen blir det endelige produktet utviklet i bacheloroppgaven Kunnskapsportalen Badminton vist fram, samt en enkel forklaring for hver komponent som utgjør produktet.

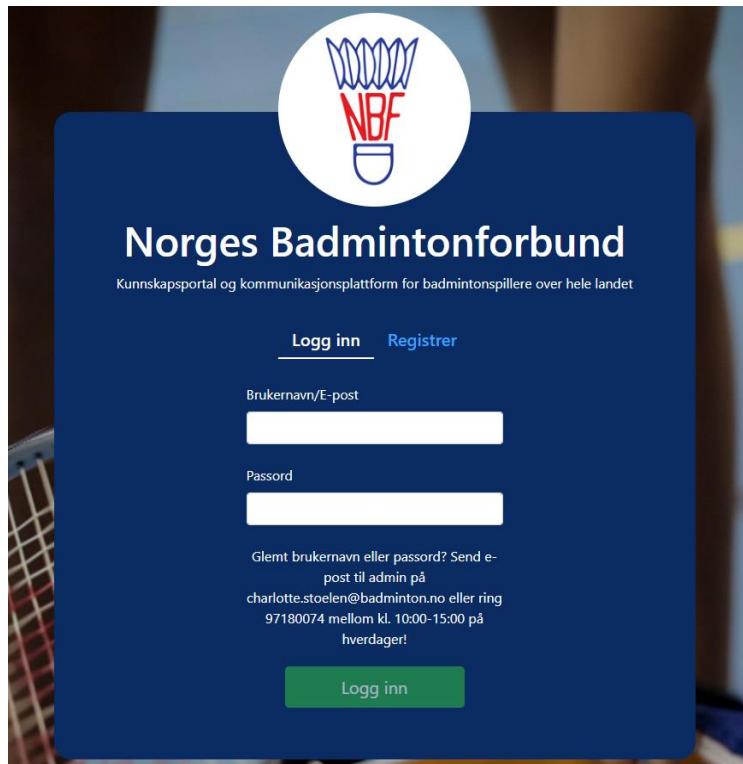
### 2.5.1 Endelig produkt

---

Sluttproduktet inneholder følgende hovedelementer: innlogging og registrering, navigasjon, forside, kunnskapsportalen, forumet og adminsiden.

#### Innlogging og registrering

Innlogging- og registreringssiden er det første en bruker ser når hen skal ta i bruk webapplikasjonen, se figur 2.5.1a. For å aksessere resten av innholdet, må brukeren registrere seg og/eller logge inn i webapplikasjonen.



Figur 2.5.1a: Utklipp av innlogging- og registreringssiden

## Navigering

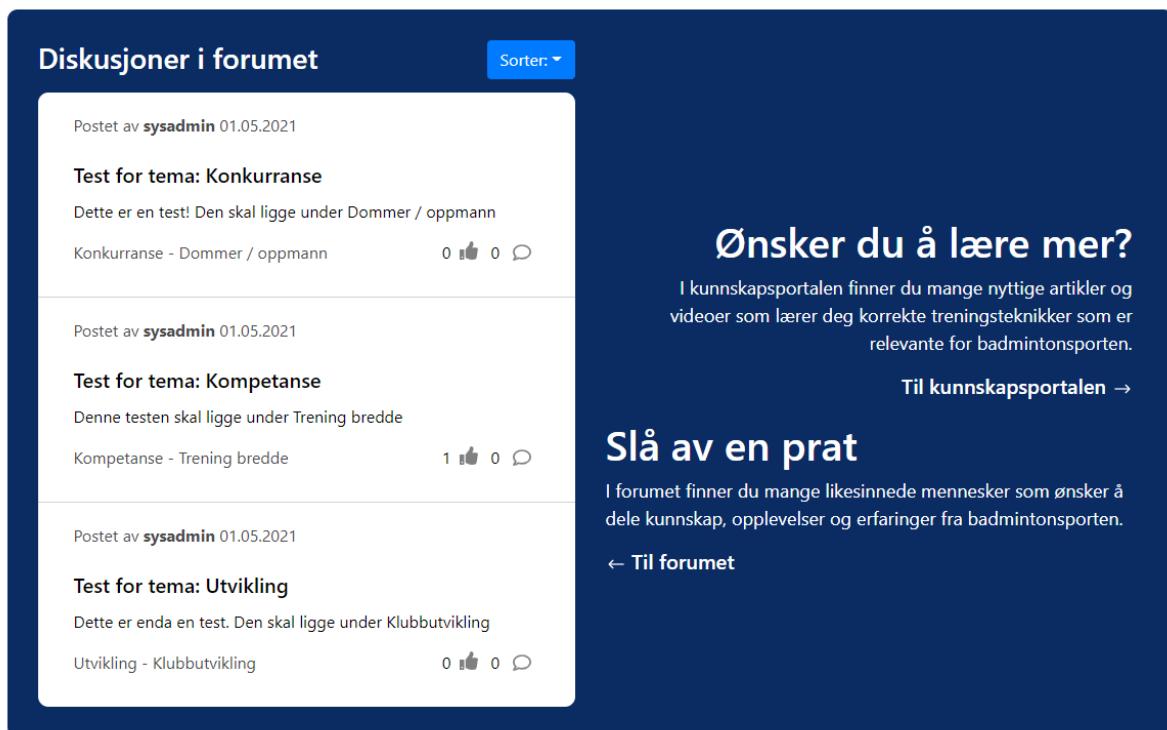
For å navigere seg rundt på de forskjellige sidene, har brukeren en navigasjonsbar øverst på siden, se figur 2.5.1b. Denne er alltid tilgjengelig hvor enn brukeren befinner seg i webapplikasjonen, bortsett fra på innlogging- og registreringssiden.



Figur 2.5.1b: Utklipp av navigasjonsbaren

## Forside

Forsiden er det brukeren ser etter å ha logget inn eller registrert seg. Denne kan ses i figur 2.5.1c. Her kan en se aktivitet fra forumet (tre poster som kan sorteres etter et sett kriterier). Brukeren blir også introdusert for både kunnskapsportalen og forumet med beskrivende tekster til hver.

The image shows a section titled 'Diskusjoner i forumet' (Discussions in the forum). It lists three posts by 'sysadmin' from 01.05.2021. Each post has a title, a brief description, and interaction counts (likes and replies).

- Test for tema: Konkurranse**  
Dette er en test! Den skal ligge under Dommer / oppmann  
Konkurranse - Dommer / oppmann 0 0 0
- Test for tema: Kompetanse**  
Denne testen skal ligge under Trening bredde  
Kompetanse - Trening bredde 1 0 0
- Test for tema: Utvikling**  
Dette er enda en test. Den skal ligge under Klubbutvikling  
Utvikling - Klubbutvikling 0 0 0

To the right of the forum list, there are two promotional boxes:

- Ønsker du å lære mer?**  
I kunnskapsportalen finner du mange nyttige artikler og videoer som lærer deg korrekte treningsteknikker som er relevante for badmintonporten.  
[Til kunnskapsportalen →](#)
- Slå av en prat**  
I forumet finner du mange likesinnede mennesker som ønsker å dele kunnskap, opplevelser og erfaringer fra badmintonporten.  
[← Til forumet](#)

Figur 2.5.1c: Utklipp av forsiden

## Kunnskapsportalen

Kunnskapsportalen, se figur 2.5.1d, er laget slik at videoer og dokumenter kan lastes opp av en administrator. Her kan en legge inn og dele tilgjengelige videoer fra YouTube. Når en video er lastet opp, skapes det automatisk en forumpost der brukere kan diskutere videoen. Videoer og dokumenter er delt hver for seg med tilhørende underkategorier. Innholdet i kunnskapsportalen skal være for å hjelpe badmintonspillere og badmintonentusiaster i form av forklarende videoer og dokumenter om teknikk og lignende, herav navnet kunnskapsportal.

The screenshot shows the 'Kunnskapsportalen' website interface. At the top, there are two green buttons: 'Videoer' and 'Dokumenter'. Below them is a horizontal menu with five blue buttons: 'Alle kategorier', 'Klubbutvikling', 'Trener', 'Spiller', and 'Dommer'. The main content area is titled 'Alle kategorier'. It features a search bar with 'Søk...', a magnifying glass icon, and a dropdown menu labeled 'Sorter: ▾'. On the right, there is a blue button labeled 'Last opp fil'. Two document entries are listed: 1) 'testdokument.docx' (uploaded on 06.05.2021 in Klubbutvikling), which has a download link and a red 'Slett' (Delete) button; 2) 'Testfil.txt' (uploaded on 06.05.2021 in Spiller), also with a download link and a red 'Slett' button. Both documents have a file icon to their right.

Figur 2.5.1d: Utklipp av kunnskapsportalen

## Forum og forumpost

Forumet er skapt for badmintonentusiaster og badmintonspillere slik at de skal kunne diskutere temaer relevante for badmintonporten, se figur 2.5.1e for hvordan den ser ut. Her skal den enkelte kunne ta opp spørsmål og få svar om sportslige temaer, så vel som administrative temaer. Forumet er delt inn i 5 kategorier; konkurranse, kompetanse, utvikling, toppidrett og kunnskapsportalen. Disse er igjen delt opp i en rekke underkategorier som skal dekke de antatte behov definert av Norges Badmintonforbund.

Søk... 

## Konkurranse

Velg en kategori og underkategori for å opprette en ny post.

Sorter: ▾

Postet av **sysadmin** 01.05.2021

**Test for tema: Konkurranse**

Dette er en test! Den skal ligge under Dommer / oppmann

Konkurranse - Dommer / oppmann  0 

---

Postet av **sysadmin** 01.05.2021

**Test for tema: Kompetanse**

Denne testen skal ligge under Trening bredde

Kompetanse - Trening bredde  1 

---

Postet av **sysadmin** 01.05.2021

**Test for tema: Utvikling**

Dette er enda en test. Den skal ligge under Klubbutvikling

Utvikling - Klubbutvikling  0 

  
Konkurranse

  
Kompetanse

  
Utvikling

Alle kategorier

- Dommer / oppmann
- Seriespill
- Lokale turneringer
- Rankingturneringer
- Mesterskap

Figur 2.5.1e: Utklipp av forumet

Figur 2.5.1f viser en post i forumet. Her kan brukere kommentere og like posten, i tillegg til en rekke andre muligheter.

← Tilbake til forum Konkurranse - Mesterskap

Postet av **sysadmin** 02.05.2021 (Redigert 02.05.2021)

**Lorem ipsum**

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Vedlegg: Test.txt  

 2 

**Kommentarer**

Søk...  Sorter: ▾

Postet av **[Slettet bruker]** 02.05.2021

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident.

Vedlegg: Test.txt 



Figur 2.5.1f: Utklipp av en post i forumet

## Adminside

Adminsiden (figur 2.5.1g) består av en tabell med oversikt over alle brukerne. Her kan admin søke gjennom og sortere brukere av webapplikasjonen. Admin har også evnen til å slette brukere, resette passord ved behov og gjøre andre brukere til administrator.

Velg bruker(e)		ID	Brukernavn	Fornavn	Etternavn	E-post	Admin	Passord	Slett bruker(e)
<input type="checkbox"/>	1	sysadmin	Superbruker	NFB		admin@badminton.no	true	<a href="#">Endre passord</a>	
<input type="checkbox"/>	3	test	Test	Testesen		test@testesen.no	true	<a href="#">Endre passord</a>	
<input type="checkbox"/>	5	mats	Mats	Matsen		mats@matsen.no	false	<a href="#">Endre passord</a>	
<input type="checkbox"/>	6	magnus	Magnus	Magnussen		magnus@magnussen.no	false	<a href="#">Endre passord</a>	
<input type="checkbox"/>	8	markus	Markus	Markussen		markus@markussen.no	false	<a href="#">Endre passord</a>	
<input type="checkbox"/>	9	ole	Ole	Olsen		ole@olsen.no	false	<a href="#">Endre passord</a>	
<input type="checkbox"/>	10	nils	Nils	Nilsen		nils@nilsen.no	false	<a href="#">Endre passord</a>	

Figur 2.5.1g: Utklipp fra adminsiden

## 2.6 Avslutning

Utviklingen av Kunnskapsportalen Badminton har vært en lærerik opplevelse. For å avslutte prosessdokumentasjonen ser vi tilbake på det vi har lært, hvordan det har gått og reflektert over dette.

### 2.6.1 Refleksjon og konklusjon

---

Gjennom utviklingen av Kunnskapsportalen Badminton har gruppen lært mye forskjellig. Å få muligheten til å bruke det vi har lært gjennom utdanningen i en tilnærmet reell situasjon har vært en uvurderlig opplevelse.

#### Refleksjon

Gjennom hele utdanningen har vi alle tilegnet oss mye kunnskap og informasjon, og i dette prosjektet fikk vi for første gang ta i bruk det vi har lært i en virkelighetsnær situasjon. Også i utviklingsprosessen av Kunnskapsportalen Badminton har alle i gruppa lært mye, fra hvordan man jobber sammen i en gruppe over en lengre periode, til å lære nye og ukjente teknologier. Ser vi tilbake på tiden som er gått, ser vi at det er mye vi kunne ha gjort annerledes. Vi kunne ha disponert tiden bedre og fått til mer. Likevel er alle sammen meget fornøyde med innsatsen vi har lagt i dette prosjektet, og føler at vi har gjort en god jobb.

Charlotte Støelen har delt følgende utsagn om vårt samarbeid:

"Prosjektgruppen har aktivt tatt ansvar for prosjektet, vært proaktive og kreative i løsninger samt pliktoppfyllende hva gjelder møter, frister og oppgaver avtalt med prosjekteier. Prosjektgruppen har vist stor samarbeidsvilje og engasjement under hele prosjektperioden."

#### Konklusjon

Utviklingen av Kunnskapsportalen Badminton har vært et spennende prosjekt. Vi er alle takknemlige for at Norges Badmintonforbund ønsket å jobbe med oss på dette prosjektet, og føler at vi har fått godt utbytte av det og er fornøyd med innsatsen vi har lagt i det. Norges Badmintonforbund ble også svært fornøyde med sluttproduktet, noe som var meget viktig for oss. Alt i alt har dette vært en svært lærerik opplevelse vi kommer til å ta med oss videre i livet.

Kapittel 3

# **Kravspesifikasjon**

# Innhold

<b>Kravspesifikasjon .....</b>	<b>1</b>
<b>3.1 Kravspesifikasjon.....</b>	<b>3</b>
3.1.1 Presentasjon.....	3
3.1.2 Funksjonelle krav.....	5
3.1.3 Ikke-funksjonelle krav.....	6

## 3.1 Kravspesifikasjon

Kravspesifikasjonen er det dokumentet som tar for seg kravene et sluttprodukt skal oppfylle. I dette dokumentet beskrives kravene for Kunnskapsportalen Badminton.

### 3.1.1 Presentasjon

---

Denne kravspesifikasjonen gjelder bachelorprosjektet i samarbeid med Norges Badmintonforbund. I dette prosjektet skal det designes og utvikles en webapplikasjon med kunnskapsportal og kommunikasjonsplattform for badmintonspillere over hele landet. Norges Badmintonforbund ønsker altså en permanent og ferdig løsning som skal kunne brukes av medlemmene deres, både på PC og mobil.

#### 3.1.1.1 Forord

Hensikten med kravspesifikasjonen er å gi både kunden og oppdragstakerne en detaljert oversikt over ønskede egenskaper som sluttproduktet skal oppfylle. Norges Badmintonforbund hadde en idé om hva de ønsket, men ikke spesifikket krav. Dermed var det viktig med tett kommunikasjon gjennom prosjektets løp. I denne perioden produserte vi minstekravene til et fungerende system og la fram forslag til skisser for hvordan det kunne se ut og fungere. Dette ga dem et innblikk i idéene våre, noe som tillot dem selv å videreutvikle mer spesifikke krav til systemet. Kravspesifikasjonens rolle gjennom prosjektutviklingen har vært svært viktig for oss som oppdragstakere da den hjalp oss med å produsere et fungerende produkt som tilfredsstiller NBF.

#### 3.1.1.2 Om bakgrunnen

Norges Badmintonforbund (NBF) ble etablert i 1938 og har i dag registrert rundt 5500 aktive medlemmer fordelt på 115 klubber i Norge. De er høyeste myndighet på badmintonsportens område i Norge og er medlem i Badminton World Federation (BWF), Badminton Europe, samt Norges Idrettsforbund.

NBF hadde et ønske om å skape en ny plattform for badmintonspillere og entusiaster for badmintonsporten. Dermed ble det aktuelt å sende et prosjektforslag til bachelorstudenter ved Oslo Metropolitan University. Dagens situasjon er at NBF betaler flere titalls tusen i året for en lignende løsning. Denne ville de utvide og gjøre om til en kunnskapsportal og kommunikasjonsplattform i én, slik at badmintonspillere over hele landet enklere kan dele kunnskap, erfaringer og sin lidenskap for badmintonsporten.

### **3.1.1.3 Leserveiledning**

I delkapittel 3.1.2 og 3.1.3 listes de funksjonelle og ikke-funksjonelle kravene til systemet. Disse kravene er igjen delt opp i underkategorier. Der kravene står i kursivt, betyr dette at disse kravene var av lavere prioritet og skulle implementeres dersom det var tid og mulighet.

## 3.1.2 Funksjonelle krav

---

Teknisk funksjonalitet i systemet kan defineres gjennom funksjonelle krav, og den viser hva utviklerne må implementere for at systemet skal ha ønsket oppførsel. En god løsning er å dele opp denne fasen i tre nivåer: forretningskrav, brukerkrav og systemkrav.

### 3.1.2.1 Forretningskrav

Hva NBF ønsker å gjøre

Id	Beskrivelse av krav
F1	Administrator skal kunne laste opp videoer
F2	Administrator skal kunne slette innlegg
F3	Administrator skal kunne gjøre andre brukere til administrator
F4	Brukere må være pålogget for å bruke nettsiden
F5	Brukere skal kunne logge inn med Idrettens ID

### 3.1.2.2 Brukerkrav

Hva sluttbrukere ønsker å gjøre

Id	Beskrivelse av krav
F6	Brukere skal kunne se på videoer
F7	Brukere skal kunne kommentere på videoer
F8	<i>Brukere skal kunne reagere på videoer</i>
F9	Brukere skal kunne opprette tråder (med vedlegg)
F10	Brukere skal kunne redigere sine tråder
F11	Brukere skal kunne kommentere i tråder (med vedlegg)

F12	Brukere skal kunne redigere sine kommentarer i tråder
F13	Brukere skal ha tilgang til informasjon om badminton
F14	Brukere skal ha tilgang til videoer/informasjon om korrekt utførelse og teknikk
F15	<i>Brukere skal kunne sende personlige meldinger til hverandre</i>
F16	<i>Brukere skal kunne se oversikt over sin egen profil</i>
F17	<i>Brukere skal kunne endre personinformasjon</i>
F18	<i>Brukere skal kunne endre passord</i>

### 3.1.2.3 Systemkrav

Hva skal implementeres sett fra bruker sitt perspektiv

Id	Beskrivelse av krav
F19	Systemet må opplyse om andre brukere har redigert sitt innlegg
F20	Systemet må opplyse om en bruker er slettet
F21	<i>Systemet må kunne gi en oversikt over alle poster og kommentarer fra bruker</i>

---

### 3.1.3 Ikke-funksjonelle krav

Ikke-funksjonelle krav er spesifikasjoner som beskriver systemets operative egenskaper. Krav til blant annet ytelse, sikkerhet og brukervennlighet er derfor viktige her.

#### 3.1.3.1 Ytelseskrav

Systemets krav til best mulig ytelse

Id	Beskrivelse av krav

I1	Filer skal ha en størrelsesgrense
I2	Filer skal kun være av bestemte filtyper

### 3.1.3.2 Brukervennlighet

Krav til brukernes opplevelse

Id	Beskrivelse av krav
I3	Systemet skal være enkelt å lære seg og bruke
I4	Systemet skal følge grunnleggende retningslinjer for universell utforming

### 3.1.3.3 Sikkerhetskrav

Krav til systemets sikkerhet

Id	Beskrivelse av krav
I5	Passord skal krypteres

### 3.1.3.4 Estetiske krav

Krav til hvordan systemet skal se ut

Id	Beskrivelse av krav
I6	Systemet skal ha et samspillende utséende med badminton.no

### 3.1.3.5 Generelle krav

Generelle krav til systemet

Id	Beskrivelse av krav
I7	Brukere skal ha en unik Id
I8	Systemet skal ha en egen administrator/superbruker
I9	Systemet skal være responsivt

Kapittel 4

# **Produktdokumentasjon**

# Innhold

<b>Produktdokumentasjon.....</b>	<b>1</b>
<b>4.1 Innledning.....</b>	<b>3</b>
4.1.1 Forord.....	3
4.1.2 Beskrivelse av program.....	3
<b>4.2 Funksjonell systembeskrivelse.....</b>	<b>4</b>
4.2.1 Komponenter .....	4
4.2.2 Brukeropplevelse .....	29
<b>4.3 Programmets oppbygging og virkemåte .....</b>	<b>31</b>
4.3.1 To applikasjoner - én løsning .....	31
4.3.2 Frontend .....	32
4.3.3 Backend .....	40

## 4.1 Innledning

Målet med produktdokumentasjon er å gi et teknisk innblikk i de forskjellige delene av webapplikasjonen.

### 4.1.1 Forord

---

Dette dokumentet er ikke en brukermanual (dette ligger i kapittel 6), men ment for de som skal drive med drift, vedlikehold og videreutvikling av webapplikasjonen. Det er derfor forventet at leserne besitter god teknisk kompetanse for å kunne dra maksimalt utbytte av dokumentet.

### 4.1.2 Beskrivelse av program

---

Produktet vi har laget er en webapplikasjon med forum og kunnskapsportal i én - badmintonspillere og badmintonentusiaster kan diskutere relevante temaer i badmintonmiljøet med hverandre, og Norges Badmintonforbund kan dele kunnskap om badmintonsporten, som for eksempel korrekt utførelse av diverse øvelser og god teknikk.

## 4.2 Funksjonell systembeskrivelse

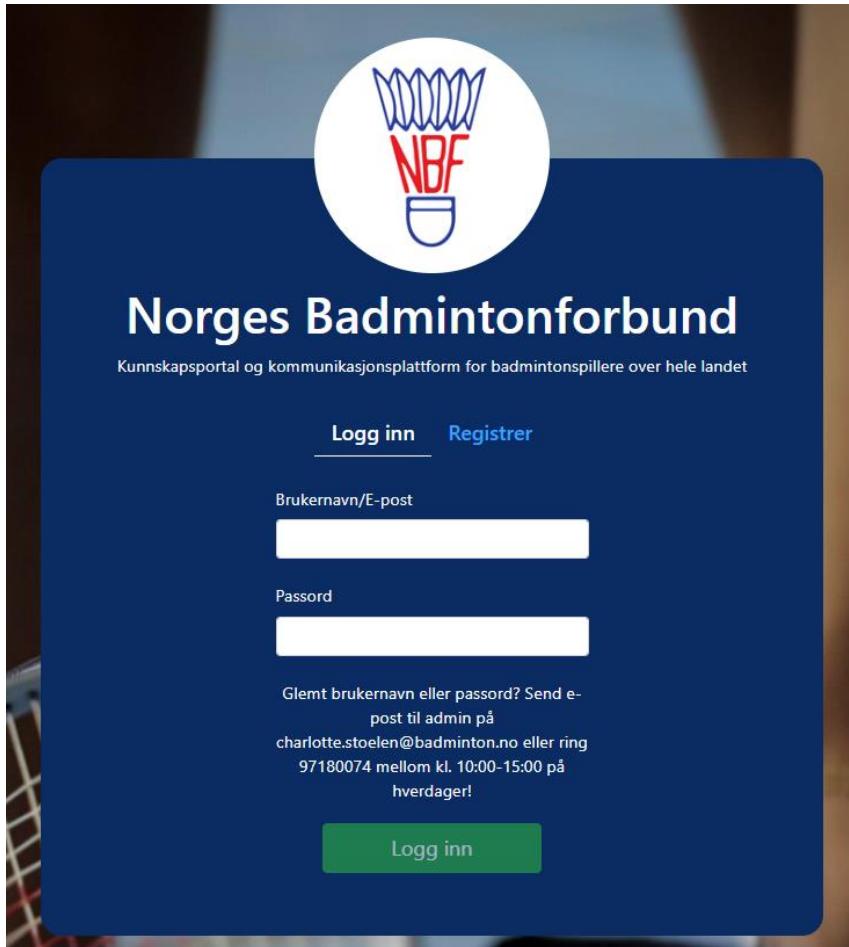
I dette underkapitlet går vi gjennom de forskjellige hovedkomponentene og underkomponentene som utgjør webapplikasjonens helhet.

### 4.2.1 Komponenter

---

Webapplikasjonen er bygget opp av mange komponenter og underkomponenter som sammen fungerer som en helhet. I dette delkapitlet går vi gjennom disse og gir en beskrivelse av hvordan de fungerer og er bygget opp.

#### 4.2.1.1 Innlogging

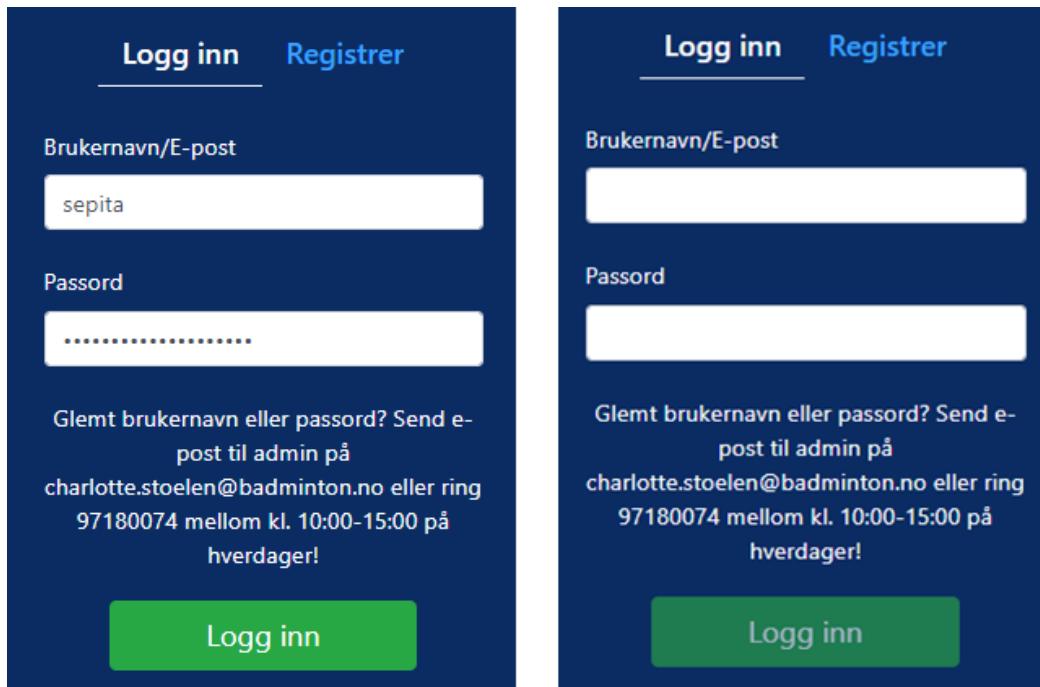


The screenshot shows the login page for the Norges Badmintonforbund website. At the top center is the NBF logo. Below it, the text "Norges Badmintonforbund" and "Kunnskapsportal og kommunikasjonsplattform for badmintonspillere over hele landet". There are two buttons: "Logg inn" (highlighted in blue) and "Registrer". Below these are two input fields: "Brukernavn/E-post" and "Passord". To the right of the "Passord" field is a link: "Glemt brukernavn eller passord? Send e-post til admin på charlotte.stoelen@badminton.no eller ring 97180074 mellom kl. 10:00-15:00 på hverdager!". At the bottom is a green "Logg inn" button.

Figur 4.2.1.1a: Innloggingssiden

Innloggingssiden har to inputfelt, et for brukernavn/e-post og et for passord, samt en knapp for å logge inn (figur 4.2.1.1a). For å aktivere logg inn-knappen må brukeren fylle ut begge

input-feltene da den innebygde valideringsmetoden sjekker at begge feltene ikke er tomme (figur 4.2.1.1b). Når brukeren trykker på logg inn-knappen, sjekkes brukernavn/e-post og passordet mot databasen.



**Logg inn**
**Registrer**

**Brukernavn/E-post**

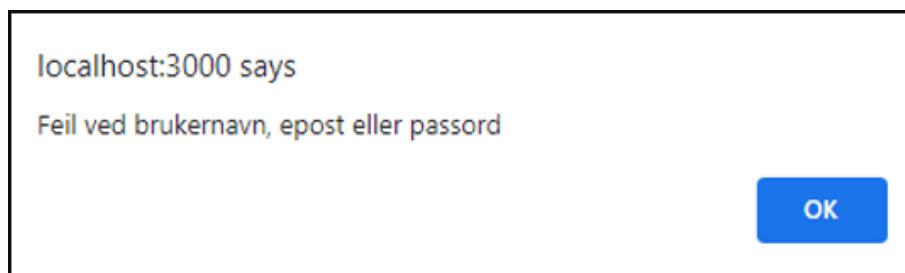
**Passord**

Glemt brukernavn eller passord? Send e-post til admin på  
 charlotte.stoelen@badminton.no eller ring  
 97180074 mellom kl. 10:00-15:00 på  
 hverdager!

**Logg inn**

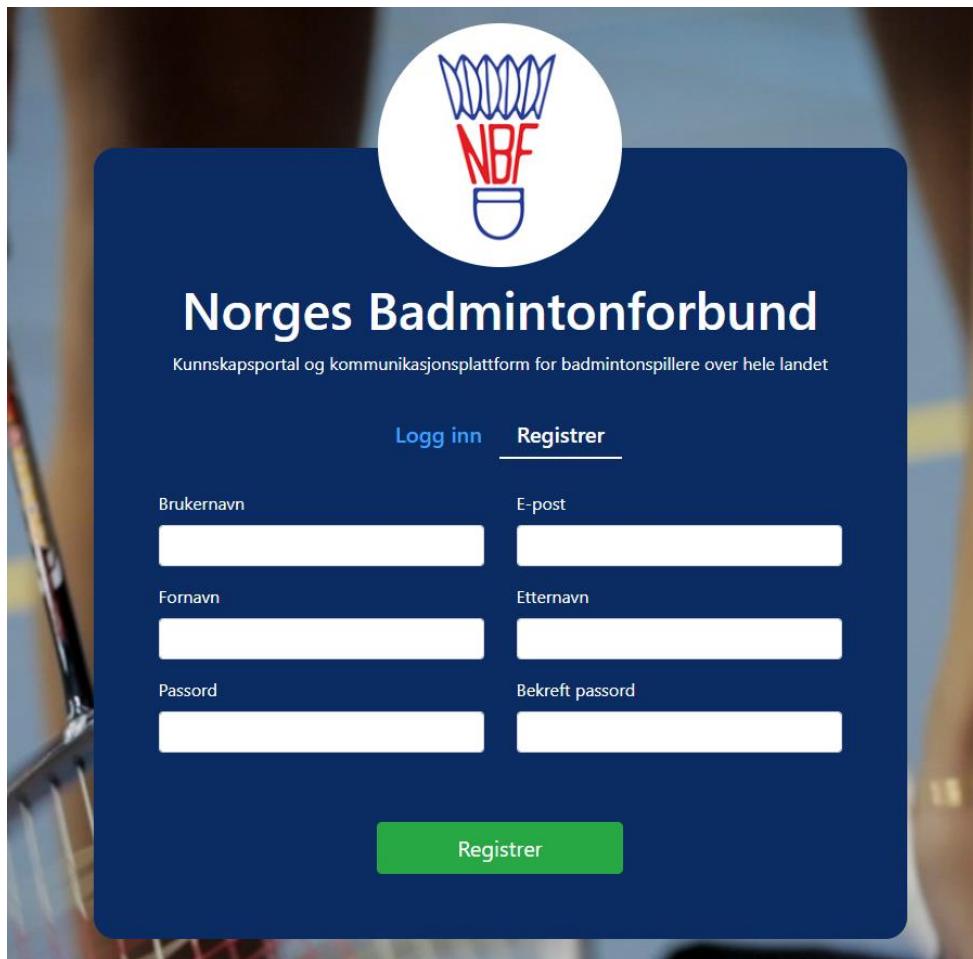
Figur 4.2.1.1b: Deaktivert-knapp

Ingen elementer av webapplikasjonen er tilgjengelig uten autentisering. Dersom brukernavn/e-post eller passord ikke stemmer, sender applikasjonen en melding om feil for dette (figur 4.2.1.1a). Ved innlogging vil brukeren bli sendt direkte inn til forsiden.



Figur 4.2.1.1c: Melding om feil ved brukernavn/e-post eller passord

#### 4.2.1.2 Registrering av nye brukere



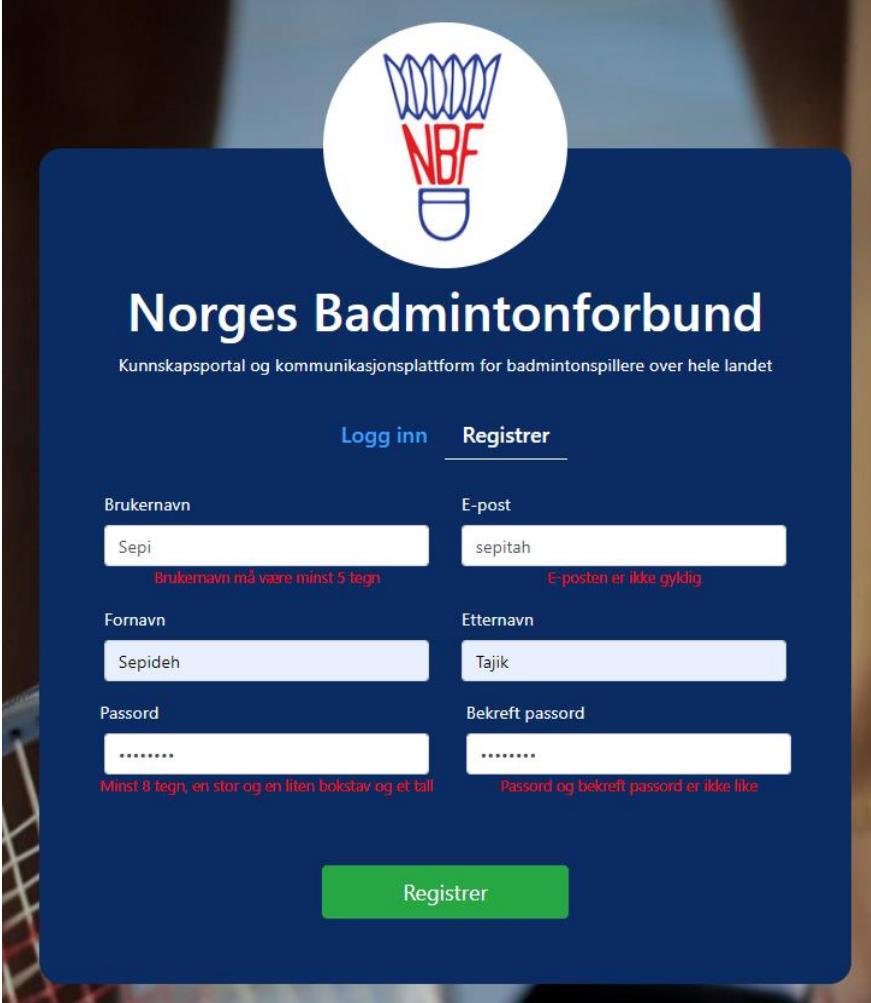
Figur 4.2.1.2a: Registreringssiden

For å kunne bruke applikasjonen for første gang, må brukere registrere seg i systemet. Brukere kan navigere fra innloggingssiden til registreringssiden ved å trykke på registrer-knappen øverst på innloggingssiden og omvendt (Figur 4.2.1.2a). Registreringssiden består av seks input-felt for brukernavn, e-post, fornavn, etternavn, passord og bekreft passord. Registrering gjøres ved å trykke på registrer-knappen nederst på siden, men det er nødvendig å fylle inn alle input-feltene før registrering. Hvis brukeren prøver å registrere tomme felter, blir en feilmelding vist (figur 4.2.1.2b).

Logg inn	Registrar
<b>Brukernavn</b>	<b>E-post</b>
<input type="text"/>	<input type="text"/>
Brukernavn må fylles ut	
<b>Fornavn</b>	<b>Etternavn</b>
<input type="text"/>	<input type="text"/>
Fornavn må fylles ut	
Etternavn må fylles ut	
<b>Passord</b>	<b>Bekreft passord</b>
<input type="password"/>	<input type="password"/>
Passord må fylles ut	
Bekreft passord må fylles ut	
Registrar	

Figur 4.2.1.2b: Feilmeldinger for tomme felter

Etter at registrer-knappen trykkes på, valideres brukernavn, e-post og passord med Yup, et JavaScript-bibliotek for validering i tillegg til en standard regex for passord. Brukernavn må inneholde minst 5 tegn og e-post må følge riktig mønster for å være gyldig. Passordet må inneholde minst 8 karakterer, en stor og en liten bokstav og minst ett tall. For å forhindre eventuelle feil må brukeren bekrefte passordet sitt. Hvis informasjonen som brukeren skriver ikke godkjennes etter validering, vises en beskrivende feilmelding (figur 4.2.1.2c). Feilmeldingen forsvinner så fort input-feltene oppfyller kravet. Etter registrering, blir man automatisk logget inn, og brukere omdirigeres til forsiden av webapplikasjonen.



The screenshot shows the registration form for the Norges Badmintonforbund website. The form includes fields for Brukernavn (Username), E-post (Email), Fornavn (First Name), Etternavn (Last Name), Passord (Password), and Bekreft passord (Confirm Password). Error messages are displayed below each field: 'Brukernavn må være minst 5 tegn' (Username must be at least 5 characters) for the username; 'E-posten er ikke gyldig' (Email is not valid) for the email; 'Minst 8 tegn, en stor og en liten bokstav og et tall' (At least 8 characters, one uppercase and one lowercase letter, and a number) for the password; and 'Passord og bekreft passord er ikke like' (Password and confirm password are not the same) for the password confirmation. A green 'Registrer' button is at the bottom.

Figur 4.2.1.2c: Feilmeldingene etter validering

#### 4.2.1.3 Navigasjon

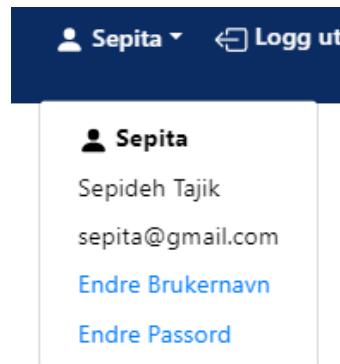
Webapplikasjonen har en enkel global navigasjon (figur 4.2.1.3a). All navigasjon på siden kan gjøres her, og den inneholder logo, navn og linker til de forskjellige sidene. Den er tilgjengelig fra alle sidene i webapplikasjonen, unntatt innlogging- og registreringssiden, og den sørger for at brukere kan navigere seg rundt på nettsiden fra hvor enn de er på en enkel måte.



Figur 4.2.1.3a: Global navigasjon

På høyresiden finnes en nedtrekksmeny ved siden av "Logg ut". Tittelen er her navnet til brukeren som er logget inn. Nedtrekksmenyen inneholder brukerinformasjon som

brukernavn, fullt navn og e-post, og her har bruker også mulighet til å endre sitt eget brukernavn og passord (figur 4.2.1.3b).



Figur 4.2.1.3b: Nedtrekksmenyen inneholder brukerinformasjon.

Endre brukernavn og endre passord i nedtrekksmenyen åpner sider som gir mulighet til å sette nytt brukernavn eller passord (figur 4.2.1.3c). Etter endring av brukernavn eller passord, får bruker en bekreftelse på dette (figur 4.2.1.3d). Hvis nytt passord og bekreft passord ikke er likt, får brukeren en feilmelding om dette (figur 4.2.1.3e).

**Endre brukernavn for Sepita**

Nytt brukernavn

**Bekreft**

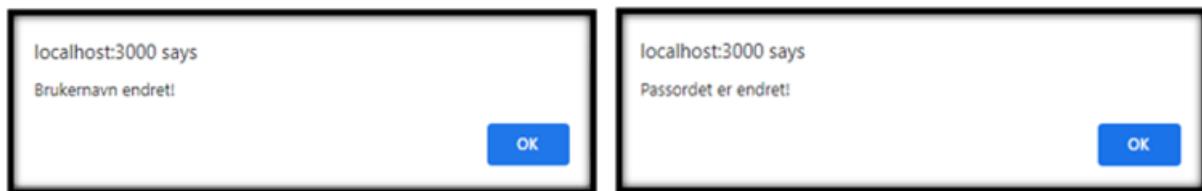
**Endre passord for Sepita**

Nytt passord

Bekreft passord

**Bekreft**

Figur 4.2.1.3c: Sidene for å endre brukernavn og passord.



Figur 4.2.1.3d: Bekreftsesmeldingen for endring av brukernavn og passord

## Endre passord for Sepita

Nytt passord

Bekreft passord

Passord og bekreft passord er ikke like

**Bekreft**

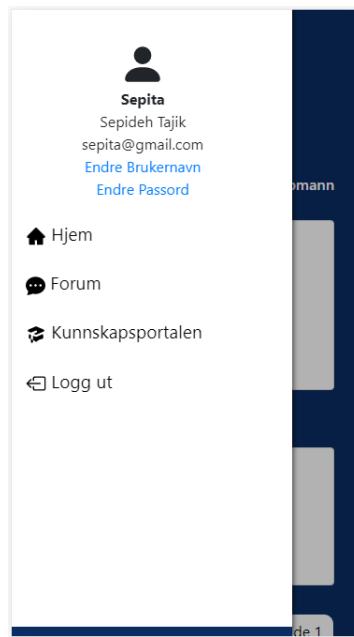
Figur 4.2.1.3e: Feilmelding om feil ved passord og bekreft passord.

Hvert element i navigasjonsbaren har entydige ikoner sammen med tekst for å hjelpe brukere å finne det de leter etter på en enkel måte. Alle ikoner som ble brukt er fra React-icons-biblioteket, og de er assosiert med teksten ved siden av. Valg av ikoner er basert på at brukere gjenkjenner symbolene fra andre steder. For bedre plassering av elementer i navigasjonsbaren på mindre skjermstørrelser, endres den automatisk til en mindre bar med en hamburger-knapp når bredden på siden blir mindre enn 1024 pixler (figur 4.2.1.3f).



Figur 4.2.1.3f: Navigasjonsbar i mindre skjermstørrelse

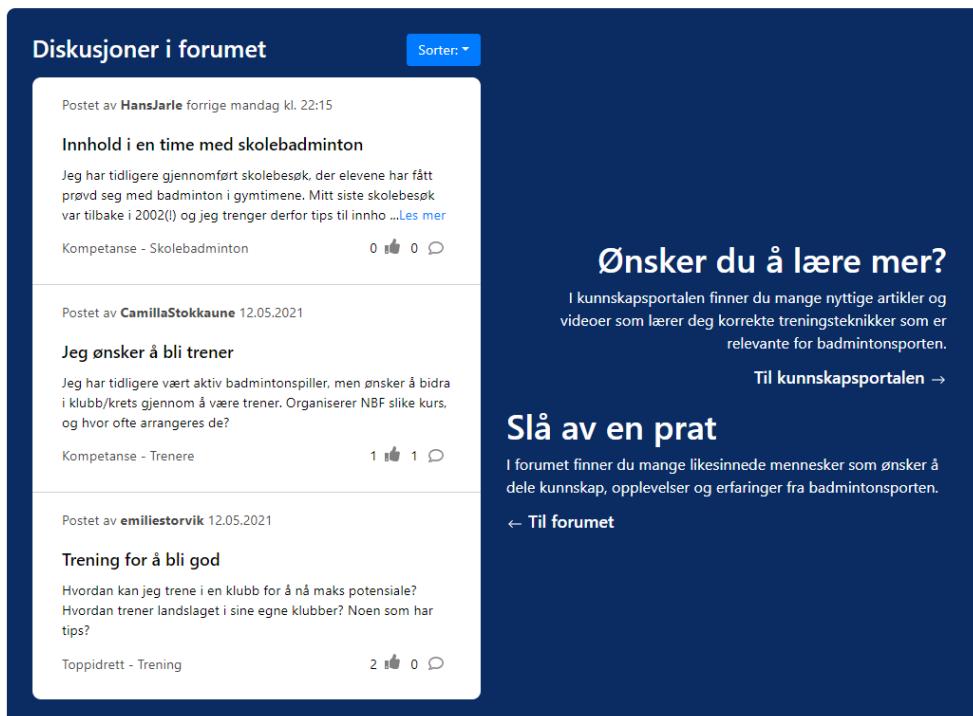
Når hamburger-knappen trykkes, åpnes en sidemeny som inneholder alle de samme linkene som elementene i navigasjonsbaren. Sidemenyen gir samme muligheter for navigasjon til de forskjellige sidene på samme måte som navigasjonsbaren, bare at denne versjonen er bedre egnet for mindre skjermer (figur 4.2.1.3g).



Figur 4.2.1.3g: Sidemeny

For å logge ut av applikasjonen kan man trykke på ”logg ut” knappen øverst i høyre hjørne i navigasjonsbaren (figur 4.2.1.3b). I sidemenyen finnes det også en logg ut-knapp (figur 4.2.1.3g).

#### 4.2.1.4 Forsiden



Figur 4.2.1.4a: Utklipp av forsiden

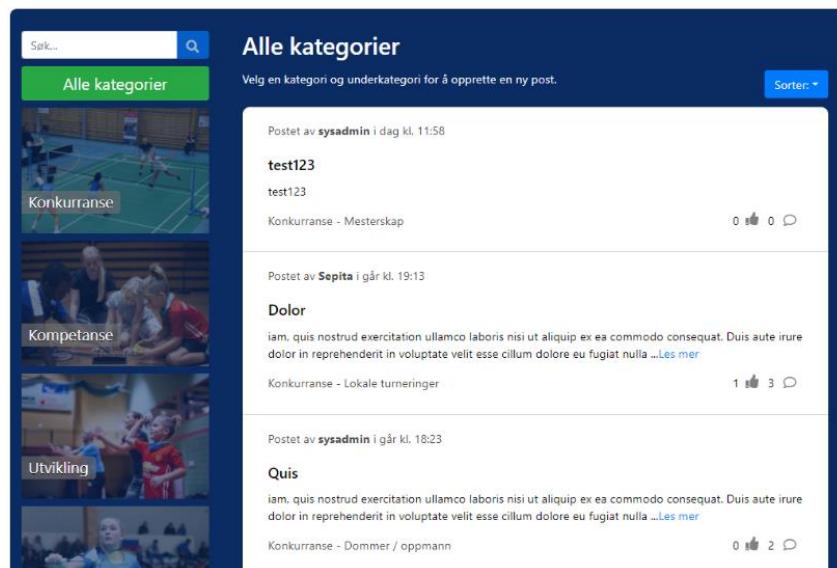
Forsiden er den første siden som vises etter innlogging (figur 4.2.1.4a). På venstre side av forsiden finnes et felt som viser en oversikt over poster fra forumet. Disse postene kan man sortere ved å bruke en egen sorteringsknapp som ligger øverst til høyre over postene. Det er mulig å sortere poster basert på dato, antall liker og antall kommentarer per post. På høyre side av forsiden finner man linker til forumet og kunnskapsportalen, med en kort forklaring av formålet til disse (figur 4.2.1.4b).



Figur 4.2.1.4b: Sorteringsknappen på forsiden

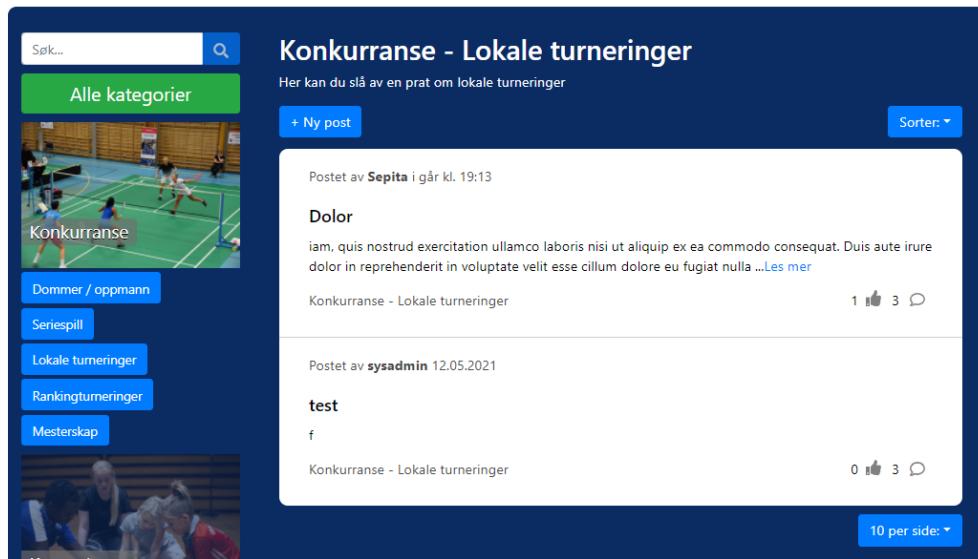
#### 4.2.1.5 Forum

Forumet gir brukere mulighet til å diskutere forskjellige temaer i badmintonporten. Når man først navigerer til forumet, får man en oversikt over alle postene i forumet i høyre felt, og alle hovedkategorier og underkategorier i venstre felt (figur 4.2.1.5a).



Figur 4.2.1.5a: Førstesiden til forumet

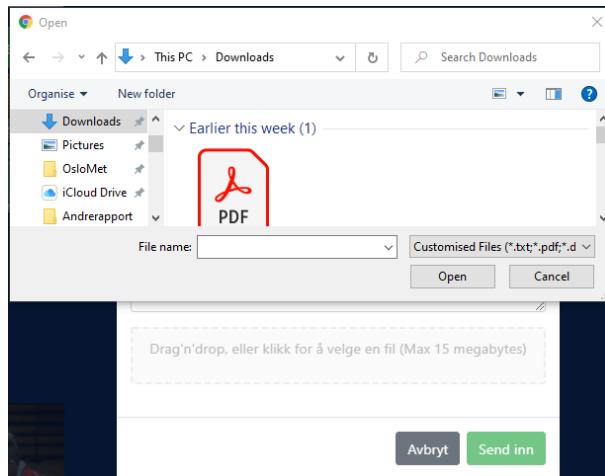
Ved å trykke på en kategori, åpnes de tilhørende underkategoriene for denne kategorien, og man kan se postene til hver enkelt underkategori ved å trykke på en av disse. Her kan man også trykke på ”+ Ny post” knappen for å opprette en ny post (figur 4.2.1.5b).



Figur 4.2.1.5b: Poster i tema ”Konkurranse” med undertema ”Lokale turneringer”

Når man trykker på ”+ Ny post”-knappen, åpnes et nytt vindu med et skjema. For å opprette en ny post, må brukeren lage en tittel og innhold til denne. Tittelen kan være maks 100 tegn, mens posten kan være maks 4000 tegn. I dette skjemaet finnes det også et felt for å legge ved en fil, enten med drag'n'drop (figur 4.2.1.5c) eller ved at man klikker her og velger en fil fra filutforskeren på egen datamaskin (figur 4.2.1.5d). Maks filstørrelse er 15 megabytes, og filtyper som er tillatt inkluderer: .txt, .pdf, .doc, .docx, .xls, og .xlsx (figur 4.2.1.5c).

Figur 4.2.1.5c: Skjemaet for å send opprette en ny post og feilmelding for file type eller størrelse



Figur 4.2.1.5d: Vindu for å velge fil fra filutforskeren i ny post

På høyre side av forumet finnes en sorteringsknapp som gir muligheten til å sortere poster på dato, antall liker og antall kommentarer per poster (figur 4.2.1.5e). Det er et søkefeltet øverst på venstre side av forumet, og her kan brukeren taste inn ønskede søkeord, og trykke "Enter" på tastaturet eller klikke på forstørrelsesglass-knappen. Søkefeltet er også tilgjengelig i kunnskapsportalen og i kommentarfeltet til en post, og alle fungerer på samme måte som forumets søkefelt.

Figur 4.2.1.5e: Søkefeltet og sorteringsknappen illustrert i forumet

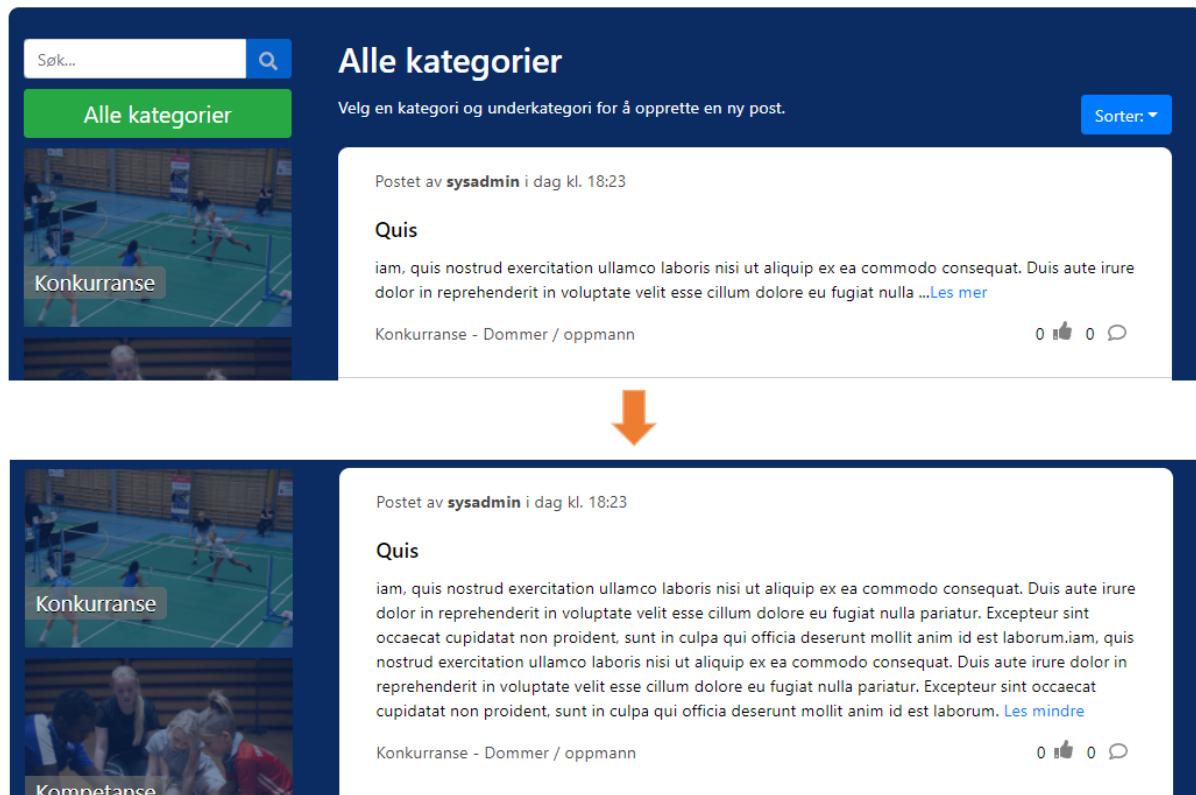
En sidetallsfunksjon brukes for å fordele listen over alle postene på flere sider. Denne viser lenker til de forskjellige sidene, og man kan navigere til dem ved å trykke på disse knappene. I figur 4.2.1.5f ser vi "Side 2" som er den siden vi står på, og man kan navigere tilbake til side 1 eller videre til side 3 ved å trykk på knappen med tilhørende tall. Man kan også navigere fremover ved bruk av ">" for å gå til neste side, eller ">>" for å gå til siste side. Det samme

gjelder for å gå tilbake til forrige side med "<", eller helt tilbake til første side med "<<". I tillegg kan man velge antall poster per side med knappen til høyre for dette (figur 4.2.1.5f).



Figur 4.2.1.5f: Sidelallsfunksjon og nedtrekksmeny for antall poster per side

Innholdet i postene kan være lange, og i dette tilfellet kortes de ned i forumet for å spare skjerm plass. Derimot kan brukere selv bestemme om de vil se mer av posten med en "Les mer"-knapp, og denne kan de trykke på for å se resten av innholdet i posten (figur 4.2.1.5g).

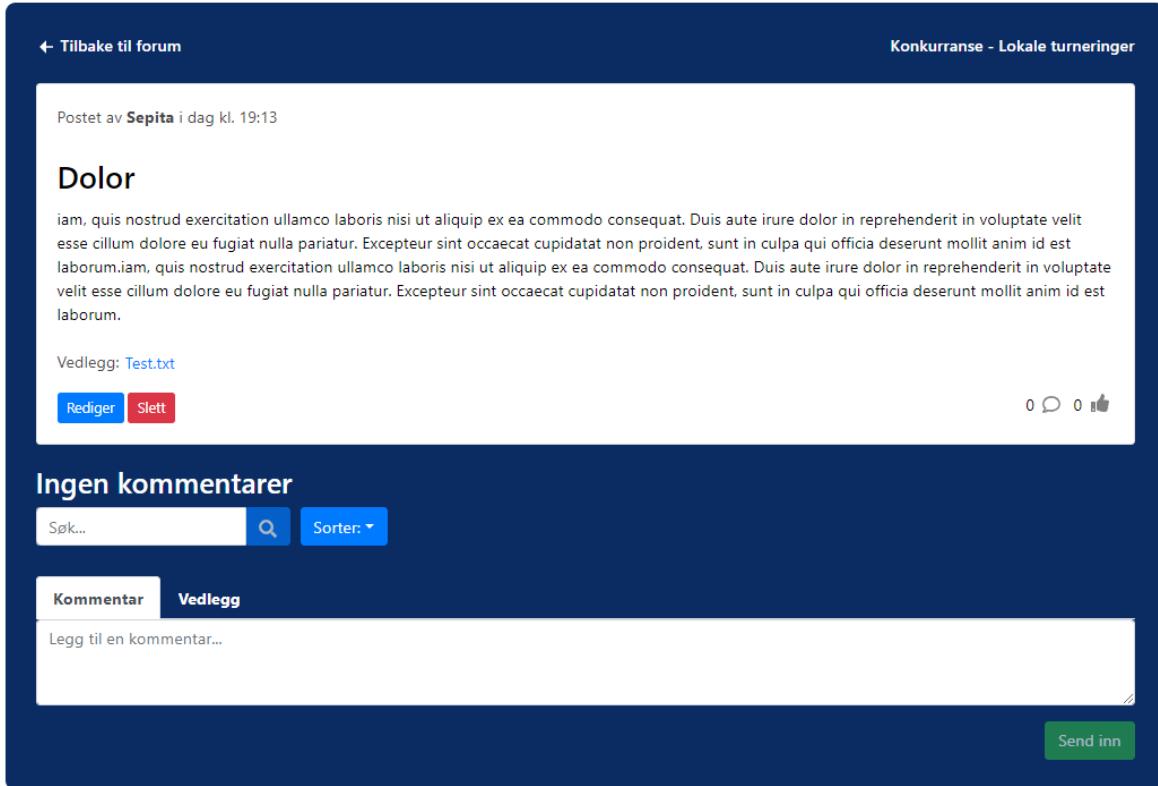


Figur 4.2.1.5g: "Les mer"-knapp i poster

#### 4.2.1.6 Poster og kommentarer

Innholdet i forumet består av forskjellige poster som brukere har sendt inn. Hver post er opprettet med en tittel og innhold, brukernavn til eieren av posten, tid for opprettelse og

eventuelt et vedlegg (figur 4.2.1.6a). Etter at man har oppretter en post med skjemaet på figur 4.2.1.5c, sendes man til en ny side som viser hele posten man opprettet (figur 4.2.1.6a).



The screenshot shows a forum post by user 'Sepita' from 19:13. The post content is 'Dolor' followed by a large block of Latin placeholder text. Below the post are 'Rediger' (Edit) and 'Slett' (Delete) buttons. To the right are interaction counts (0 comments, 0 likes). The interface includes a search bar, a sorting dropdown, and tabs for 'Kommentar' and 'Vedlegg'. A text input field for comments is present, along with a 'Send inn' button.

*Figur 4.2.1.6a: Eksempel på en post i forumet*

Her har brukere mulighet til å redigere eller slette posten sin med bruk av knapper. Hvis man trykker på rediger-knappen, åpnes et skjema som fungerer på samme måte som skjemaet for å opprette en ny post (figur 4.2.1.6b). Etter at posten er redigert, blir den merket med dato og tidspunktet for redigering (figur 4.2.1.6c).

**Rediger post**

Tittel  
Dolor

Innhold

iam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id.

Vedlegg:  
Drag'n'drop, eller klikk for å velge en fil (Max 15 megabytes)

**Avbryt** **Send inn**

Figur 4.2.1.6b: Skjema for å redigere en post

← Tilbake til forum

Postet av **Sepita** i dag kl. 19:13 (Redigert i dag kl. 19:16)

## Dolor

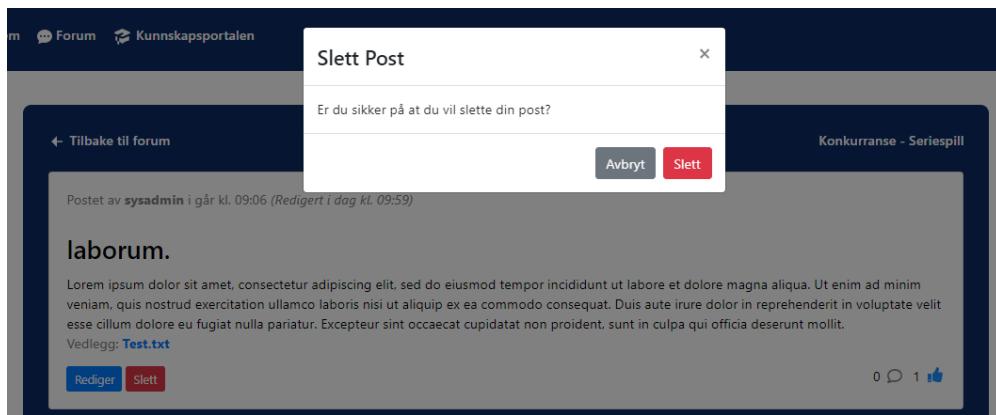
iam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id.

Vedlegg: [Text2.txt](#)

**Rediger** **Slett**

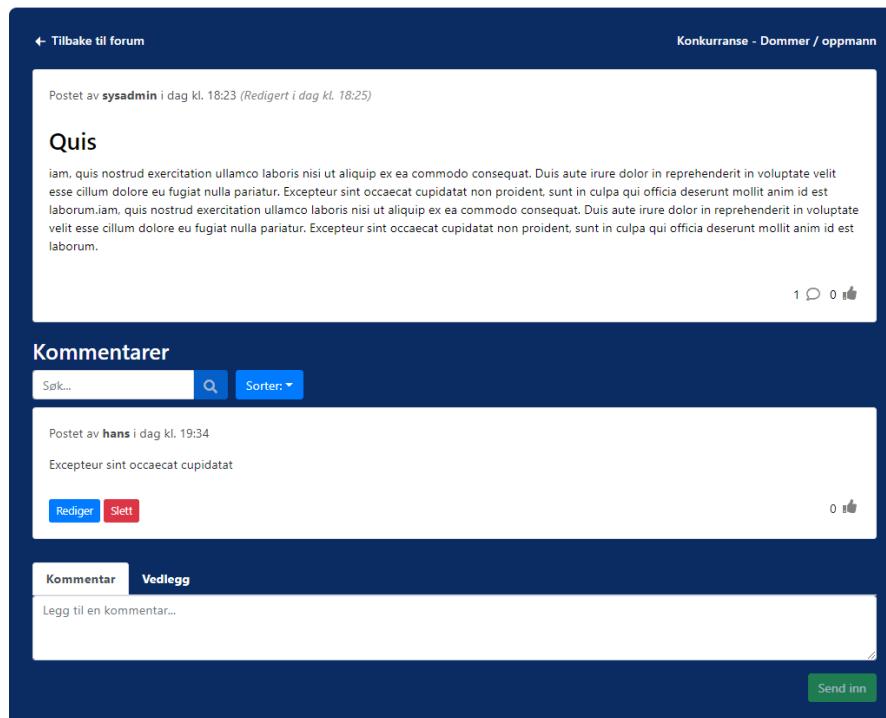
Figur 4.2.1.6c: Dato og tidspunktet for redigering

Sletting av poster er permanent, men for å forhindre at dette gjøres ved en feiltagelse, får brukeren en bekreftelesmelding når det trykkes på slett-knappen. Dette er for å sikre at brukeren faktisk ønsker å slette hele posten (figur 4.2.1.6d).



Figur 4.2.1.6d: Bekreftelsesmelding for sletting av en post

Andre brukere kan “like” poster med “tommel opp”-ikonet og legge igjen kommentarer på disse postene, med eventuelle vedlegg om det ønskes (figur 4.2.1.6e).



Figur 4.2.1.6e: En kommentar under posten

Sletting og redigering av kommentarer fungerer på samme måte som for poster. Hver bruker kan kun slette eller redigere sine egne poster og kommentarer ved bruk av et redigeringsskjema (figur 4.2.1.6f). Redigerte kommentar er merket med dato og tidspunktet for redigering (figur 4.2.1.6g). Hvis brukere ønsker å slette sin kommentar, viser systemet en bekreftelsesmelding for dette (figur 4.2.1.6h). Hvis filtype eller størrelse er ugyldig, vises en feilmelding for dette (figur 4.2.1.6i).

**Rediger kommentar**

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Vedlegg: Test.txt

Drag'n'drop, eller klikk for å velge en fil (Max 15 megabytes)

**Avbryt** **Send inn**

Figur 4.2.1.6f: Skjema for å redigere en kommentar

---

Postet av **Sepita** i dag kl. 11:08 (Redigert i dag kl. 11:08)

Excepteur sint occaecat cupidatat non proident, sunt in cul

**Rediger** **Slett**

---

Figur 4.2.1.6g: Dato og tidspunktet for redigering

**Slett kommentar**

Er du sikker på at du vil slette din kommentar?

**Avbryt** **Slett**

Figur 4.2.1.6h: Bekreftelsesmelding for sletting av kommentar

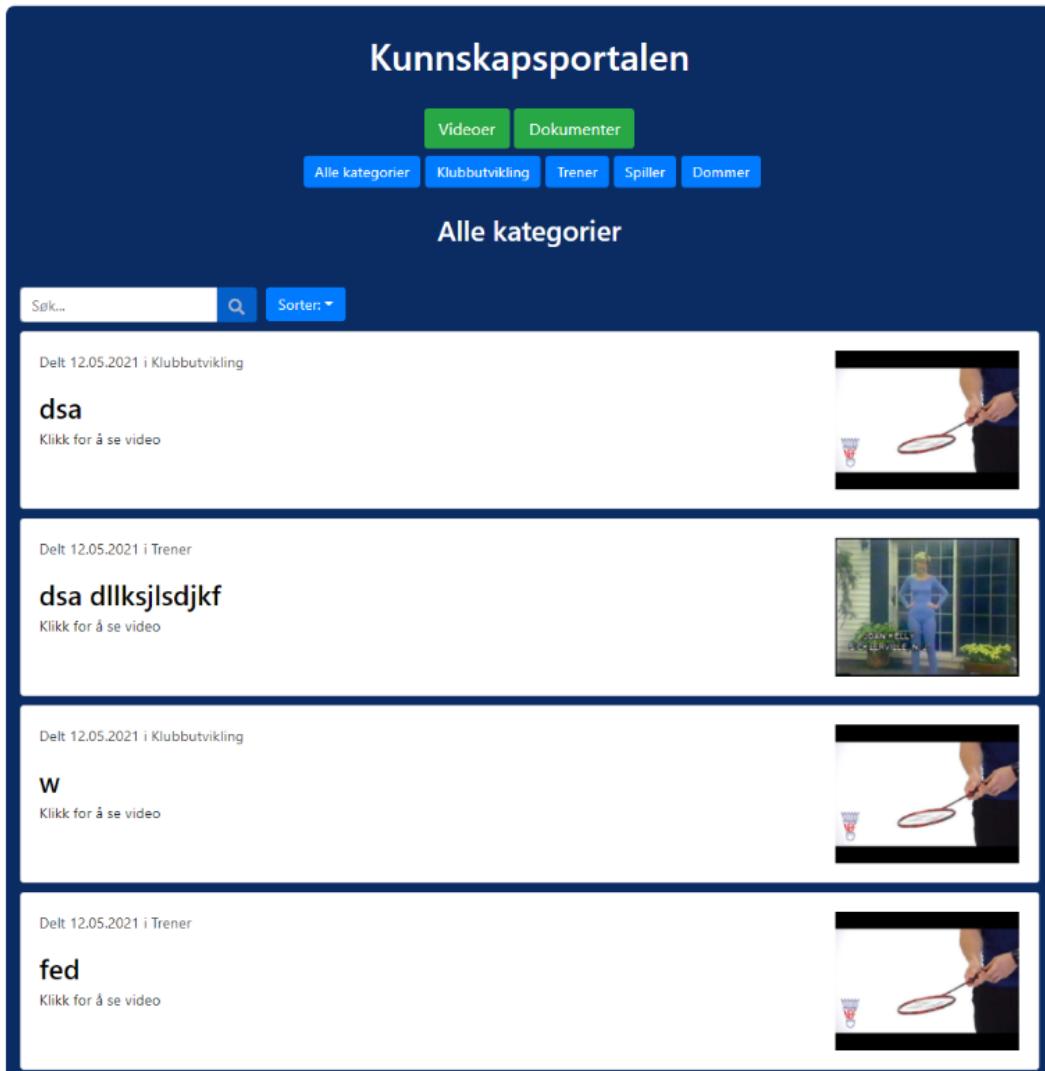
Kommentar
Vedlegg

Filen er for stor eller filtypen er ugyldig ✖

Figur 4.2.1.6i: Feilmelding for feil filstørrelse eller type

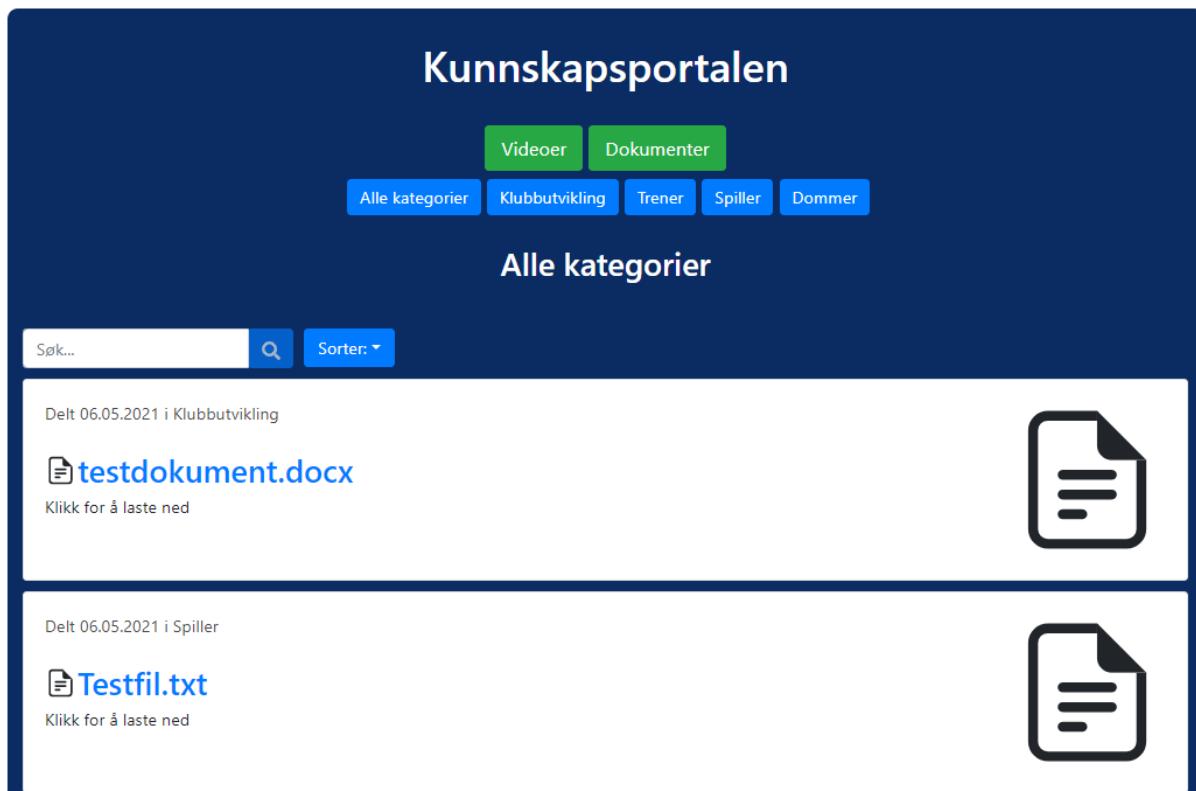
#### 4.2.1.7 Kunnskapsportalen

Kunnskapsportalen er en side der brukere kan se på instruksjonsvideoer og laste ned dokumenter som administrator har lagt ut. Det finnes to knapper: "videoer" og "dokumenter". Under videoer (figur 4.2.1.7a) og dokumenter (figur 4.2.1.7b) finnes underkategoriene for disse. Alle kategoriene til videoer og dokumenter er like.



The screenshot shows the 'Kunnskapsportalen' website interface. At the top, there are two green buttons: 'Videoer' and 'Dokumenter'. Below them is a horizontal menu with five blue buttons: 'Alle kategorier', 'Klubbutvikling', 'Trener', 'Spiller', and 'Dommer'. The main content area is titled 'Alle kategorier'. It features four video thumbnails, each with a title, a 'Klikk for å se video' link, and a small preview image. The first video is titled 'Delt 12.05.2021 i Klubbutvikling' and has the title 'dsa'. The second video is titled 'Delt 12.05.2021 i Trener' and has the title 'dsa dllksjlsdjkf'. The third video is titled 'Delt 12.05.2021 i Klubbutvikling' and has the title 'W'. The fourth video is titled 'Delt 12.05.2021 i Trener' and has the title 'fed'.

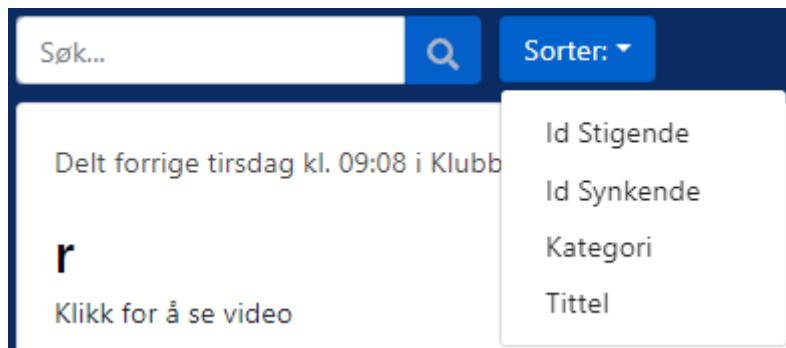
Figur 4.2.1.7a: Kunnskapsportalen med videoer fra alle kategorier



The screenshot shows the 'Alle kategorier' (All categories) page of the Kunnskapsportalen. At the top, there are green buttons for 'Videoer' (Videos) and 'Dokumenter' (Documents). Below them are blue buttons for 'Alle kategorier' (All categories), 'Klubbutvikling' (Club development), 'Trener' (Coach), 'Spiller' (Player), and 'Dommer' (Referee). A search bar with placeholder 'Søk...' and a magnifying glass icon is at the top left. A dropdown menu labeled 'Sorter:' is open, showing options: 'Id Stigende' (Id ascending), 'Id Synkende' (Id descending), 'Kategori' (Category), and 'Tittel' (Title). Two document entries are listed: 'testdokument.docx' (Delt 06.05.2021 i Klubbutvikling) and 'Testfil.txt' (Delt 06.05.2021 i Spiller).

Figur 4.2.1.7b: Kunnskapsportalen med dokumenter fra alle kategorier

På venstre side av kunnskapsportalen finnes en sorteringsknapp som gir muligheten til å sortere videoer og dokumenter på id, kategori eller tittel (figur 4.2.1.7c). Her er også søkefeltet som fungerer på lignende måte som i forumet.



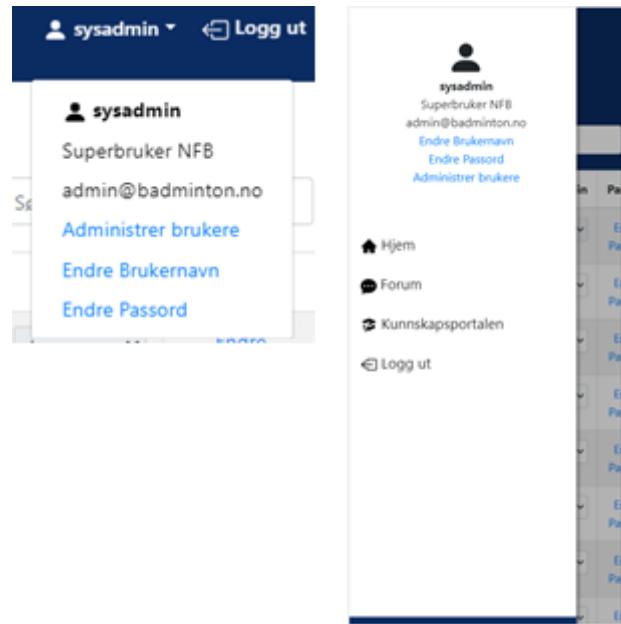
The screenshot shows the search and sorting interface. A search bar with placeholder 'Søk...' and a magnifying glass icon is at the top left. To its right is a dropdown menu labeled 'Sorter:' with the following options: 'Id Stigende', 'Id Synkende', 'Kategori', and 'Tittel'. Below the search bar, a video thumbnail for 'r' (Delt forrige tirsdag kl. 09:08 i Klubb) is shown with the text 'Klikk for å se video' (Click to see video).

Figur 4.2.1.7c: Søkefeltet og sorteringsknappen illustrert i kunnskapsportalen

#### 4.2.1.8 Superbruker

Superbruker eller administrator er ansvarlig for styring av innhold i webapplikasjon. Etter at administratoren er logget på systemet, viser nedtrekksmenyen på navigasjonsbaren brukernavnet til administratoren, og den inneholder informasjon om administrator som fullt navn og e-post, samt at hen kan endre sitt brukernavn og passord, likt som for en vanlig

bruker. Men her har admin også mulighet til å administrere brukere gjennom en egen side, kalt adminpanelet., og kan endre sitt eget brukernavn og passord (figur 4.2.1.8a). Endring av administratorens brukernavn og passord fungerer på samme måte som for vanlige brukere (figur 4.2.1.3c). Sidemenyen som er tilgjengelig for mindre skjermer inneholder alle de samme linkene som elementene i navigasjonsbaren for administrator (figur 4.2.1.8a).



Figur 4.2.1.8a: Nedtrekksmenyen og sidemenyen

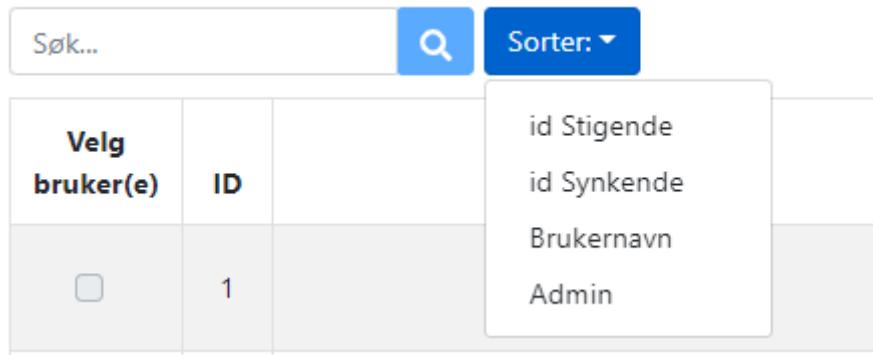
Adminpanelet gir admin mulighet til å administrere brukere og gi dem et høyere tilgangsnivå ved å velge true eller false i "admin" kolonnen, eller fjerne tilgang til webapplikasjonen ved å velge brukere fra "Velg bruker(e)" kolonnen og deretter trykke på "Slett bruker(e)"-knappen (figur 4.2.1.8c).

Søk...		<input type="button" value="Søk"/>	Sorter: ▾	<input type="button" value="Slett bruker(e)"/>	
Velg bruker(e)	ID	Brukernavn	Admin	Passord	
<input type="checkbox"/>	1	sysadmin	true	<a href="#">Endre passord</a>	
<input type="checkbox"/>	3	test	true	<a href="#">Endre passord</a>	
<input checked="" type="checkbox"/>	5	mats	true	<a href="#">Endre passord</a>	
<input type="checkbox"/>	80	tommy	true	<a href="#">Endre passord</a>	
<input type="checkbox"/>	161	.mbn	false	<a href="#">Endre passord</a>	
<input type="checkbox"/>	163	wer	false	<a href="#">Endre passord</a>	
<input type="checkbox"/>	164	gunnar	false	<a href="#">Endre passord</a>	
<input type="checkbox"/>	165	xcvb	false	<a href="#">Endre passord</a>	
<input type="checkbox"/>	169	zxcv	false	<a href="#">Endre passord</a>	
<input type="checkbox"/>	171	hei	false	<a href="#">Endre passord</a>	

Side 1 2 > >>

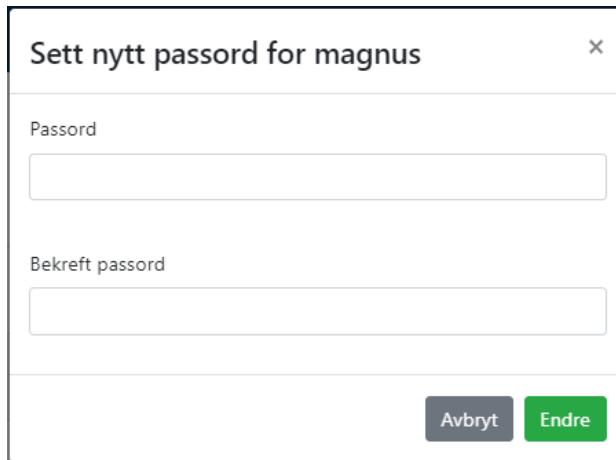
Figur 4.2.1.8c: Admin panel

For å finne enkelte brukere, kan administrator søke i brukerdatabasen ved å benytte søkerfeltet eller sorteringsknappen for dette panelet. Sorteringsknappen kan sortere brukere basert på id, alfabetisk rekkefølge eller tilgangsnivå (figur 4.2.1.8d).



Figur 4.2.1.8d: søkerfelt og sorter knappen i admin panel

Administrator har også muligheten til å endre andre brukeres sitt passord dersom en bruker har glemt passordet og tar kontakt. Dette gjøres ved å trykke på ”Endre passord” i kolonnen for ”Passord” (figur 4.2.1.8c). Her må administrator da fylle inn det nye passordet i skjemaet vi ser i figur 4.2.1.8e.



The form is titled "Sett nytt passord for magnus". It contains two input fields: "Passord" and "Bekreft passord", both represented by empty text input boxes. At the bottom right are two buttons: "Avbryt" (grey) and "Endre" (green).

Figur 4.2.1.8e: Skjemaet for setting av nytt passord for bruker

Gjennom kunnskapsportalen, kan administrator publisere både instruksjonsvideoer og dokumenter ved bruk av ”Last opp video”-knappen (figur 4.2.1.8f) og ”Last opp fil”-knappen (figur 4.2.1.8g).

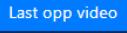
# Kunnskapsportalen

Videoer Dokumenter

Alle kategorier Klubbutvikling Trener Spiller Dommer

## Dommer

Informasjon om Dommer.

Søk...  Sorter: ▾ 

Delt 12.05.2021 i Dommer

**dsa**  
Klikk for å se video 

**adsad**  
Klikk for å se video 

**Slett**

Figur 4.2.1.8f: "Last opp video"-knapp

# Kunnskapsportalen

Videoer Dokumenter

Alle kategorier Klubbutvikling Trener Spiller Dommer

## Trener

Informasjon om Trener.

Søk...  Sorter: ▾ 

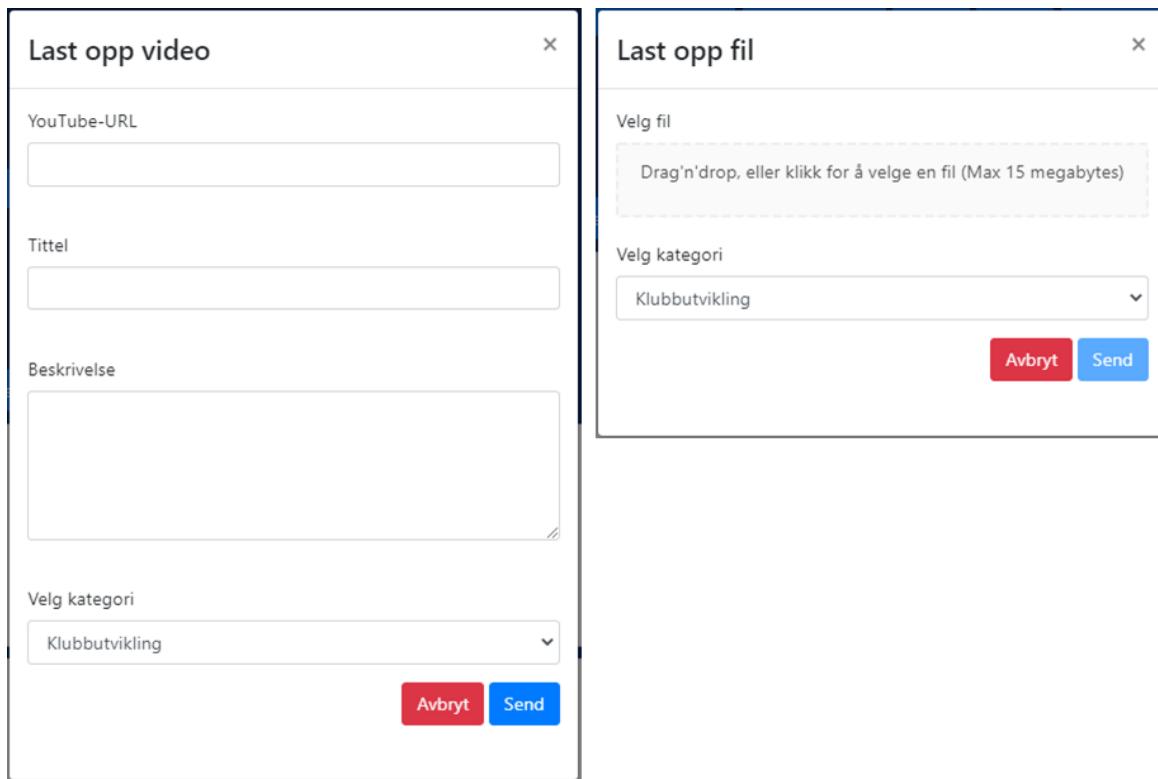
Delt forrige mandag kl. 08:34 i Trener

 **test.txt**  
Klikk for å laste ned 

**Slett**

Figur 4.2.1.8g: "Last opp fil"-knapp

Figur 4.2.1.8h viser to forskjellige skjemaer for å publisere videoer og dokumenter. Administrator må fylle ut alle inputfeltene i videoskjemaet, filstørrelse og filtype i dokumentskjemaet må være gyldig før publisering.



The image shows two separate forms side-by-side, both titled with a close button (X).

**Left Form: Last opp video**

- YouTube-URL: An empty text input field.
- Tittel: An empty text input field.
- Beskrivelse: A large empty text area.
- Velg kategori: A dropdown menu set to "Klubbutvikling".
- Buttons: Red "Avbryt" button and blue "Send" button.

**Right Form: Last opp fil**

- Velg fil: A placeholder text area with instructions: "Drag'n'drop, eller klikk for å velge en fil (Max 15 megabytes)".
- Velg kategori: A dropdown menu set to "Klubbutvikling".
- Buttons: Red "Avbryt" button and blue "Send" button.

Figur 4.2.1.8h: skjemaene for publisere videoer og dokumenter

Hvis noen felt mangler, eller dokumentet ikke er gyldig, viser systemet en relatert feilmelding (figur 4.2.1.8i).

**Last opp video**

---

YouTube-URL

  
Må fylles ut

Tittel

  
Må fylles ut

Beskrivelse

Må fylles ut

Velg kategori

Trener

Avbryt Send

**Last opp fil**

---

Velg fil

Filten er for stor eller filtypen er ugyldig ✖

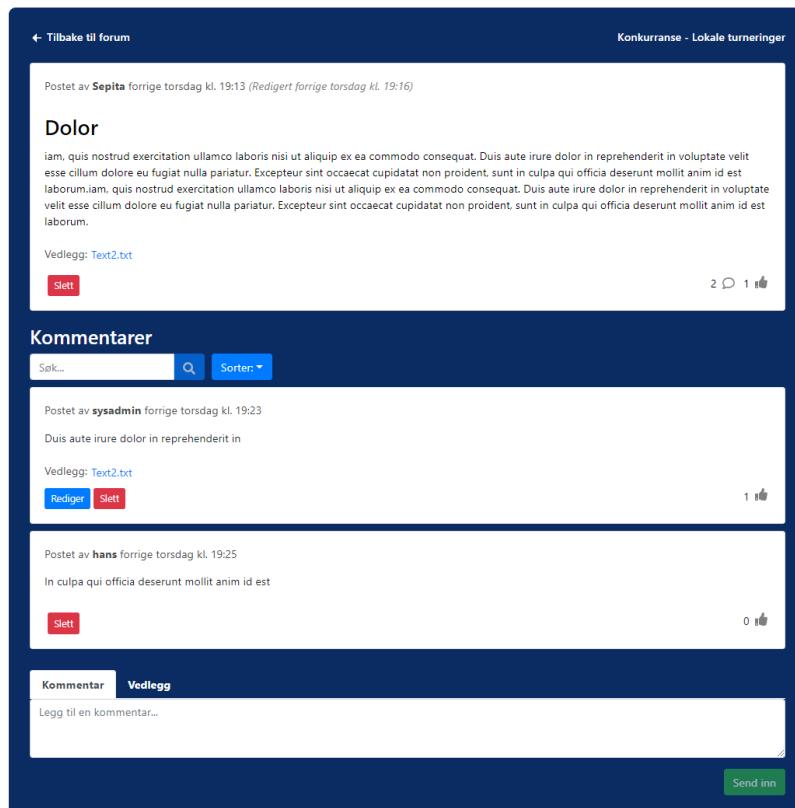
Velg kategori

Klubbutvikling

Avbryt Send

*Figur 4.2.1.8i: Feilmeldingene for å laste opp videoer og dokumenter*

Admin har full kontroll over poster og kommentarer fra andre brukere, og kan slette disse ved behov (figur 4.2.1.8j). Administrator må godkjenne bekreftelsesmeldingen som vises for sletting av hver post eller kommentarer for å forhindre at noe slettes ved en feiltagelse.



Figur 4.2.1.8j: Slett-knapper for å fjerne poster og kommentarer

#### 4.2.1.9 Footer

Footer er tilgjengelig fra alle sidene i webapplikasjonen for å gi brukere lettere tilgang til all nødvendige informasjonen om badmintonforbundet, som e-post, adresse og personvernregler. Ikonene for Facebook, Twitter og Instagram gir direkte adgang til badmintonforbundets sider på sosial media (figur 4.2.1.9a). Disse åpnes i en ny fane.



Figur 4.2.1.9a: Footer

## 4.2.2 Brukeropplevelse

Brukernes opplevelse av webapplikasjonen er avgjørende for om hen ønsker å tilbringe tid på den eller ikke. I denne delen forklarer vi tiltakene vi har gjort for å sikre en god brukeropplevelse.

### 4.2.2.1 Design av brukergrensesnitt

Design av brukergrensesnitt fokuserer på brukerens behov for å sørge for at grensesnittet har elementer som er enkle å få tilgang til, å forstå og å bruke. Konsistens er en nøkkelfaktor i webdesign for både visuelle elementer og funksjonalitet. Derfor var målet å holde konsistens og å sikre at nettstedet ser sammenhengende ut og fungerer harmonisk på tvers av alle forskjellige elementer, som for eksempel knapper, footer, sidebar og navigasjonsfelt.

### 4.2.2.2 Fargepalett

Fargetema til webapplikasjonen er inspirert av NBF sine eksisterende løsninger.

Fargekombinasjonen av blått og hvitt ble brukt for å skape høy kontrast som gjør teksten godt synlig og lesbar. De tre siste fargene er farger som ble brukt i all hovedsak på knapper. Grønn ble brukt for å akseptere eller vise kunnskapsportalen, forum kategorier og logge inn, rød for å slette eller avslå, grå for å avbryte og blå for å sortere, åpne, laste opp og endre. Disse fargene har klare betydninger, så brukere kan skjønne hva hver enkelt knapp signaliserer.

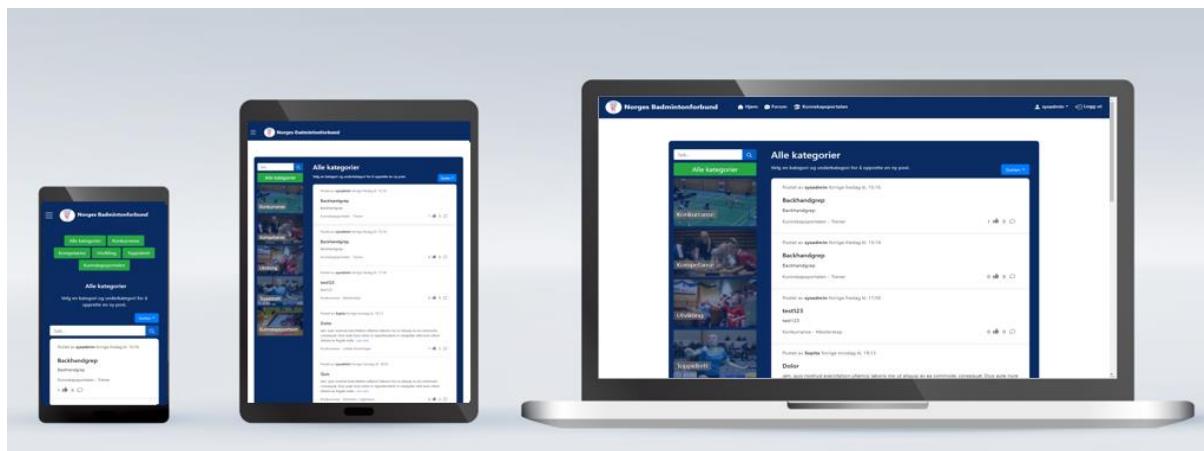


Figur 4.3.2.2a: Oversikt over fargene

### 4.2.2.3 Responsivt webdesign

Webapplikasjonens design er responsivt for å forbedre brukeropplevelsen på tvers av flere plattformer. Dette er hovedsakelig gjort fordi mange bruker smarttelefon og ikke alltid har med seg en datamaskin. Webapplikasjonen har en fleksibel layout som automatisk justeres

etter forskjellige skermstørrelser, som sett i figur 4.3.2.3a. Automatisk justeringer skjer via bruk av @media-query i CSS og Bootstrap. Det er mange skjermer og enheter med forskjellige høyder og bredder, så det er vanskelig å lage et eksakt avbruddspunkt for hver enhet, og en løsning for dette problemet er å bare velge noen avbruddspunkter (w3schools, u.å.). Derfor ble noen avbruddspunkter (breakpoints) lagt til der visse deler av designet ville oppføre seg annerledes på hver side av hvert avbruddspunkt. Et eksempel på et av avbruddspunktene som ble brukt i applikasjonen er "max-width: 1024px".



Figur 4.3.2.3a: Forumsiden på forskjellige skermstørrelser.

Gridsystemet i Bootstrap gjør webapplikasjonen responsiv, og kolonnene omordnes på nytt avhengig av skermstørrelse. På en stor skjerm blir innholdselementene av webapplikasjonen organisert horisontalt i noen kolonner, men på en liten skjerm blir innholdselementene stablet oppå hverandre.

## 4.3 Programmets oppbygging og virkemåte

Etter å ha gått gjennom det sluttbrukeren ser og interagerer med i webapplikasjonen, er det nødvendig å forklare det bakenforliggende og hvordan det er bygget opp.

### 4.3.1 To applikasjoner - én løsning

---

Vår løsning består av to separate deler som til sammen utgjør en fullstendig applikasjon.

#### Teknologistack

Vår teknologistack er splittet i to deler, frontend og backend. I backend har vi ASP.NET (C#) og Azure SQL Database. I frontend har vi utviklet en webapplikasjon med react (javascript). Begge delene er "hostet" på azure sin skyplattform.

#### Asynkronitet

Det er benyttet asynkrone metoder både i backend og frontend. Når en asynkron metode blir kalt i et program, vil programmet fortsette å eksekvere annen kode mens det venter på at metoden skal fullføre. En synkron løsning er nødt til å vente på at hver eneste oppgave blir utført i sekvensiell orden.

I frontend er dette gjort i forbindelse med å hente data fra backend, mens i backend har vi brukt asynkrone metoder når vi gjør operasjoner mot databasen. Det er spesielt viktig i backend, da vi har et max antall tråder til rådighet til enhver tid. Om backend får mange forespørsler på en gang, og overskridet max antall tråder vil forespørsler bli nødt til å vente på at andre skal bli ferdige. En asynkron metode vil frigjøre tråden i påvente av at operasjonen skal bli fullført, og vil da være tilgjengelig for andre oppgaver.

## 4.3.2 Frontend

---

For at webapplikasjonen skal fungere optimalt, trengs en rekke teknologier for å støtte oppunder dette. I denne delen forklares Frontend-delen av webapplikasjonen samt en rekke andre verktøy.

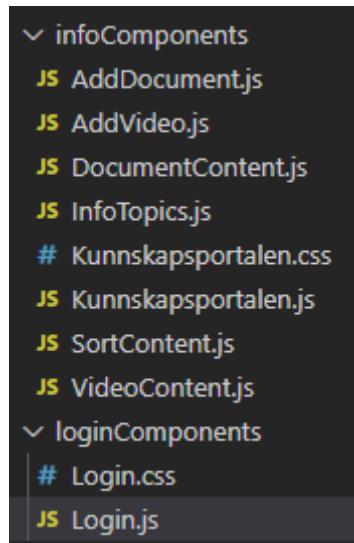
### 4.3.2.1 Webapplikasjonens grunnlag

Webapplikasjonen er en React.js-applikasjon med støtte av JavaScript og CSS, samt en rekke andre biblioteker (libraries) og verktøy. Den består av forskjellige komponenter som er satt sammen til en helhet, se kapittel 4.2. De ulike komponentene blir vist (rendered) i andre komponenter som til slutt blir rendered i App.js. Der blir de ulike komponentene kompilert til JavaScript, og kjørt i en enkel index.js fil som vises som index.html.

React-appen er basert på den offisielle boilerplate React-appen man får ved å kjøre kommandoen “npx create-react-app react-app-name” i et konsollvindu med bruk av Node Package Manager (npm/npx) som følger med i Node.js. Node Package Manager er først og fremst et online repository (oppbevaringssted) for publiseringen av Node.js open source-prosjekter, men også et command-line-verktøy for å interagere med repositoriet som støtter pakkeinstallering, versjonskontroll og avhengighetsstyring (Node.js, 2011).

#### Filstruktur

I React står man fritt frem til å organisere filene i prosjektet. Siden man bygger en React-applikasjon opp med komponenter, har vi valgt å organisere filene nettopp rundt dette. De ulike komponentene (components), representert som Javascript- (eller JSX-) filer, ligger i en mappestruktur etter funksjonalitet og “geografisk” tilhørighet (Startside-komponenter ligger under homeComponents-mappen, Innloggings-komponenter ligger under, loginComponents-mappen osv., se figur 4.3.2.1a). Dette er bevisst gjort for å få en helhetlig og ryddig oversikt over de ulike funksjonalitetene og UI-ene (brukergrensesnittene).



Figur 4.3.2.1a: Illustrasjon av filstruktur i webapplikasjonen

## Fetch API

Fetch API er et grensesnitt for å lage HTTP-forespørsler (requests) som GET, POST, PUT, og DELETE. En HTTP-forespørsel ber en server (tilkoblet en database) om å opprette, hente, slette, eller forandre elementer i databasen. Dette brukes i webapplikasjonen for å kommunisere med backend for å håndtere brukere, poster, kommentarer og det meste annet brukerne har tilgang på. Se eksempel på dette i figur 4.3.2.1b.

```

useEffect(async () => {
  const res = await fetch(host+`GetDocumentInfo/${fileId}`)
  const data = await res.json()
  setFileInfo(data)
}, [])
  
```

Figur 4.3.2.1b: illustrasjon av Fetch API

## Autentisering og autorisasjon i frontend

Bruker er nødt til å logge inn for å kunne anvende applikasjonen. Bruker autentiserer seg med brukernavn/email og passord. Om informasjonen brukeren sender til API-et stemmer overens med databasen, vil brukeren bli logget inn. Klienten mottar også en JWT (JSON Web Token) fra backend som autoriserer brukeren. Så lenge denne tokenen er gyldig (ikke har utgått), vil brukeren forbli innlogget. Når tokenen er utgått (og da blir ugyldig), vil brukeren bli logget ut.

## React router

Webapplikasjonen er en SPA (single page applikasjon) og benytter klient-side rendering. For å unngå komplisert betinget rendering (if/else utsagn) om hvilke komponenter som skal vises til enhver tid, deler vi applikasjonen inn i forskjellige ruter. Dette hjelper React Router oss med. Når en bruker navigerer til f.eks. forumet, vil URL-en utvides med /forum, og komponentene som trengs for å vise denne delen av applikasjonen vil bli renderet. React router lar oss også navigere i appen uten å laste hele applikasjonen på nytt med deres Link-komponent. Mer dokumentasjon om React Router finnes på deres nettsider (<https://reactrouter.com/>).

Som det ble nevnt under autentisering og autorisasjon i frontend, er brukeren nødt til å logge inn for å anvende applikasjonen. For å løse dette har vi definert en HOC (higher order component). En HOC er en komponent som tar inn en annen komponent som argument og returnerer en ny komponent, gjerne med ekstra data eller tilleggsfunksjoner fra den opprinnelige.

```
const ProtectedRoute = ({ children, ...rest }) => {

  const token = localStorage.getItem('token')

  return (
    <Route
      {...rest}
      render={() =>
        token ? (
          children
        ) : (
          <Redirect to={'/Login'} />
        )
      }
    />
  )
}

export default ProtectedRoute
```

Figur 4.3.2.1c: ProtectedRoute.js

ProtectedRoute tar inn et objekt som argument (figur 4.3.2.1c og 4.3.2.1d). Vi plukker ut egendefinerte komponent(er) som children, mens vi bruker spread syntaks for å få med data props til Route komponenten uten å eksplisitt liste dem. Spread syntaks tar en iterabel liste, f.eks et objekt, og sprer objektets felter til individuelle elementer. Om det finnes en token i

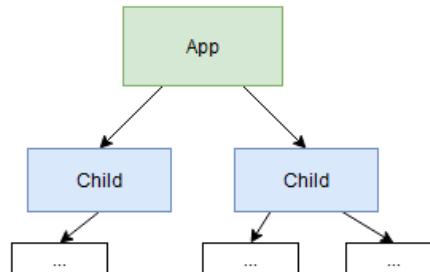
localStorage (lagring av nøkkel-verdi-par i nettleser), vil det bli returnert en Route komponent med vår egen komponent, men om det ikke finnes noen token vil brukeren bli omdirigert til innloggingssiden.

```
<ProtectedRoute exact path="/Forum">
  {initialized ? <Forum
    addPost={addPost}
    subtopics={subtopics}
    topics={topics}
    history={history}
  /> : <SpinnerDiv />}
</ProtectedRoute>
```

Figur 4.3.2.1d: Bruk av ProtectedRoute.js. Her ”wrappes” Forum-komponenten av ProtectedRoute

#### 4.3.2.2 Dataflyt i React

I react flyter data kun en vei, og det er nedover (om vi ser for oss applikasjonen som et datatre, se figur 4.3.2.2a). Mye av utfordringen med å utvikle en React-applikasjon er å håndtere denne dataflyten. Om flere komponenter trenger samme data er vi nødt til å definere dette et sted i treet hvor dataen når disse komponentene, ofte er dette helt i toppen av treet. Dette har vi gjort i App.js.

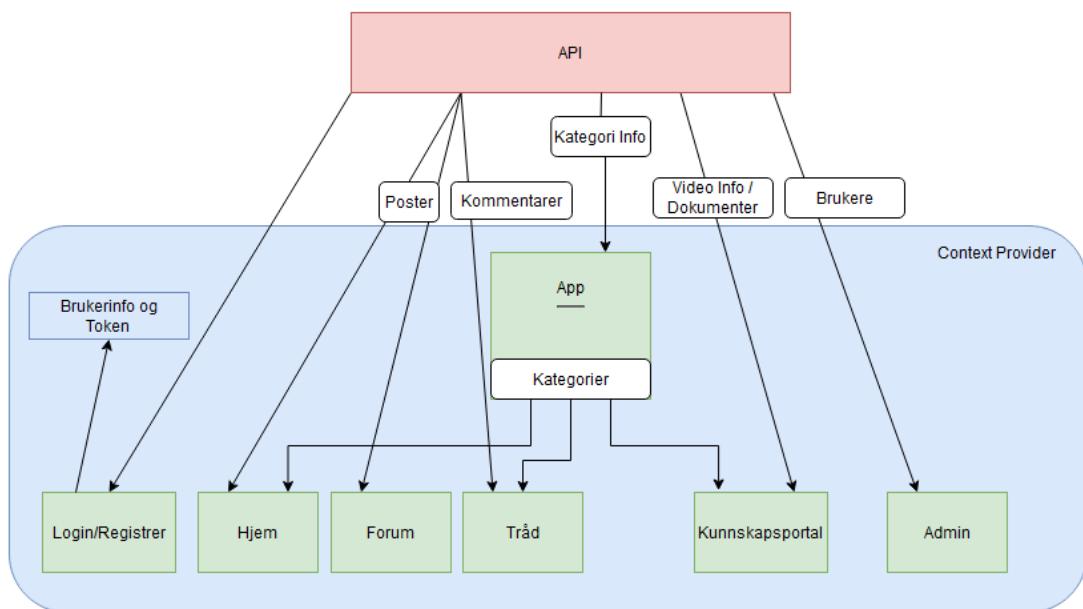


Figur 4.3.2.2a: Illustrasjon av dataflyt i React

I applikasjonen har vi benyttet oss av React Context. React Context gir oss muligheten til å sende data til komponenter uten å måtte sende det nedover i komponent treet manuelt. Vi har implementert React Context for informasjon om innlogget bruker, som da hele applikasjonen har tilgang til.

## Dataflyt i webapplikasjonen

I toppen av applikasjonen henter vi data for kategorier (topics og subtopics til forumet og infotopics til kunnskapsportalen) fra API. Disse distribueres videre ned i trelet til komponentene som skal anvende dem, se figur 4.3.2.2b. Kategori-informasjonen blir brukt til å organisere data som applikasjonen skal vise til brukeren. Informasjon om brukeren som er logget inn er tilgjengelig i hele applikasjonen via Context. Mange av komponentene samhandler med API'et selv istedenfor å få data fra toppen av applikasjons trelet siden vi paginerer dataen i backend. Dette fører til at det er lite tilstandshåndtering i applikasjonen.



Figur 4.3.2.2b: Oversikt over dataflyt og hvor i applikasjonen ulik data blir hentet fra api

Som et eksempel på hvordan vi henter og viser data til klienten kan vi ta en nærmere titt på forumet i applikasjonen. I denne delen av appen vises alle poster, organisert etter kategorier. useEffect er en hook som kjøres etter at komponenten har blitt lastet inn, eller når tilstanden til en av variablene endrer seg i avhengighetsmatrisen (nederst i figur 4.3.2.2c). Så når brukeren trykker på f.eks. en knapp for subtopic eller klikker på neste side, vil denne koden kjøres og oppdatere hvilke poster som brukeren kan se.

```

useEffect(async () => {
  const res = await fetch(host + postsURL, {
    headers: {
      Authorization: `Bearer ${user.token}`
    }
  })
  const posts = await res.json()
  setFilteredPosts(posts.data)
  setTotalPages(posts.totalPages)
  setLoading(false)
  return (() => {
    setFilteredPosts([])
    setTotalPages(null)
  })
}, [subtopicFocus, currentPage, postsPerPage, sort, searchValue])

```

*Figur 4.3.2.2c: Henting av poster i Forum.js*

#### 4.3.2.3 Andre biblioteker

##### Moment

Moment er et JavaScript-bibliotek for å hjelpe til med å tolke og vise dato og tid. Dette har vi for eksempel brukt når vi skal vise når en post ble lagt ut og lignende.

##### React-dropzone

En praktisk pakke som lar oss drag'n'drop filer i nettleseren til opplasting av filer. Man kan enten benytte seg av en ferdig komponent, eller lage en custom component ved hjelp av deres hook. Her har vi valgt å gjøre sistnevnte (se figur 4.3.2.3a). Mer dokumentasjon om React-dropzone kan leses på deres nettsider (<https://react-dropzone.js.org/>).

```

const FileDrop = ({ file, setFile }) => {
  const onDrop = useCallback((acceptedFiles) => setFile(acceptedFiles[0]));
  const {
    acceptedFiles,
    fileRejections,
    getRootProps,
    getInputProps,
  } = useDropzone({
    onDrop,
    maxFiles: 1,
    maxSize: 15728640,
    accept: ".txt, .pdf, .doc, .docx, .xls, .xlsx",
  });
}

```

*Figur 4.3.2.3a: React-dropzone custom component*

## React-Bootstrap

React-Bootstrap er et rammeverk som gjør implementasjon av frontend-komponenter enklere. Den inneholder designmaler for skjemaer, knapper, navigering og andre grensesnitt komponenter og det hjelper med plassering og tilpassing av elementer. Hver komponent er bygget fra bunnen som en ekte React-komponent, uten unødvendige dependencies som jQuery. Mer dokumentasjon om React-Bootstrap finnes på deres nettsider (<https://react-bootstrap.github.io/>).

## React-Icons

React-Icons er et bibliotek som har samlet mange populære ikon-biblioteker inn i en pakke. Dette kan for eksempel ses i webapplikasjonens navigasjonsbar, der vi har brukt ikoner for å formidle ytterligere hva de forskjellige sidene representerer.

## Jwt-decode

Jwt-decode er et bibliotek for å dekode JSON Web Token (jwt). Jwt er en foreslått standard som hjelper med å autentisere brukere og for å overføre informasjon om en bruker på en sikker måte (Bekk Teknologiradar, u.å.). Vi har brukt jwt i webapplikasjonen for å gjøre nettopp dette. Mer dokumentasjon om jwt kan leses på deres nettsider (<https://jwt.io/introduction/>).

## React-hook-form

React-hook-form er et bibliotek som hjelper med å validere skjemaer i React. I mange av input feltene i applikasjonen har vi brukt React-hook-form, sammen med Yup for å oppnå optimal validering.

### Yup

Yup er et bibliotek som lar brukeren enkelt lage valideringsskjema som vi kobler til react-hook-form for input-validering. Med hjelp av Yup kan vi passe på at det sluttbrukerne av webapplikasjonen skriver inn i inputfelt ikke overskridt bestemte grenser, som lengde på brukernavn og lignende, se figur 4.3.2.3b.

```
const schema = yup.object().shape({
  username: yup
    .string()
    .required("Brukernavn må fylles ut")
    .min(5, "Brukernavn må være minst 5 tegn")
    .max(30, "Brukernavn må være maks 30 tegn"),
```

Figur 4.3.2.3b: Illustrasjon av Yup i webapplikasjonen

### 4.3.3 Backend

---

Backend-systemet tar seg av alle funksjonene på serversiden, samt konfigurasjon og håndtering av database. I de kommende kapitlene skal vi se nærmere på strukturen til dette systemet, gå gjennom teknologier og se på kode-eksempler som forklarer hvordan systemet fungerer.

#### 4.3.3.1 RESTful API og Web API

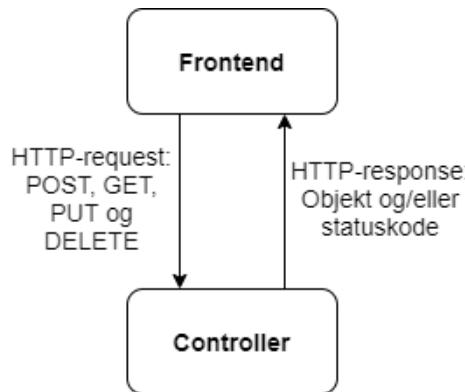
REST står for Representational State Transfer. Det ble utarbeidet av Roy Fielding i 2000, og er en arkitekturstil som er designet på løst koblede klasser (REST API Tutorial, u.å.). Det er flere begrensninger ved denne arkitekturen for å oppfylle kravet om en sann RESTful API-tjeneste. Systemet vi har utviklet er basert på dette konseptet, men oppfyller nødvendigvis ikke alle kravene. Vi har derfor utviklet et Web API som benytter seg av REST sine HTTP-verb GET, POST, PUT og DELETE. Web API-et er basert på Microsoft sitt MVC-rammeverk.

#### 4.3.3.2 MVC-rammeverk

Model-View-Controller (MVC) er et rammeverk som separerer en applikasjon inn i tre logiske komponenter; Model, View og Controller (Guru99, u.å.). Her er det Controller som utgjør koblingen mellom frontend- og backend-systemet. View representerer det som er synlig for sluttbrukere og ville blitt det grafiske brukergrensesnittet. Det er ikke implementert et View i dette systemet, da det kun fungerer som et API for frontend-systemet. Model holder på data, og representerer derfor den dataen som sendes mellom frontend- og backend-systemet via Controlleren.

#### 4.3.3.3 Controllere

En typisk Controller i systemet vi har utviklet, basert på REST API og MVC (som beskrevet i kapittel 4.3.2.1 og 4.3.2.2), tar som tidligere nevnt HTTP-verb for å utføre handlinger. Disse HTTP-verbene tilsvarer det vi kaller CRUD-metoder, som står for Create, Read, Update og Delete. De tilsvarende HTTP-verbene for dette er POST for Create, GET for Read, PUT for Update og DELETE for Delete. I figur 4.3.3.3a ser vi at det sendes en HTTP-request fra frontend til Controller, og vi får tilbake en HTTP-response med det berørte objektet og korrekt statuskode, eller en feilmelding om noe har gått galt.



*Figur 4.3.3.3a: Request og response mellom frontend og backend*

### HTTP metoder

POST brukes for å opprette nye objekter. Her må objektet som skal opprettes sendes med i HTTP-requesten. HTTP-responsen består av objektet som ble opprettet og en "200 OK" statuskode. Hvis objektet som skal opprettes ikke blir sendt med i HTTP-requesten, returneres en "400 Bad Request" statuskode.

GET brukes for å hente data. Disse metodene kan brukes både med og uten en ID-variabel. Uten ID-variabelen får vi tilbake en liste over alle objekter, og med ID vil vi få tilbake det objektet som tilsvarer ID-en til variabelen. HTTP-responsen består derfor av en liste over alle objektene eller kun det etterspurte objektet sammen med en "200 OK" statuskode. En tom liste vil fortsatt gi en "200 OK" statuskode, men om det søkes etter et objekt som ikke eksisterer, vil det returneres en "404 Not Found" statuskode.

PUT brukes til å oppdatere objekter. Dette fungerer nesten på samme måte som POST, men her må det legges ved en ID-variabel for å identifisere objektet som skal oppdateres. HTTP-responsen vil da bestå av det oppdaterte objektet med en "200 OK" statuskode. Hvis det legges ved en ID-variabel som ikke eksisterer i databasen, returneres det en "404 Not Found" statuskode. Hvis ID-variabelen i objektet som skal oppdateres ikke stemmer overens med objektet i databasen, returneres det en "400 Bad Request" statuskode.

DELETE brukes for å slette objekter. Her sendes det en HTTP-request med en ID-variabel til objektet vi ønsker å slette. HTTP-responsen består av det objektet som ble slettet og en "200 OK" statuskode. Hvis ID-variabelen som ble sendt med ikke eksisterer i databasen, returneres det en "404 Not Found" statuskode.

Alle metodene har også en try-catch blokk som returnerer en "500 Internal Server Error" statuskode om noe skulle gå galt i backend-systemet

## Typiske Controllere

De typiske Controllerene som benyttes slik vi har beskrevet over er som følge:

- CommentsController. Metoder for kommentarer i forumet
- InfoTopicsController. Metoder for temaer til kunnskapsportalen
- PostsController. Metoder for poster i forumet
- SubTopicsController. Metoder for undertemaer i forumet
- TopicsController. Metoder for temaer i forumet
- UsersController. Metoder for brukere av systemet
- VideosController. Metoder for videoer i kunnskapsportalen

## Ikke-typiske Controllere

Controllere som faller litt bort fra standardiseringen vi ellers har brukt, er Controllere med litt mere spesifikke oppgaver. Disse er som følge:

- CustomController. Diverse tilpassede metoder
- LikesController. Metoder for "likes" av poster og kommentarer

CustomController inneholder alle metodene som brukes for å laste opp, liste, slette og laste ned dokumenter (les mer om filhåndtering i kapittel 4.3.3.10). Her finner vi også metoden for å logge inn og autentisere/autorisere brukere. I tillegg er det lagt til metoder for å endre admin-brukere og endre brukernavn uten passord. Alle disse metodene er tilpasset bruk i frontend-systemet, og kan derfor ikke sees på som vanlige CRUD-metoder.

LikesController er også tilpasset frontend-systemet. Her har vi 3 metoder. Den første er GetLike for å sjekke statusen til en post/kommentar med gitt bruker. Vi har AddLike for å legge til en «like» på en post/kommentar med gitt bruker, og til slutt DeleteLike for å fjerne en «like» fra en post/kommentar med gitt bruker.

## Utvidelser

Etter hvert som systemet vokste, ble noen Controllere utvidet med paginerte lister, sortering og søkemetoder. Dette gjelder Controllere for poster, kommentarer, brukere, videoer og dokumenter. Disse metodene ligner mye på hverandre, og beskrives nærmere i kapittel 4.3.3.8. Det er lagt ved en fullstendig liste over alle funksjoner i kapittel 4.3.3.12.

#### 4.3.3.4 Oppstart av applikasjonen

Det er to filer som er sentrale for oppstart av applikasjonen, dette er Startup.cs og Program.cs. Når en lager et nytt ASP.NET Core prosjekt i Visual Studio, blir begge disse filene automatisk opprettet. Alt vi trenger å gjøre selv, er å konfigurere de.

I Startup.cs-klassen konfigureres servicer. Disse servicene er gjerne gjenbruksbare komponenter med funksjonalitet som brukes på kryss av hele applikasjonen via dependency injection (Anderson et al., 2019). Eksempler på slike servicer er database-, AzureStorage- og JWT Token-konfigurering. Alle klassene til applikasjonen som er lagdelt er også definert her med en levetid (les mer i kapittel 4.3.3.5).

Program.cs er den første filen som kjøres, og denne klassen er et konsollprogram tilsvarende en «public static void Main()» metode (TutorialsTeacher, u.å.). I en ASP.NET Core applikasjon kaller vi på alle hosting- og startup.cs-metoder fra denne Main() metoden (WebTrainingRoom, u.å.).

#### 4.3.3.5 Dependency Injection

Dette er en teknikk for å oppnå Inversion of Control (IoC) mellom klasser som er avhengig av hverandre (Microsoft, 2020). Med dette får vi en abstrakt implementasjon og løst koblede klasser (TutorialsTeacher, u.å.).

Når vi skal lagdele applikasjonen, deles alle klasser opp slik vi har forklart i kapittel 4.3.3.7. Vi har implementert interfaces til de forskjellige klassene med metodene de inneholder. Disse interfacene og klassene med metoder registreres i Startup.cs filen med en levetid. Det er tre forskjellige levetider vi må forholde oss til. Dette er «Singleton» som kun opprettes én gang for applikasjonens levetid, «Transient» som alltid opprettes på nytt hver gang det gjøres en spørring til det og «Scoped» som opprettes en gang for hvert kall på dette objektet og bruker det på nytt innenfor den samme spørringen (TekTutorialsHub, u.å.). Ut fra dette kan vi si at et «Transient»-objekt aldri er det samme, og instansieres på nytt hver gang vi bruker det i koden. Et «Scoped»-objekt er likt innenfor den gitte spørringen, og instansieres derfor kun én gang per HTTP-spørring til applikasjonen. Et «Singleton»-objekt er alltid helt likt, og vil derfor kun instansieres en gang så lenge applikasjonen kjører.

En service med lavere levetid som blir injisert inn i en service med høyere levetid, vil endre levetiden for servicen med lavere levetid til høyere levetid, og dette vil gjøre det vanskelig å feilsøke i applikasjonen (TekTutorialsHub, u.å.). Når vi lagdeler applikasjonen har de forskjellige interfacene en avhengighet til hverandre, og når vi definerer levetiden til disse må vi derfor ta hensyn til det avhengige interfacet sin levetid. På figur 4.3.3.5a ser vi en skisse som viser avhengigheten mellom lagene.



Figur 4.3.3.5a: Avhengighet mellom de forskjellige lagene

Data Access Layer-klasser injiseres derfor inn i Business Layer-klasser som igjen injiseres inn i Controller-klasser. Data Access Layer-klassene har derfor den lengste levetiden og er registrert som «Scoped»-objekter. Business Layer-klassene har en lavere levetid og er registrert som «Transient»-objekter. Controller-klassene er kun avhengig av Business Layer, og injiseres ikke inn noen andre steder, og er derfor heller ikke registrert med en levetid.

### 4.3.3.6 Databasen

Som en del av Microsoft sin .NET-plattform vi har benyttet i backend systemet (ASP.NET Core), har vi tatt i bruk NuGet-pakken til Entity Framework for å opprette og konfigurere databasen. Entity Framework støtter tre forskjellige fremgangsmåter som heter Code First, Database First og Model First, hvor vi har tatt i bruk den første, også kalt Code First Approach (Tutorialspoint, u.å.).

#### Code First

Når vi skal lage en ny database med Code First, oppretter vi først en klasse med modellen for den entiteten vi skal lage. I denne klassen lager vi deretter metoder for alle attributtene til entiteten. Disse metodene opprettes etter hva slags type attributtet skal være, f.eks. er ID et tall (representeres her som int) og tittel består av tekst (representeres her som string). Ved hjelp av Data Annotations definerer vi så hvilke egenskaper attributtene skal ha, slik som primærnøkkel og fremmednøkkel. Data Annotations brukes av ASP.NET for å konfigurere klassemodeller (Tutorialspoint, u.å.). Entity Framework leser disse annoteringene når databasen genereres, og oppretter attributter med de definerte egenskapene. Et eksempel på dette kan vi se på figur 4.3.3.6a hvor «Id» blir primærnøkkel siden vi har definert denne

med en [Key] annotering og «UserId» blir fremmednøkkel siden den er definert med en [ForeignKey] annotering. Alle metodene må opprettes som «public» med «get» og «set»-funksjoner slik at de blir tilgjengelige for andre klasser i systemet.

```

public class SubTopic
{
    // Database for undertemaer
    [Key]
    7 references
    public int Id { get; set; }

    19 references
    public string Title { get; set; }

    19 references
    public string Description { get; set; }

    // Relasjoner
    [ForeignKey("Topic")]
    21 references
    public int TopicId { get; set; }

    0 references
    public Topic Topic { get; set; }

    0 references
    public ICollection<Post> Posts { get; set; }
}

```

Figur 4.3.3.6a: Her ser vi et eksempel på modellen for et underemne (SubTopic).

## Fluent API

I tillegg til Data Annotations, kan vi også bruke Fluent API for å konfigurere klassemfellene. Begge disse konfigurasjonene kan brukes samtidig, og i de tilfellene, vil konfigurasjoner gjort med Fluent API komme over de som er gjort med Data Annotations. Med Fluent API kan vi også gjøre mer avanserte konfigurasjoner (Tutorialspoint, u.å.). På figur 4.3.3.6d ser vi et eksempel på dette.

```

// En bruker kan ha mange poster
modelBuilder.Entity<User>()
    .HasMany<Post>(u => u.Posts)
    .WithOne(p => p.User)
    .HasForeignKey(p => p.UserId)
    .OnDelete(DeleteBehavior.SetNull); // Sett poster sin userId til null om bruker slettes

```

Figur 4.3.3.6d: Konfigurasjon med Fluent API

Her har vi definert at feltet for fremmednøkkelen til bruker i alle poster hvor den eksisterer skal settes til en nullverdi om den tilhørende brukeren slettes. Vi har tilsvarende konfigurasjoner for kommentarer, dokumenter og videoer. I frontend-systemet har vi definert at det skal vises [Slettet bruker] når disse feltene har en nullverdi. På den måten kan vi

beholde innlegg fra slettede brukere i systemet, og samtidig informere andre sluttbrukere at den aktuelle brukeren ikke lenger eksisterer.

## Generere database

Etter at vi har laget modeller for entitetene og skrevet koder som kan bruke disse, vil Entity Framework generere databasen med kodene vi har skrevet når vi kompilerer og kjører programmet. Dette lar oss skrive klasser for alle modellene fremfor å utvikle en hel database i forkant (Entity Framework Tutorial, u.å.). Denne fremgangsmåten har passet godt til dette prosjektet siden vi har modellert databasen selv på en måte hvor vi begynte med minimalt av funksjoner og utvidet databasen etter hvert som vi fikk behov for det. Vi kunne derfor starte utviklingen av selve systemet med en gang.

Det eksisterer også metoder for å migrere en database i Entity Framework, men siden vi har vært de eneste brukerne av systemet under utviklingen har vi ikke hatt behov for dette. Vi valgte isteden en tilnærming hvor vi opprettet databasen på nytt hver gang vi gjorde endringer.

## Lokal og online database

Helt i begynnelsen av utviklingen kjørte vi systemet lokalt på maskinen vi brukte og opprettet en lokal database i form av en flatfil (.mdf). Dette tillot oss å generere nye databaser på få sekunder når vi utvidet databasen med nye attributter og flere entiteter. Etter hvert som vi fikk et mer komplett system og inkluderte oppdragsgiver i prosessen, ble applikasjonen publisert på Azure.

For å enkelt bytte mellom Azure og en lokal database, laget vi et lite kommentarfelt i filen appsettings.json som inneholdt de strenge vi brukte for begge disse metodene. Disse strenge kopierte og limte vi inn i feltet for «AzureDatabase» etter behov. Figur 4.3.3.6b viser et utklipt av dette oppsettet.

---

```
// Disse kommentarene må fjernes når vi er ferdig, skal kun brukes av oss for å limes inn i connection string under her
// Lokal database: Server=(localdb)\mssqllocaldb;Database=webforum;Trusted_Connection=True;ConnectRetryCount=0
// Azure database: Data Source=tcp:webforum.database.windows.net,1433;Initial Catalog=webforum;User Id=forumadmin@webforum;Password=la8PWkaller!
{
    "ConnectionStrings": {
        "AzureDatabase": "Server=(localdb)\mssqllocaldb;Database=webforum;Trusted_Connection=True;ConnectRetryCount=0",
    }
}
```

Figur 4.3.3.6b: Innstillingar for databasen i appsettings.json.

På denne måten kunne vi enkelt ha en database for produksjon på Azure, mens vi kunne teste nye metoder under utviklingen lokalt på maskinen vi brukte. Når oppdragsgiver overtar dette systemet, er det her de skal konfigurere sin egen database.

## Databaseinitialisering

For å slippe og legge inn data i diverse felter i databasen hver gang den ble opprettet på nytt, ble løsningen å bruke forhåndsgenererte «seeds» som fylte ut disse feltene for oss. Dette sparte oss for mye arbeid siden vi ikke lengre var avhengig av å gjøre dette manuelt hver gang. For å lage slike «seeds» i ASP.NET, opprettet vi en egen klasse som tok seg av dette. I klassen Program.cs har vi definert at databasen skal opprettes og klassen DBInit.cs skal brukes til initialisering av denne. På figur 4.3.3.6c kan vi se et eksempel på initialisering av temaer.

```
// Opprette nye temaer (TOPICS)
var topics = new Topic[]
{
    new Topic{Title="Konkurranse",Description="Informasjon om Konkurranse. Filles ut senere!",ImageUrl="images/kategori.konkurranse.jpg"},
    new Topic{Title="Kompetanse",Description="Informasjon om Kompetanse. Filles ut senere!",ImageUrl="images/kategori.kompetanse.jpg"},
    new Topic{Title="Utvikling",Description="Informasjon om Utvikling. Filles ut senere!",ImageUrl="images/kategori.utvikling.jpg"},
    new Topic{Title="Toppidrett",Description="Informasjon om Toppidrett. Filles ut senere!",ImageUrl="images/kategori.toppidrett.jpg"}
};
foreach (Topic topic in topics)
{
    context.Topics.Add(topic);
    context.SaveChanges();
}
```

*Figur 4.3.3.6c: Et eksempel på «seeds» for emner.*

Her blir hvert tema lagt inn i et objektarray, og en foreach-løkke leser så gjennom dette og legger til hvert objekt i databasen. Som vi kan se blir endringene lagret for hvert objekt som legges til. Dette er for å gi disse objektene et ID-nummer i den rekkefølgen vi har lagt dem til slik av vi kan bruke denne ID-en når vi skal knytte et undertema til et temaet. Figur 4.3.3.6d viser et eksempel på et undertema som har en tilknytning til et tema.

```
// Opprette nye undertemaer (SUBTOPICS)
var subtopics = new SubTopic[]
{
    new SubTopic{Title="Dommer / oppmann",Description="Informasjon om Dommer / oppmann. Filles ut senere!",TopicId=1},
}
```

*Figur 4.3.3.6d: Viser et underemne som har tilknytning til et emne.*

Legg merke til feltet «TopicId=1». Siden temaet «Konkurranse» var det øverste objektet på figur 4.3.3.6c, vil dette temaet derfor få ID-nummer 1, og da vil derfor undertemaet «Dommer / oppmann» tilhøre dette temaet.

Foruten temaer og undertemaer, legges det også til en bruker under denne initialiseringen. Dette er en administratorbruker som er tiltenkt IT-ansvarlig hos oppdragsgiver. Med denne brukeren vil det være mulig å tildele administrator-rollen til andre brukere etter behov. Denne brukeren er opprettet med følgende:

Brukernavn: sysadmin

Passord: password

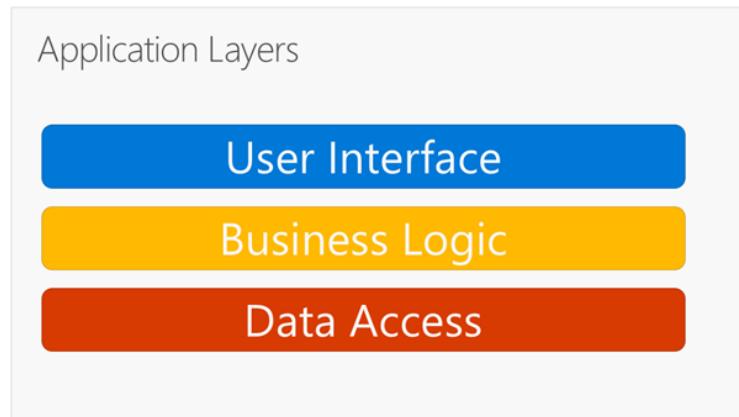
Dette kan endres her i DBInit-klassen før systemet publiseres, eller i webapplikasjonen (som vist i kapittel 4.2.1.3) etter publisering.

## **DbContext**

En instans av DbContext klassen representerer en sesjon med databasen, og brukes til å konfigurere, gjøre spørninger og lagre data til databasen (Entity Framework Tutorial, u.å.). Alle entitetene i databasen registreres i denne klassen med «DbSet<Entitet>» og vil være tilgjengelig med dependency injection i de klassene hvor vi setter opp dette. Som beskrevet i kapittel 4.3.3.7 om lagdeling, ser vi at aksessering av databasen skjer i Data Access Layer-klassene, og det er her dette brukes. Konfigurasjoner gjort med Fluent API skal også ligge i DbContext klassen.

### **4.3.3.7 Lagdeling**

Lagdeling, også kalt N-Layer architecture (Altvater, 2017), er en vanlig måte å dele inn større applikasjoner i flere lag for å separere User Interface (heretter UI) fra databasen slik at UI ikke har en direkte innvirkning på hvordan data håndteres. Den vanligste måten å gjøre dette på er å dele applikasjonen inn i tre forskjellige lag, UI, Business Logic Layer (heretter BLL) og Data Access Layer (heretter DAL) (Microsoft, 2020). På figur 4.3.3.7a kan vi se en illustrasjon av hvordan disse lagene representeres ovenfra og ned.



*Figur 4.3.3.7a: Typical application layers., 2020, <https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures>*

UI er det øverste laget og er det som er synlig fra utsiden. BLL ligger mellom UI og DAL og fungerer som et mellomlag mellom disse. DAL ligger nederst og det er her koden som aksesserer databasen skal ligge. UI er selve hovedprosjektet til applikasjonen. Under dette er BLL og DAL opprettet som egne klassebiblioteker.

### User Interface

Det øverste laget som representeres av UI er det første laget i applikasjonen hvor metodene som er tilgjengelig for brukere av systemet eksisterer. Disse metodene skal ikke aksessere databasen direkte. I backend-systemet vi har utviklet, er dette laget representert som API-et. Alle modellene (som er tabellene fra databasen) har sin egen Controller, og i disse Controllerene finner vi de tilhørende metodene. Ser vi nærmere på en av disse Controllerene, kan vi se at metoden i eksemplet på figur 4.3.3.7b kaller på en metode i BLL for å utføre ønsket handling.

```
// GET: Users/1
[HttpGet("{id:int}")]
4 references | 3/3 passing
public async Task<ActionResult<UserDTO>> GetUser(int id)
{
    try
    {
        var user = await _userBLL.GetUser(id);
        if (user != null)
        {
            return Ok(user);
        }
        else
        {
            return NotFound($"Bruker med ID {id} ble ikke funnet");
        }
    }
    catch (Exception)
    {
        return StatusCode(StatusCodes.Status500InternalServerError, "Feil ved henting av bruker");
    }
}
```

*Figur 4.3.3.7b: Viser en metode for å hente brukerinformasjon i UsersController.cs*

Her ønsker vi altså å hente informasjon om en bruker. Vi har en metode som tar ID til ønsket bruker, gjør et kall til tilsvarende metode i interfacet til BLL med dependency injection og sender den gitte ID-en med videre i dette kallet. BLL vil da returnere ønsket bruker-objekt som et Data Transfer Object (heretter DTO) med en «200 OK» HTTP-responskode, eller en feilmelding med «404 Not Found»-kode hvis vi får tilbake en nullverdi. En try-catch-blokk er lagt inn i denne metoden for å sikre at applikasjonen ikke slutter å fungere om det skulle oppstått en feil i dette kallet.

### **Business Logic Layer**

Det neste laget i applikasjonen er BLL. Vi fortsetter med eksempelet fra UI over. På figur 4.3.3.7c kan vi se koden til denne metoden. Her får vi med ID-en som ble sendt fra UI. I BLL gjøres det et nytt kall til tilsvarende metode i interfacet til DAL med dependency injection. Den samme ID-en sendes videre til metoden i DAL, og vi får enten tilbake riktig bruker-objekt eller en nullverdi hvis det ikke ble funnet. Gitt at dette bruker-objektet eksisterer, sendes det tilbake til UI som et DTO.

```
5 references | 3/3 passing
public async Task<UserDTO> GetUser(int id)
{
    var getUser = await _repository.GetUser(id);
    if (getUser != null)
    {
        return new UserDTO(getUser);
    }
    else
    {
        return null;
    }
}
```

Figur 4.3.3.7c: Viser koden for å hente en bruker i BLL

Når vi skal sende bruker-objektet tilbake til UI, er det naturlig å sende det tilbake som et DTO siden bruker-modellen kan inneholde felter vi ikke ønsker å vise til sluttbruker (Kanjilal, 2020). Dette kan for eksempel være passord og annen «hemmelig» informasjon. Konverteringen fra bruker-objekt til DTO skjer her i BLL, og kan sees på denne kodelinjen hvor vi sender objektet tilbake «return new UserDTO(getUser)». Figur 4.3.3.7d viser hvordan UserDTO-modellen ser ut.

```
72 references
public class UserDTO
{
    8 references | 6/6 passing
    public int Id { get; set; }
    3 references | 1/1 passing
    public string Username { get; set; }
    1 reference
    public string FirstName { get; set; }
    1 reference
    public string LastName { get; set; }
    1 reference
    public string Email { get; set; }
    1 reference
    public bool Admin { get; set; }

    10 references
    public UserDTO(User user)
    {
        Id = user.Id;
        Username = user.Username;
        FirstName = user.FirstName;
        LastName = user.LastName;
        Email = user.Email;
        Admin = user.Admin;
    }
}
```

Figur 4.3.3.7d: Her kan vi se modellen for UserDTO med tilhørende konstruktør.

Her kan vi se at UserDTO har alle feltene vi ønsker å gjøre tilgjengelig. Vi har også en konstruktør som tar inn et vanlig bruker-objekt og setter de forskjellige verdiene i DTO-et til ønsket verdi fra bruker-objektet vi tok inn.

## Data Access Layer

Helt nederst har vi DAL som er det siste laget i lagdelingen. Det er her koden som aksesserer databasen ligger. Som vi har sett til nå er det altså ingen direkte kontakt mellom UI og DAL. Vi fortsetter med eksempelet hvor vi skal finne en bruker. Figur 4.3.3.7e viser koden til denne metoden fra DAL.

```
2 references
public async Task<User> GetUser(int id)
{
    var user = await _context.Users.FindAsync(id);
    if (user != null)
    {
        return user;
    }
    else
    {
        return null;
    }
}
```

*Figur 4.3.3.7e: Dette er koden for å hente en bruker i DAL*

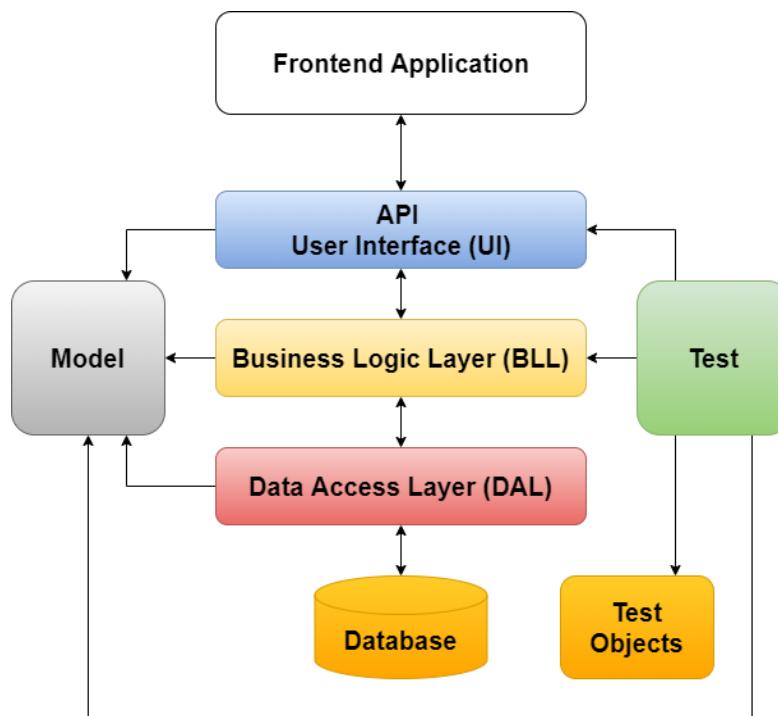
Her ser vi at metoden «GetUser(int id)» tar ID-en til en bruker vi har fått fra BLL. På denne kodelinjen «var user = await \_context.Users.FindAsync(id)», ser vi at user instansieres som et objekt og vi finner brukeren via modellen til «Users» med metoden FindAsync(id). Det som skjer, er at vi aksesserer Users i databasen med dependency injection fra DbContext klassen og bruker en asynkron metode (les om asynkronitet i kapittel 4.3.1) til å hente ut objektet vi leter etter. Finnes ikke dette objektet, returneres en nullverdi.

## Sammendrag

Vi har nå sett hvordan et metodekall starter i UI, sendes videre til BLL og til slutt ender opp i DAL hvor databasen aksesseres. Deretter sendes enten objektet vi har forespurt tilbake til BLL, ellers returneres en nullverdi. I BLL får vi tilbake objektet fra DAL og konverterer dette til et DTO og sender det tilbake til UI. UI gir så sluttbruker korrekt HTTP-responskode sammen med objektet som ble forespurt, eller en feilmelding om det ikke eksisterer. På denne måten har vi separert UI fra databasen og filtrert objektet gjennom BLL for å gjemme felter vi ikke vil vise til sluttbruker.

## Andre klassebiblioteker

Vi har også noen klassebibliotek som ikke er med i denne lagdelingen. Dette er Test og Model. Test er et eget prosjekt for enhetstesting og Model er alle modellene til databasen og DTO-ene. Figur 4.3.3.7f viser hvordan backend-systemet er strukturert i systemet vi har utviklet.



Figur 4.3.3.7f: Backend applikasjonsstruktur

Lagdelingen er som forklart tidligere, og her er det kun frontend-systemet som benytter seg av API-et i UI-laget. Enhetstester tar i bruk både UI og BLL for å simulere objekter som sendes begge veier. Det er nødvendig for å teste at vi kan sende en spørring eller et objekt til BLL og få returnert korrekt DTO. Det er ingen referanse mellom enhetstest og DAL eller database. Test bruker egne testobjekter. For å teste DAL og databasen måtte det ha vært implementert en integrasjonstest. Alle lagene i backend-systemet har en referanse til Model, og det er nødvendig for at applikasjonen skal kunne lese de forskjellige objektene korrekt i de forskjellige lagene. Model inneholder ikke noe informasjon om selve databasen utover de modellene som blir brukt av Entity Framework og Code First metoden under generering av databasen (se kapittel 4.3.3.6). Selve konfigurasjonsfilene til databasen ligger under DAL.

#### 4.3.3.8 Ytelse og optimalisering

For å teste ytelsen til systemet opprettet vi en ny lokal database, hvor vi autogenererte en del poster og kommentarer. Når vi startet med disse testene fungerte systemet slik at alle poster som lå i databasen ble sendt i en samlet liste til frontend hver gang hovedsiden ble lastet inn. Et forsøk med 100000 poster resulterte i en liste som var litt i underkant av 30 MB. Responsen ble betraktelig tregere som et resultat av dette. La vi til kommentarer på disse postene, ble de sortert i frontend når man åpnet en post. Dette reduserte ytelsen ytterligere,

og i et forsøk med en million kommentarer, fordelt på disse 100000 postene, stoppet hele systemet.

## Tiltak

For å forbedre ytelsen, ble løsningen å flytte paginering, søk og sortering til backend-systemet. Alle metodene som brukes av frontend-systemet hvor det sendes lister ble derfor endret. Når vi gjør kall til disse metodene nå, sender vi med noen variabler for å fortelle API-et hva vi ønsker å hente ut. Et eksempel på dette kan vi se på figur 4.3.3.8a som viser koden i Controlleren for å hente ut en liste over kommentarer.

```
// GET: Comments
// GET: Comments?postId=1&pageNumber=1&pageSize=10&sortOrder=Asc&sortType=Date
[HttpGet]
3 references
public async Task<ActionResult<IEnumerable<CommentDTO>>> GetComments(int? postId, int? pageNumber, int? pageSize, string sortOrder, string sortType)
{
    try
    {
        // Liste kommentarer med paging
        var page = pageNumber ?? 1;
        var size = pageSize ?? 10;
        var order = sortOrder ?? "Asc";
        var type = sortType ?? "Date";

        return Ok(await _commentBLL.PagedList(postId, page, size, order, type));

        // Liste kommentarer uten paging
        //return Ok(await _commentBLL.GetComments(postId));
    }
    catch (Exception)
    {
        return StatusCode(StatusCodes.Status500InternalServerError, "Feil ved henting av kommentarer");
    }
}
```

*Figur 4.3.3.8a: Metoden for å hente liste over kommentarer*

Variablene som brukes av denne metoden er som følger

- postId for å velge den posten vi ønsker kommentarene til
- pageNumber for å velge sideantall i den paginerte listen
- pageSize for å velge hvor mange kommentarer det skal vises på en side
- sortOrder for å velge stigende eller synkende rekkefølge på sortering
- sortType for å velge hvilket attributt fra databasen det skal sorteres etter

Metodene for å søke er nesten identiske med metodene for paginerte lister. Den eneste forskjellen er at disse metodene også tar en variabel for søkeordet. På figur 4.3.3.8b ser vi koden til søkefunksjonen for kommentarer.

```
// GET: Comments/Search?query=eksempel tekst
// GET: Comments/Search?query=eksempel tekst&postId=1&pageNumber=1&pageSize=10&sortOrder=Asc&sortType=Date
[HttpGet("{Search}")]
3 references
public async Task<ActionResult<IEnumerable<CommentDTO>>> Search(string query, int? postId, int? pageNumber, int? pageSize, string sortOrder, string sortType)
{
    try
    {
        // Liste søker i kommentarer med paging
        var page = pageNumber ?? 1;
        var size = pageSize ?? 10;
        var order = sortOrder ?? "Asc";
        var type = sortType ?? "Date";

        return Ok(await _commentBLL.Search(query, postId, page, size, order, type));
    }
    catch (Exception)
    {
        return StatusCode(StatusCodes.Status500InternalServerError, "Feil ved søker i kommentarer");
    }
}
```

Figur 4.3.3.8b: Metoden for å søke i kommentarer

Som vi kan se, tar denne metoden en variabel som heter "query" i tillegg til de samme variablene vi så i metoden for liste. Query-variablene er en string med søkeord det skal hentes ut objekter etter.

I koden som aksesserer databasen i Data Access Layer-klassene til alle metodene som bruker paginering og søker har vi benyttet oss av to forskjellige NuGet-pakker. Disse er Dynamic LINQ for å forenkle sorteringen med dynamiske spørrenger og X.PagedList for selve pagineringen. Siden paginerte lister og søkermetoder er strukturert ganske likt, går vi kun gjennom koden for søker. På figur 4.3.3.8c ser vi nærmere på koden for søker i kommentarer.

```
2 references
public async Task<Response<IEnumerable<Comment>>> Search(string query, int? postId, int page, int size, string order, string type)
{
    if (!string.IsNullOrEmpty(query))
    {
        IEnumerable<Comment> list;
        if (postId != null)
        {
            list = await _context.Comments.AsQueryable().Where(q => q.PostId == postId).OrderBy(type + " " + order).ToListAsync();
        }
        else
        {
            list = await _context.Comments.AsQueryable().OrderBy(type + " " + order).ToListAsync();
        }

        var searchList = list.Where(q => q.Content.ToLower().Contains(query.ToLower()));
        var count = searchList.Count();
        var pagedSearchList = await searchList.ToPagedListAsync(page, size);
        return new Response<IEnumerable<Comment>>(pagedSearchList, count);
    }
    else
    {
        return null;
    }
}
```

Figur 4.3.3.8c: Kodene som aksesserer databasen for søker i kommentarer

Det som skiller søker fra lister er som tidligere nevnt at disse metodene også tar inn en variabel for søkerordet. Det er naturlig å returnere en tom liste hvis det ikke er inkludert et søkerord. Det gjøres derfor en sjekk (som vist på figur 4.3.3.8d) som kun kjører koden om

query-variabelen har innhold, ellers returneres en nullverdi. Denne løkken er derfor kun med i koden for søker, og ikke i koden for liste.

```
if (!string.IsNullOrEmpty(query))...
else
{
    return null;
}
```

Figur 4.3.3.8d: If-løkken som sjekker query-variabelen sitt innhold

I denne løkken har vi nøstet en ny if-løkke som ser etter "postId". Angis ikke postId, hentes hele listen, ellers hentes kun kommentarer som tilhører den angitte posten. Denne løkken brukes derfor i koden for både søker og liste. På figur 4.3.3.8e ser vi nærmere på denne koden.

```
if (postId != null)
{
    list = await _context.Comments.AsQueryable().Where(q => q.PostId == postId).OrderBy(type + " " + order).ToListAsync();
}
else
{
    list = await _context.Comments.AsQueryable().OrderBy(type + " " + order).ToListAsync();
}
```

Figur 4.3.3.8e: Listene over kommentarer vi skal søker i

Tar vi for oss listen over kommentarer som tilhører en post, ser vi at disse objektene filtreres ut med koden "Where(q => q.PostId == postId)". Her henter vi altså kun ut de objektene som har tilsvarende ID lik den vi sendte med i metodekallet. Begge listene sorteres på samme måte. Det ser vi på koden "OrderBy(type + " " + order)". Dette er en dynamisk spørring som tar i bruk Dynamic LINQ pakken. Dette har forenklet koden så mye at vi kun trenger en kodelinje for å hente ut listen. Til å begynne med besto denne kode-blokkene av en større og mer komplisert Switch Statement. Med den dynamiske spørringen forteller vi at det skal sorteres etter type og order. Type er attributtet til kommentaren fra databasen, som for eksempel kan være dato. Order angir om det skal være en stigende eller synkende rekkefølge som bestemmes etter attributtets verdi.

Når vi skal søker i den bestemte listen, lager vi en ny liste med en where-klausul som søker etter det angitte søkerordet fra query-variabelen. På figur 4.3.3.8f ser vi at denne spørringen konverterer alle bokstaver fra både innholdet i databasen og query-variabelen til små bokstaver. Dette bidrar til å gjøre søkerfunksjonen enklere for sluttbrukere, da de ikke trenger å være så nøyaktig med søkerordene de taster inn.

```

var searchList = list.Where(q => q.Content.ToLower().Contains(query.ToLower()));
var count = searchList.Count();
var pagedSearchList = await searchList.ToPagedListAsync(page, size);
return new Response<IEnumerable<Comment>>(pagedSearchList, count);

```

*Figur 4.3.3.8f: Koden for søk og paginering*

I koden for å kun hente ut en liste er ikke denne where-klausulen med. Når vi skal returnere lister tilbake til Controlleren, lages det en ny liste av resultatet som gjøres om til en paginert liste ved hjelp av x.PagedList pakken. Dette forenklet koden ytterligere, da vi her også kun trenger en kode-linje fremfor en større og mer komplisert Switch Statement. Totalt sett har disse to NuGet-pakkene gitt oss en meget pen og oversiktlig kode som vil være enklere for oppdragsgiver å sette seg inn i om de skulle ønske å forandre på dette ved et senere tidspunkt.

Listene som sendes tilbake er formatert med en Response-konstruktør vi har laget for å bygge opp JSON-objektet vi skal sende til frontend-systemet på en måte som gjør det logisk å lese essensiell informasjon som sidenummer, antall objekter per side, totalt antall sider og totalt antall objekter.

## Resultater

Vi testet ytelsen til systemet på nytt i postman med de nye endringene. I forsøket med en million kommentarer, så vi det beste resultatet når vi sorterte etter bestemte poster. På figur 4.3.3.8g viser vi et av forsøkene fra postman hvor det hentes ut en liste over kommentarer til en bestemt post.

Body Cookies Headers (5) Test Results

200 OK 15 ms 2.27 KB Save Response ▾

Pretty Raw Preview Visualize JSON

```

1
2   "id": 1,
3   "pageNumber": 1,
4   "pageSize": 10,
5   "totalPages": 1,
6   "totalRecords": 10,
7   "sortOrder": "Asc",
8   "sortType": "Date",
9   "data": [
10    {
11      "id": 97489,
12      "content": "Ja neida, sååå... OKEY!!!",
13      "date": "2021-05-18T17:30:16.8234138",
14      "editDate": null,
15      "edited": false,
16      "like_Count": 0,
17      "username": "testbruker71",
18      "userId": 5,
19      "postId": 1,
20      "documentId": null
21    },

```

Figur 4.3.3.8g: Resultat fra postman med paginert liste fra kommentarer

Denne posten har 10 kommentarer. Størrelsen på denne listen er så vidt i overkant av 2 KB og den ble hentet på noen få millisekunder. Som vi kan se her ble systemet meget responsivt igjen. Ved henting av større lister, tar det litt lengre tid. Det vil aldri være nødvendig å hente ut hele listen med alle kommentarer samtidig, så den største listen som hentes ut er derfor over alle poster. På figur 4.3.3.8h kan vi se resultatet av en liste med 100000 poster.

Body Cookies Headers (5) Test Results

200 OK 924 ms 2.93 KB Save Response ▾

Pretty Raw Preview Visualize JSON

```

1
2   "id": null,
3   "pageNumber": 1,
4   "pageSize": 10,
5   "totalPages": 10000,
6   "totalRecords": 100000,
7   "sortOrder": "Asc",
8   "sortType": "Date",
9   "data": [
10    {
11      "id": 1,
12      "title": "En dag på banen",
13      "content": "Nei forresten, bare glem det!",
14      "date": "2021-05-18T17:30:16.8234138",
15      "editDate": null,
16      "edited": false,
17      "comment_Count": 0,
18      "like_Count": 0,
19      "username": "testbruker98",
20      "userId": 28,
21      "topicId": 5,

```

Figur 4.3.3.8h: Resultat fra postman med paginert liste fra poster

Som vi kan se er responstiden for å få en paginert liste med 100000 poster litt i underkant av ett sekund. Dette anser vi som et meget godt resultat, og denne responstiden er omrent ikke merkbar for sluttbrukere av frontend-applikasjonen. Vi observerte tilsvarende resultater når vi testet søkefunksjonene.

#### 4.3.3.9 Brukerautentisering

I henhold til kravspesifikasjonen på kapittel 3.1.3 under 3.1.3.1 Forretningskrav, ytret oppdragsgiver et ønske om integrering med Idrettens ID. Ettersom vi ikke fikk implementert dette (som beskrevet i kapittel 2.2.1.4), valgte vi en løsning hvor sluttbrukere registerer egne brukere med passord. Tidlig i utviklingen hadde vi allerede lagt til brukernavn og passord som felter i databasen, så vi bygget videre på dette konseptet.

I kravspesifikasjonen (kapittel 3.1.3.1) under punkt "F4: Brukere må være pålogget for å bruke nettsiden", ønsker ikke oppdragsgiver at uregistrerte brukere skal ha tilgang til systemet. For å autentisere brukere og begrense tilgang til systemet for uautoriserte brukere, ble det derfor implementert ekstra sikkerhet rundt brukere, i form av krypterte passord og autorisasjon med JSON Web Tokens.

#### Kryptering av passord

Lagring av passord med klartekst i databasen er en særdeles usikker måte å lagre sensitiv brukerinformasjon på, derfor implementerte vi noen steg for å sikre disse passordene på en forsvarlig måte. Når en ny bruker registreres i backend-systemet, generes det først en tilfeldig verdi for salting, deretter lages en hash av passordet som lagres i databasen.

Salting ble gjort med RNGCryptoServiceProvider som er en innebygget klasse i C# for generering av tilfeldige siffer. RNG står for Random Number Generator. I figur 4.3.3.9a vises koden vi har lagt for dette.

```
2 references
public static byte[] AddSalt()
{
    var csprng = new RNGCryptoServiceProvider();
    var salt = new byte[24];
    csprng.GetBytes(salt);
    return salt;
}
```

*Figur 4.3.3.9a: Metoden for salting av passord*

Denne metoden lager et byte array vi skal bruke til saltingen av passordet. Variabelen csprng er en forkortelse for cryptographically secure pseudorandom number generator, som er navnet på funksjonen vi bruker når det genereres uforutsigbare sifre.

Hashing av passordet ble gjort med Rfc2898DeriveBytes som er en annen innebygget klasse i C# biblioteket som benytter salting for å generere hash. Figur 4.3.3.9b viser koden vi laget for dette.

```
2 references
public static byte[] AddHash(string password, byte[] salt)
{
    const int keyLength = 24;
    var pbkdf2 = new Rfc2898DeriveBytes(password, salt, 1000);
    return pbkdf2.GetBytes(keyLength);
}
```

Figur 4.3.3.9b: Metoden for hashing av passord

Variabelen pbkdf2 er en forkortelse for Password-Based Key Derivation Function 2, som er navnet på funksjonen vi bruker for hashing. Her ser vi at funksjonen tar inn passordet, salten og antall ganger denne algoritmen skal itereres (vi har satt 1000).

Både salting og hashing er unikt for hvert passord som lages, og begge deler lagres i databasen. Når en bruker skal logge inn, hentes salten og en ny hash genereres av passordet brukeren har oppgitt. Om den nye hashen stemmer med hashen som ligger i databasen, stemmer passordet, og brukeren logges inn. Vi ser koden for dette på figur 4.3.3.9c.

```

2 references
public async Task<AuthResponse> Login(AuthRequest request)
{
    // Liste over alle eksisterende brukere
    var users = await _context.Users.ToListAsync();
    if (users != null)
    {
        foreach (var user in users)
        {
            // Sjekk om brukernavn eller epost stemmer
            if (user.Username == request.Username || user.Email == request.Email)
            {
                // Sjekk passord med kryptering
                const int keyLength = 24;
                var pbkdf2 = new Rfc2898DeriveBytes(request.Password, user.Salt, 1000);
                byte[] passwordTest = pbkdf2.GetBytes(keyLength);
                bool passwordCheck = user.Password.SequenceEqual(passwordTest);
                if (passwordCheck == true)
                {
                    // Sende tilbake brukerobjekt hvis ok
                    return new AuthResponse(user);
                }
            }
        }
    }

    // Sende tilbake null hvis bruker eller passord er feil
    return null;
}

```

*Figur 4.3.3.9c: Koden for autentisering av brukere*

Her gjøres det en sjekk om sekvensen på bruker sitt passord og passwordTest-variabelen er lik. Hvis dette stemmer, returneres et nytt AuthResponse-objekt av denne brukeren. Dette objektet har et eget felt for JSON Web Token som er sikkerhetsnøkkelen denne brukeren får tildelt hvis innloggingen lykkes.

### JSON Web Token

Alle metoder utenom registrering av nye brukere og innlogging er låst for uautoriserte brukere. Dette betyr at bruker må være logget inn for å gjøre noe annet en dette. For å håndtere denne autorisasjonen har vi tatt i bruk NuGet-pakken Microsoft.AspNetCore.Authentication.JwtBearer. Vi har til nå sett på sikkerheten rundt passordet. Når en bruker logges inn blir det tildelt en sikkerhetsnøkkelen. I figur 4.3.3.9d ser vi på koden fra CustomController når en bruker logger inn.

```
// POST: Login
[AllowAnonymous]
[HttpPost]
3 references
public async Task<ActionResult<AuthResponse>> Login([FromForm] AuthRequest resquest)
{
    try
    {
        var response = await _customBLL.Login(resquest);
        if (response != null)
        {
            // Ok hvis brukernavn/epost og passord stemmer
            var jwtToken = _tokenService.GenerateJwtToken(response);
            response.Token = jwtToken;
            return Ok(response);
        }
        else
        {
            return Unauthorized("Feil ved brukernavn, epost eller passord");
        }
    }
    catch (Exception)
    {
        return StatusCode(StatusCodes.Status500InternalServerError, "Feil ved login");
    }
}
```

*Figur 4.3.3.9d: Login-metoden fra CustomController*

Denne metoden tar inn et AuthRequest-objekt og returnerer et AuthResponse-objekt hvis innloggingen var vellykket. Hvis det returneres null fra underliggende klasser, stemmer ikke passordet bruker har oppgitt, og innloggingen mislykkes. AuthRequest-objektet består av brukernavn, epost og passord (se figur 4.3.3.9e).

```
6 references
public class AuthRequest
{
    3 references
    public string Username { get; set; }
    1 reference
    public string Email { get; set; }
    2 references
    public string Password { get; set; }
}
```

*Figur 4.3.3.9e: AuthRequest-objektet*

Dette er alt vi trenger for å autentisere en bruker. Det er valgfritt for brukere om de vil logge inn med brukernavn eller epost. Gitt at innloggingen var vellykket, returneres AuthResponse-objektet vi kan se på figur 4.3.3.9f.

```

15 references
public class AuthResponse
{
    2 references
    public int Id { get; set; }
    3 references
    public string Username { get; set; }
    1 reference
    public string FirstName { get; set; }
    1 reference
    public string LastName { get; set; }
    2 references
    public string Email { get; set; }
    1 reference
    public bool Admin { get; set; }
    3 references
    public string Token { get; set; }

    2 references
    public AuthResponse(User user)
    {
        Id = user.Id;
        Username = user.Username;
        FirstName = user.FirstName;
        LastName = user.LastName;
        Email = user.Email;
        Admin = user.Admin;
        Token = null;
    }
}

```

Figur 4.3.3.9f: AuthResponse-objektet med tilhørende konstruktør

Dette objektet har alle feltene vi ønsker å vise til frontend-systemet, og har en egen konstruktør som fyller ut gitte felter fra bruker-objektet i databasen. Det som skiller AuthResponse-objektet fra et vanlig bruker-objekt er feltet for Token. Det er her vi legger ved sikkerhetsnøkkelen som autentiserer en bruker i systemet. Denne sikkerhetsnøkkelen er en autogenerert streng basert på en hemmelighet vi henter fra filen appsettings.json. Når frontend-systemet skal gjøre metodekall til API-et, inkluderes denne nøkkelen for å fortelle backend-systemet at brukeren er autorisert.

ASP.NET har meget god innebygget støtte for autentisering og autorisasjon av brukere. NuGet-pakken vi har brukt for dette registreres i Startup.cs klassen for dependency injection, og brukes i Controllere for å fortelle systemet hvilke metoder som krever autorisasjon. Hvis en metode er merket med [Authorize] betyr det at denne metoden krever autorisasjon. Vi kan også merke hele Controlleren på denne måten, og da låses alle funksjonene. De metodene som skal være tilgjengelig for alle, merkes med [AllowAnonymous]. Da trenges ingen autorisasjon for å benytte denne metoden.

#### 4.3.3.10 Filhåndtering

En viktig del av webapplikasjonen var at brukere skulle dele dokumenter med hverandre. Vi testet noen forskjellige løsninger for dette før vi var fornøyde. Vi ønsket også at filer som ble lastet opp skulle være tilgjengelig for oppdragsgiver og IT-ansvarlig utenfor selve applikasjonen.

For å ha en logisk forbindelse mellom filer, brukere og posten/kommentaren filen ble lastet opp i, var det nødvendig å lage en egen modell for dette i databasen. I et av de tidligste forsøkene lagret vi også filinnhold i databasen som et byte array, men denne løsningen kunne vi ikke bruke. Med filinnhold liggende i databasen, vil ikke bare databasen bli unødvendig stor, men ytelsen vil også rammes. Det ble derfor nødvendig å ha filer for seg selv, og kun en referanse med informasjon om filene i databasen. Med en modell for filer på plass, implementerte vi deretter en løsning hvor filene ble lastet opp i en egen mappe i backend-systemet. Dette tillot oss å separere filinnhold fra databasen, men vi manglet fortsatt en enkel måte å gi oppdragsgiver tilgang til disse filene utenfor webapplikasjonen.

Siden oppdragsgiver hadde ytret et ønske om å publisere webapplikasjonen på Azure, ble det endelige valget å laste opp filer til Azure også. Azure har en egen service for behandling av filer som er enkel å bruke, og samtidig gir tilgang til alle filer via Azure Portal. For å håndtere filer, har vi tatt i bruk forskjellige NuGet-pakker som forenkler denne prosessen.

Funksjoner for å laste opp, laste ned og slette filer er gjort med Azure.Storage.Blob-pakken. Med denne pakken kan vi lage «Containers» som blir synlige mapper på Azure Portal. For å ha en logisk sortering på filer, fikk disse mappene et navn som tilsvarer «måned-årstall» (for eksempel «mai-2021»). I denne mappen opprettes nye mapper, hvor brukernavn blir navnet, og i disse mappene ligger selve filene. For at systemet skal skille på forskjellige filer, siden en bruker kan laste opp filer med samme filnavn, har en unik ID blitt lagt til filnavnet. Referansen til denne ID-en lagres i databasen sammen med originale filnavnet. Strukturen på filer i Azure Portal er derfor: Dato -> Brukernavn -> Filnavn (ID).

Filer som skal lastes opp i backend-systemet, er definert som «`IFormFile`»-objekter. Dette er en del av `Microsoft.AspNetCore.Http.Features`-pakken. `IFormFile`-interfacet gir oss en meget simpel metode for å behandle filer som sendes over HTTP-requests.

For at sluttbrukere ikke skal laste ned filer slik de er lagret i Azure (filnavn med unik ID), brukte vi Microsoft.AspNetCore.Mvc.Core-pakken. Denne pakken har en egen klasse som heter FileStreamResult. Med konstruktøren til denne klassen kan vi opprette en fil med ønsket filnavn og innhold. Når en fil lastes ned, vil selve filen hentes fra Azure og det originale filnavnet som ligger i databasen brukes.

Med disse løsningene fikk vi implementert filhåndtering slik vi ønsket. Alle filer som lastes opp er nå tilgjengelig via webapplikasjonen og Azure Portal. IT-ansvarlig hos oppdragsgiver har da tilgang til alle filer om webapplikasjonen skulle bli utilgjengelig. Det er viktig å merke seg at filer ikke må slettes direkte i Azure Portal, siden det fortsatt vil ligge en oppføring av denne filen i databasen. Om en fil slettes i Azure Portal, vil det se ut som den er tilgjengelig i webapplikasjonen, men det vil ikke være mulig å laste den ned.

#### **4.3.3.11 NuGet-pakker**

##### **Azure.Storage.Blobs**

Azure Storage Blobs client library for .NET brukes for håndtering av dokumenter som skal lagres og hentes fra skyene. Dette krever en egen Azure Storage konto som må konfigureres i appsettings.json før publisering av systemet. Alle filer som lastes opp vil da også være tilgjengelig via Azure Portal.

##### **Microsoft.Extensions.Azure**

Azure client library integration for ASP.NET Core brukes for å registrere Azure Storage Blob i Startup.cs klassen med dependency injection. Dette tillater konfigurering av Azure-kontoen i appsettings.json filen.

##### **Microsoft.AspNetCore.Http.Features**

Denne pakken heter ASP.NET Core HTTP feature interface definitions. Dette er en utvidelse av ASP.NET vi har brukt for å forenkle hvordan systemet tar imot og sender tilbake filer i tilhørende metodekall ved å benytte IFromFile interfacet.

##### **Microsoft.AspNetCore.Mvc.Core**

ASP.NET Core MVC core components er enda en ASP.NET utvidelse vi har brukt for håndtering av filer. Denne pakken tillater oss å sende filer som et FileStreamResult, som kort fortalt tillater oss å tilpasse filen som skal lastes ned av sluttbrukere i frontend-systemet. Filen tilpasses ved at filen hentes fra skyene, og filnavnet hentes fra databasen. Uten dette

ville sluttbrukere lastet ned filer med filnavn basert på en autogenerert ID, som igjen ville ført til en ulogisk og dårlig brukeropplevelse fra en sluttbruker sitt synspunkt.

### **Microsoft.EntityFrameworkCore.SqlServer**

Dette er pakken for Entity Framework Core som brukes av systemet for alle funksjoner som er knyttet til databasen.

### **System.Linq.Dynamic.Core**

Dynamic LINQ er et åpent kildekode-prosjekt som tillater bruk av dynamiske spørninger.

Denne pakken er brukt for å forenkle spørninger ved søk i databasen som beskrevet i kapittel 4.3.3.8 under tiltak.

### **X.PagedList**

Denne pakken er brukt for å forenkle paginering av diverse lister, slik vi har beskrevet i under tiltak i kapittel 4.3.3.8.

### **Moq**

Denne pakken har vi brukt for å simulere interfaces i enhetstesting. Les mer om dette i kapittel 2.4.2.

### **xUnit.net**

xUnit.net er et åpent kildekode-rammeverk for å lage enhetstester. Denne pakken har god støtte for C#-kodespråket, og er meget godt dokumentert på nettet med diverse tutorials og diskusjoner på forskjellige forum, noe vi har benyttet oss av når vi tok det i bruk.

Enhetstesting er beskrevet i kapittel 2.4.2.

### **xunit.runner.visualstudio**

Dette er en tilleggspakke for xUnit-rammeverket som tillater oss å kjøre enhetstestene vi har laget direkte i Visual Studio.

### **Microsoft.AspNetCore.Authentication.JwtBearer**

Dette er pakken vi har brukt for brukerautentisering. Denne pakken er et ASP.NET Core middleware som gjør det mulig å ta i bruk JSON Web Tokens (kapittel 4.3.3.9).

### 4.3.3.12 Liste API-funksjoner

Følgende liste viser en oversikt over alle metoder til typiske Controllere.

#### **CommentsController**

- GET: Comments
- GET: Comments?postId=1&pageNumber=1&pageSize=10&sortOrder=Asc&sortType=Date
- GET: Comments/1
- GET: Comments/Search?query=eksempel tekst
- GET: Comments/Search?query=eksempel tekst&postId=1&pageNumber=1&pageSize=10&sortOrder=Asc&sortType=Date
- POST: Comments
- PUT: Comments/1
- DELETE: Comments/1

#### **InfoTopicsController**

- GET: InfoTopics
- GET: InfoTopics/1
- POST: InfoTopics
- PUT: InfoTopics/1
- DELETE: InfoTopics/1

#### **PostsController**

- GET: Posts
- GET: Posts?subTopicId=1&pageNumber=1&pageSize=10&sortOrder=Asc&sortType=Date
- GET: Posts/1
- GET: Posts/Search?query=eksempel tekst
- GET: Posts/Search?query=eksempel tekst&subTopicId=1&pageNumber=1&pageSize=10&sortOrder=Asc&sortType=Date
- POST: Posts
- PUT: Posts/1
- DELETE: Posts/1

#### **SubTopicsController**

- GET: SubTopics
- GET: SubTopics/1
- POST: SubTopics
- PUT: SubTopics/1
- DELETE: SubTopics/1

#### **TopicsController**

- GET: Topics
- GET: Topics/1
- POST: Topics
- PUT: Topics/1
- DELETE: Topics/1

#### **UsersController**

- GET: Users
- GET: Users?pageNumber=1&pageSize=10&sortOrder=Asc&sortType=Id
- GET: Users/1

GET: Users/Search?query=eksempel tekst  
 GET: Users/Search?query=eksempel tekst&pageNumber=1&pageSize=10&sortOrder=Asc&sortType=Id  
 [AllowAnonymous] POST: Users  
 PUT: Users/1  
 DELETE: Users/1

#### **VideosController**

GET: Videos  
 GET: Videos?infoTopicId=1&pageNumber=1&pageSize=10&sortOrder=Asc&sortType=Id  
 GET: Videos/1  
 GET: Videos/Search?query=eksempel tekst  
 GET: Videos/Search?query=eksempel tekst&infoTopicId=1&pageNumber=1&pageSize=10&sortOrder=Asc&sortType=Id  
 POST: Videos  
 PUT: Videos/1  
 DELETE: Videos/1

Følgende liste viser en oversikt over alle metoder til ikke-typiske Controllere.

#### **CustomController**

GET: GetDocuments  
 GET: GetDocuments?infoTopicId=1&pageNumber=1&pageSize=10&sortOrder=Asc&sortType=Id  
 GET: GetDocumentInfo/1  
 GET: GetDocument/1  
 GET: SearchDocuments?query=eksempel tekst  
 GET: SearchDocuments?query=eksempel tekst&infoTopicId=1&pageNumber=1&pageSize=10&sortOrder=Asc&sortType=Id  
 POST: UploadDocument  
 [AllowAnonymous] POST: Login  
 POST: SetAdmin  
 POST: SetUsername  
 DELETE: DeleteDocument/1

#### **LikesController**

POST: GetLike  
 POST: AddLike  
 DELETE: DeleteLike

I disse listene kan vi se hvilke metoder som er tilgjengelige, og hvordan de brukes. Det er mange metoder, og flere av dem kan ta forskjellige parametere. Et eksempel på dette kan vi se under tiltak i kapittel 4.3.3.8, hvor vi utvidet systemet med paginerte lister, sortering og søk. Det er to metoder som er merket med [AllowAnonymous], og disse er beskrevet i kapittel 4.3.3.9.

For å bruke de forskjellige metodene, må det gjøres metodekall til backend-systemet sin URL etterfulgt av Controlleren det skal gjøres et kall mot (HTTP metoder er beskrevet i kapittel 4.3.3.3). Ønsker vi for eksempel å hente en liste over poster, skal dette gjøres med en GET

metode til <https://example.com/Posts> (hvor example.com erstattes med URL til systemet). Alle variablene er vist på listen, men det er valgfritt hvilke variabler som brukes. Skal det hentes en spesifikk post, må det legges ved ID. Da bruker vi Posts/1 hvor tallet 1 her representerer post med ID nr.1.

#### 4.3.3.13 Til ettertanke

Backend-systemet er tilpasset frontend-systemet i en så høy grad, at det kan være vanskelig å gjøre endringer for noen uten god kjennskap til begge systemene. Metoder som ikke er selvforklarende, eller som har en utvidet/tilpasset funksjon utover navnet på selve metoden, har derfor blitt kommentert i backend-koden. Disse kommentarene er rettet mot individer som eventuelt skal gjøre endringer eller fortsette med utviklingen av systemet senere.

I kapittel 4.3.3.8 under Resultat så vi på forbedringer av ytelsen til systemet. Disse resultatene kunne vært forbedret ytterligere, men det ville gått på bekostning av noe funksjon. I Business Logic Layer-klassen hvor det lages DTO-er over objekter (kapittel 4.3.3.7), er det lagt til en ekstra spørring når vi henter liste over alle poster. Denne spørringen henter brukernavnet på eieren til hver enkelt post i listen. Vi valgte å gjøre det på denne måten så brukernavnet som vises på forumet alltid er oppdatert med gjeldende brukernavn.

Systemet er enhetstestet (se kapittel 2.4.2), men det er ikke implementert noen integrasjonstester. Ideelt sett burde dette vært lagt til, men vi valgt å prioritere ferdigstilling av systemet og brukertesting over dette. Vi har selv testet alle metoder og funksjoner underveis i utviklingen, og plukket derfor opp feil og mangler underveis. Med en kontinuerlig testing fra vår side, føler vi at systemet lever opp til den kvaliteten vi så for oss når vi startet med bacheloroppgaven, og vi er totalt sett fornøyd med produktet vi har utviklet.

Kapittel 5

# **Resultat av brukertesting**

# Innhold

<b>Resultat av brukertesting .....</b>	<b>1</b>
<b>5.1 Brukertesting .....</b>	<b>3</b>
5.1.1 Innledning.....	3
5.1.2 Brukertester .....	4
5.1.3 Kartlegging av samlet resultat.....	9

## 5.1 Brukertesting

I dette kapittelet gjennomgås brukertestingaen som ble gjennomført av systemet, samt resultatene av disse og hvilke eventuelle tiltak som ble tatt for å sikre sluttbrukernes opplevelse av webapplikasjonen.

### 5.1.1 Innledning

---

For å finne ut hvorvidt NBF var fornøyd med resultatet av utviklingen, sendte vi ut et skjema med testscenarioer slik at de kunne utføre brukertesting internt. Ideelt hadde vi ønsket å utføre testene på annet vis, men på grunn av Covid-19-pandemien ble ikke dette aktuelt.

#### 5.1.1.1 Formål

Formålet med brukertestene var å finne ut av om NBF var fornøyde med sluttproduktet og hvordan sluttbrukere opplever produktet. For å gjøre dette, sendte vi ut et testskjema til NBF nærmest slutten av prosjektet. Dette testskjemaet (se 5.1.2) inneholdt en rekke scenarier testpersonene skulle gjennomføre etterfulgt av noen kontrollspørsmål om deres opplevelse av webapplikasjonen. Deretter gjorde vi de forandringene som var nødvendige og gjennomførbare basert på testpersonenes tilbakemelding.

#### 5.1.1.2 Omfang

Testen så på alle mulige scenarier som kunne gjennomføres i webapplikasjonen, altså ble alt mulig testet. Da det skulle simulere en sluttbrukers bruk av webapplikasjonen, måtte alt som kunne testes, testes. Som utviklere må vi tilrettelegge for at alt som er inkludert i løsningen, skal brukes på et tidspunkt. Dermed ønsket vi å hente inn så mye data som mulig for å gjøre det enklere for oss selv å utrede eventuelle feil og mangler som kan påvirke sluttbrukernes opplevelse. Dog er det viktig å nevne at vi fremdeles implementerte funksjoner i webapplikasjonen parallelt med dette, og ikke alle disse funksjonene kunne testes på samme linje.

## 5.1.2 Brukertester

---

Dette delkapitlet består av dokumentet gruppa sendte ut til NBF for å utføre brukertester. Her er testens formål beskrevet, scenariene testpersonene skulle gjennomføre, samt spørsmål de skulle fylle ut etter å ha gjennomført testen.

### 5.1.2.1 Brukertest av Kunnskapsportalen Badminton

Dette dokumentet definerer brukertestene for Kunnskapsportalen Badminton. Formålet med denne brukertesten er å få informasjon om hvordan sluttbrukere av webapplikasjonen opplever den, slik at utviklerne kan tilpasse den etter sluttbrukernes behov.

Dersom du har mulighet, ber vi deg om å bruke både PC/Mac og mobil. Markér dette i kolonnene til høyre nedenfor. Dersom du ikke kan bruke begge, trenger du kun å fylle ut kolonnene som gjelder.

PC eller Mac	
Nettleser	
Mobil og modell	
Nettleser	

Testene er delt inn forskjellige deler etter hvilken side man er på i applikasjonen. Alle disse delene har hver sin tabell. I kolonnen til venstre finner du test-Id. Denne trenger ikke du som bruker å forholde deg til, den er kun ment for utviklerne til å navigere enklest mulig gjennom resultatet av de forskjellige testene. I midterste kolonne er scenarioet, altså testen du som testbruker skal gjennomføre. I høyre kolonne defineres resultatet av testen. Her skal du markere om testen var vellykket, for eksempel med "Ja" eller "x". Dersom testen ikke var vellykket, altså det skjedde en feil eller du som bruker ikke ble tilfredsstilt av resultatet, kan du legge til en kort beskrivelse av hva som skjedde (eks. "Fikk feilmelding" o.l.). Etter det ber vi deg om å svare kort på noen spørsmål om din opplevelse av testene og generelt som bruker av applikasjonen.

Vi vil gjøre oppmerksom på at ikke alle funksjonaliteter og elementer er implementert enda.

Vi jobber kontinuerlig med webapplikasjonen, så det du ser nå er antageligvis ikke det endelige resultatet.

Til slutt ber vi deg om å ikke spørre om hjelp fra andre til å gjennomføre testene; dersom det ikke er innlysende og du ikke klarer det selv, er dette kritikk til oss som utviklere som vi kan bruke til å forbedre webapplikasjonen.

## Tester

### A: Innloggings- og registreringsside (/Login)

Id	Scenario	Resultat
A1	Jeg vil registrere meg som ny bruker	
A2	Jeg vil logge meg inn med e-post	
A3	Jeg vil logge meg inn med brukernavn	
A4	Jeg vil nullstille passord dersom jeg har glemt det	

### B: Navigasjon

Id	Scenario	Resultat
B1	Jeg vil navigere til "Hjem"	
B2	Jeg vil navigere til "Forum"	
B3	Jeg vil navigere til "Kunnskapsportalen"	
B4	Jeg vil se min oppgitte brukerinformasjon	
B5	Jeg vil logge ut	

**C: Forside (Siden man kommer til etter å ha logget inn)**

Id	Scenario	Resultat
C1	Jeg vil se et utsnitt av forumpostene	
C2	Jeg vil sortere utsnittet av forumpostene	
C3	Jeg vil gå inn på en av trådene i utsnittet av forumpostene	

**D: Forum (/Forum)**

Id	Scenario	Resultat
D1	Jeg vil sortere etter kategorier	
D2	Jeg vil sortere etter underkategorier	
D3	Jeg vil se en kort beskrivelse om en underkategori	
D4	Jeg vil søke i poster	
D5	Jeg vil sortere poster	
D6	Jeg vil navigere meg gjennom sidene i forumet	
D7	Jeg vil justere antall poster per side	
D8	Jeg vil lese mer av en post uten å gå inn på den (gjelder kun hvis en post er lang)	
D9	Jeg vil lage en post uten et vedlegg i forumet	
D10	Jeg vil lage en post med et vedlegg	

**E: Tråd 1 - Egen tråd (/Forum/Id (Id er det unike tallet for en tråd))**

Id	Scenario	Resultat
E1	Jeg vil redigere min post	
E2	Jeg vil slette min post	

**F: Tråd 2 - En annen sin tråd**

Id	Scenario	Resultat
F1	Jeg vil se hvem som har laget posten og når den ble laget	
F2	Jeg vil se hvilken kategori og underkategori posten hører til	
F3	Jeg vil laste ned vedlegget til en post	
F4	Jeg vil like en post	
F5	Jeg vil unlike en post	
F6	Jeg vil kommentere på en post	
F7	Jeg vil legge til et vedlegg i en ny kommentar	
F8	Jeg vil redigere min kommentar	
F9	Jeg vil slette min kommentar	
F10	Jeg vil bla gjennom sidene i en tråd (dersom det er mange nok kommentarer)	
F11	Jeg vil gå tilbake til forumet fra en tråd	

**G: Kunnskapsportalen (/Kunnskapsportalen)**

Id	Scenario	Resultat
G1	Jeg vil sortere etter kategorier	
G2	Jeg vil se en kort beskrivelse om en kategori	
G3	Jeg vil se en liste over videoer	
G4	Jeg vil se på en video	
G5	Jeg vil gå til diskusjonstråden for en video	
G6	Jeg vil se en liste over dokumenter	
G7	Jeg vil laste ned et dokument	
G8	Jeg vil søke i kunnskapsportalen	

## H: Generelt

Id	Scenario	Resultat
H1	Jeg vil finne kontaktinformasjonen til Norges Badmintonforbund	
H2	Jeg vil finne Norges Badmintonforbund på sosiale medier	
H3	Jeg vil endre brukernavn	
H4	Jeg vil endre passord	

## Spørsmål

Hva var dine første tanker om webapplikasjonen?	
Hvordan opplevde du å bruke webapplikasjonen?	
Hva likte du?	
Hva likte du ikke?	
Noe annet å legge til?	

## 5.1.3 Kartlegging av samlet resultat

For å kartlegge resultatene, brukte vi en prosentskala for å avgjøre om endringer var nødvendige eller ikke. I tillegg måtte vi bruke mye av vår egen opplevelse av webapplikasjonen for å bestemme om noe måtte endres.

### 5.1.3.1 Metode

Ved å bruke en prosentskala for tilbakemeldingene, kunne vi enkelt avgjøre hvorvidt vi måtte gjøre endringer eller ikke. Her definerte vi en test med 100% gjennomføringsrate dersom testpersonen klarte å gjennomføre den uten problem, altså godkjent. Hvis en bruker klarte å gjennomføre testen, men møtte på noen problem underveis, ble dette definert som 50%, men likevel godkjent. Dersom forekomsten av 50% gjennomføringsrate for en test var stor, kunne dette bety at vi måtte gjøre endringer i webapplikasjonen. 0% ble angitt hvis brukeren ikke klarte å gjennomføre testen i det hele tatt. Deretter kunne vi regne ut den samlede prosent gjennomførelse for testen og vurdere hvorvidt vi burde gjøre tiltak for å rette opp i eventuelle feil utredet gjennom testingen. For eksemplen; hvis en test hadde 7 gjennomførelser med 100%, 1 med 50% og 1 med 0%, ville testen likevel få 89% gjennomføringsrate, og vi trengte ikke nødvendigvis å gjøre store endringer i webapplikasjonen basert på dette resultatet.

Innlogging- og registreringsside	100%	50%	0%	Sum %
A1	9			100%
A2	8		1	89%
A3	7		2	78%
A4	8		1	89%

Figur 5.1.3.1a: Eksempel for utregning av gjennomførelse i prosent

På grunn av korttenkhet fra vår side, og fordi vi tenkte at testpersonene ville interagere mer med nettsiden utover brukertestene, publiserte vi ikke nok innhold i testmiljøet som sluttbrukerne kunne teste med, se resultat for test-id D4 og D7 i figur 5.1.3.1b. Dermed kunne ikke enkelte av testene gjennomføres, og da så vi oss nødt til å bruke våre egne opplevelser med webapplikasjonen for å fylle ut scenarier som for testpersonene var uggjennomførbare.

#### D: Forum (/Forum)

Id	Scenario	Resultat
D1	Jeg vil sortere etter kategorier	X
D2	Jeg vil sortere etter underkategorier	X
D3	Jeg vil se en kort beskrivelse om en underkategori	X
D4	Jeg vil søke i poster	Vanskelig å sjekke med kun 1 sak
D5	Jeg vil sortere poster	X
D6	Jeg vil navigere meg gjennom sidene i forumet	X
D7	Jeg vil justere antall poster per side	Vanskelig å sjekke med kun 1 sak
D8	Jeg vil lese mer av en post uten å gå inn på den (gjelder kun hvis en post er lang)	X
D9	Jeg vil lage en post uten et vedlegg i forumet	X
D10	Jeg vil lage en post med et vedlegg	X

Figur 5.1.3.1b: Eksempel på tilbakemelding fra testperson

#### 5.1.3.2 Resultat

I alt fikk vi 9 tilbakemeldinger, ikke inkludert oss selv. Tabellen nedenfor tar for seg alle testene og den samlede prosenten for gjennomførelse. I tillegg forklarer vi eventuelle utstikkende resultater fra testene.

#### A: Innlogging- og registreringsside

Id	% Gjennomført
A1	100%
A2	89%
A3	78%
A4	89%

I innlogging- og registreringssida var gjennomføringsraten høy, men "A3: Jeg vil logge meg inn med brukernavn" stakk seg litt ut. Etter dialog med Charlotte Støelen viste det seg at enkelte av testpersonene trodde webapplikasjonen allerede var integrert med deres andre tjenester, og kunne derfor logge seg inn på den måten. Dette ble dermed en intern oppklaringssak fremfor noe vi som utviklere burde ta tak i.

### B: Navigasjon

Id	% Gjennomført
B1	100%
B2	100%
B3	100%
B4	89%
B5	100%

### C: Forside

Id	% Gjennomført
C1	100%
C2	100%
C3	100%

### D: Forum

Id	% Gjennomført
D1	100%
D2	89%
D3	100%
D4	100%
D5	100%
D6	100%
D7	100%
D8	89%

D9	89%
D10	89%

**E: Tråd 1**

Id	% Gjennomført
E1	89%
E2	89%

**F: Tråd 2**

Id	% Gjennomført
F1	100%
F2	100%
F3	78%
F4	100%
F5	100%
F6	100%
F7	100%
F8	100%
F9	100%
F10	100%
F11	100%

I tråd 2, test "F3: Jeg vil laste ned vedlegget til en post" var gjennomføringsraten 78%. Dette kan komme av at det ikke var noen tilgjengelige poster med vedlegg på tidspunktet for testingen, slik at enkelte av testpersonene ikke kunne gjennomføre.

## G: Kunnskapsportalen

Id	% Gjennomført
G1	100%
G2	100%
G3	100%
G4	100%
G5	100%
G6	100%
G7	78%
G8	78%

I kunnskapsportalen, test "G7: Jeg vil laste ned et dokument" kan det samme problemet som i F3 ha skjedd, at det ikke var noen dokumenter å laste ned på tidspunktet. "G8: Jeg vil søke i kunnskapsportalen" kan også komme av at det ikke var nok innhold å søke i.

## H: Generelt

Id	% Gjennomført
H1	100%
H2	100%
H3	100%
H4	78%

For "H4: Jeg vil endre passord" er det ukjent hvorfor enkelte av testpersonene ikke klarte å gjennomføre. Men en gjennomføringsrate på 78% er likevel ikke grunn til bekymring.

### 5.1.3.3 Skriftlige tilbakemeldinger

Samlet så det ut til å være stort sett stor enighet blant testpersonene om at webapplikasjonens utséendet falt i god smak og at det stod i god stil med forbundet. Flere av testpersonene nevnte også det var enkelt å navigere seg rundt. Brukerne hadde også mange forslag til videreutvikling av produktet. Dessverre hadde vi ikke nok tid til å implementere nye funksjoner mot slutten.

#### Sitat fra testpersoner

“Jeg likte at den var veldig enkel og oversiktig. Dette er et pluss for veldig mange, da brukere i alle aldersgrupper kan enkelt lære å bruke applikasjonen. Veldig kult å ha mulighet til å bare kommentere andre innlegg, like andre innlegg, og lage sine egne innlegg med/uten vedlegg. Terskelen for å delta i en diskusjon er lav, og tror vi få inkludert flere gjennom en slik løsning.”

“Søket fungerte dårlig. Jeg opplevde også at den ikke hoppet til toppen av siden når jeg valgte mellom kategorier og det er litt irriterende.”

“Veldig fin å se på, grafisk perfekt utformet, lett å forholde seg til og ikke mye informasjon på en gang. Savner profilbilde på personer som legger ut!”

“Jeg synes webapplikasjonen (PC og iPhone) var veldig fin og enkel å forstå. Applikasjonen står i stil med forbundet, og det kommer klart frem på forsiden.”

### 5.1.3.4 Konklusjon av brukertesting

I alt var testpersonene jevnt over svært fornøyd med webapplikasjonen, og det var lite å ta tak i. Etter hvert som svar fra testpersonene kom inn, ble også feil i webapplikasjonen rettet opp i, også de som ikke ble avdekket under brukertesting. Dersom situasjonen med Covid-19 hadde vært annerledes, ville vi ha foretrukket å utføre flere tester i et mer kontrollert miljø da dette kunne ha gitt oss flere, samt mer mangfoldige tilbakemeldinger. NBF står fritt til å videreutvikle webapplikasjonen og inkludere mer brukertesting. Som gruppe er vi fornøyd med utfallet, men vi hadde gjort grundigere testing dersom det ikke hadde vært for pandemien.

Kapittel 6

# **Brukermanual**

# Innhold

<b>Brukermanual .....</b>	<b>1</b>
<b>    6.1 Forord .....</b>	<b>3</b>
<b>    6.2 Innledning .....</b>	<b>4</b>
<b>    6.3 Brukermanual for sluttbrukere.....</b>	<b>5</b>
<b>    6.4 Brukermanual for IT-ansvarlig .....</b>	<b>17</b>

## 6.1 Forord

---

Den første delen av brukermanualen gjelder webapplikasjonen Kunnskapsportalen Badminton. Her skal du som sluttbruker av webapplikasjonen få en gjennomgang av hvordan produktet fungerer. Det er ikke forventet eller påkrevd noen forkunnskaper om produktet - det er meningen at webapplikasjonen skal være intuitiv i seg selv, men ved behov kan denne brukermanualen benyttes dersom noe i webapplikasjonen er uklart.

Den andre delen av brukermanualen er ment for IT-ansvarlig som skal konfigurere og installere systemet på deres systemer. Det vil være nødvendig for vedkommende å lese denne, så konfigureringen av webapplikasjonen blir korrekt.

## 6.2 Innledning

---

Kunnskapsportalen Badminton er et forum og kunnskapsportal i én - badmintonspillere og badmintonentusiaster kan diskutere relevante temaer i badmintonmiljøet med hverandre, og Norges Badmintonforbund kan dele kunnskap om badmintonsporten.

### 6.2.1 Problemstilling

Dagens situasjon er at NBF betaler flere titalls tusen kroner i året til en tredjepart for kun en videotjeneste. Hensikten med dette prosjektet er å gi NBF en felles plattform som vil samle all kommunikasjon og visning av videoer uten involvering av en tredjepart. Dette er ikke den eneste grunnen til at dette prosjektet ble aktuelt. Flere personer fra Norges Badmintonforbund har uttalt seg rundt hvorfor dette prosjektet ble relevant. Oppsummert er det et generelt ønske om å skape en plattform for samhandling, erfearingsutveksling og inspirasjon for badmintonsporten. Det var også et håp om å motivere flere til å melde seg på trenerkurs, blant annet fordi mindre sportsklubber ofte mangler frivillige eller i det hele tatt gode trenere, noe som fører til at man ”blir stående alene på trening uten øvelser/kompetanse til å utføre øvelser [sic]”. Foreløpig har klubbene i Norges Badmintonforbund kun frivillige i administrasjonen, og de tror at en god delingskultur kan gi et bedre utgangspunkt for at flere frivillige trenere skal bli inspirert til å lære bort kunnskapen sin.

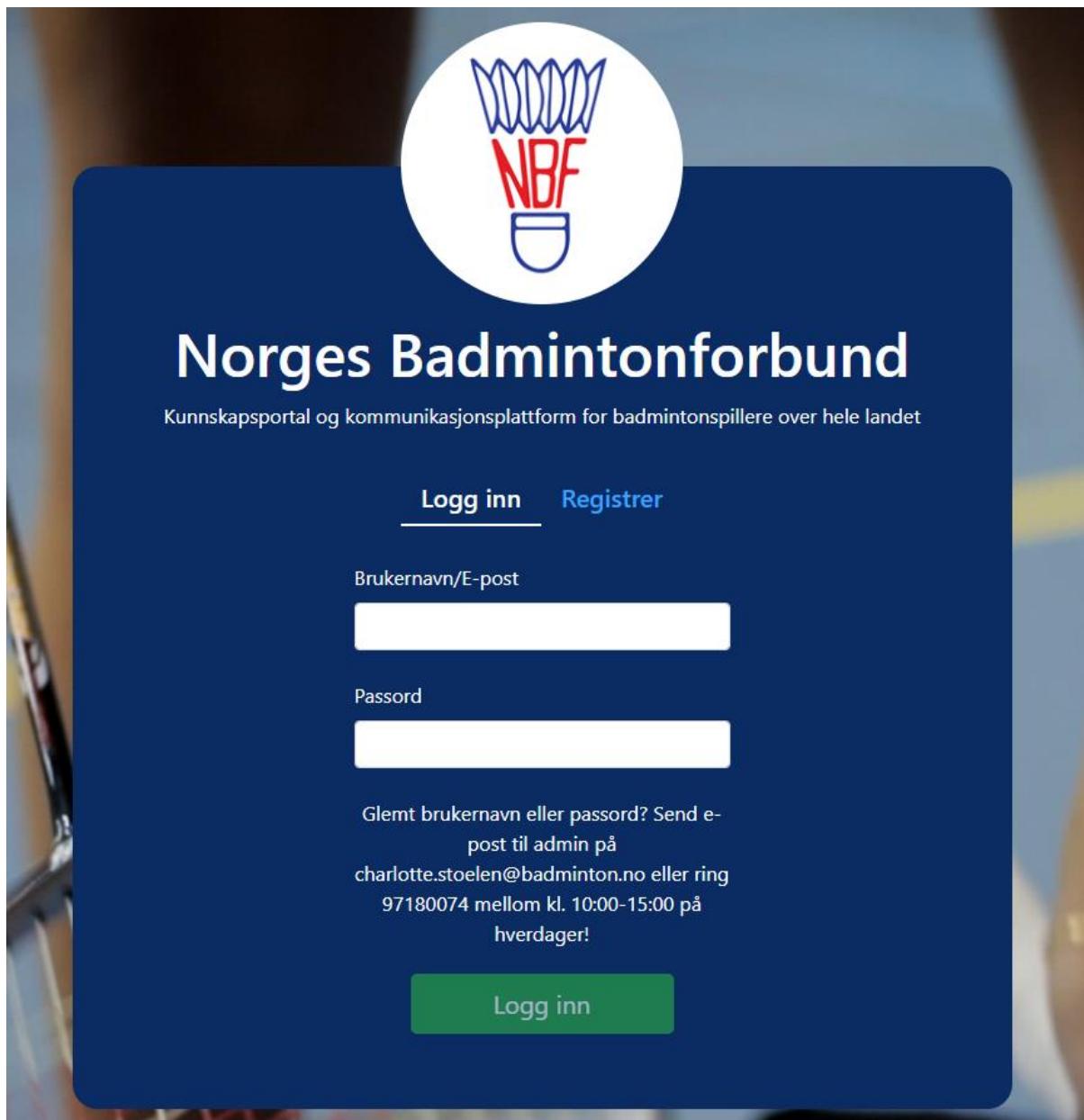
### 6.2.2 Formål

I webapplikasjonen kan Norges Badmintonforbund dele dokumenter og videoer som omhandler badmintonsporten, som blant annet riktig teknikk og utførelse av diverse øvelser. Brukere av webapplikasjonen skal kunne opprette poster i forumet der de for eksempel kan stille spørsmål relevante for badmintonsporten eller dele erfaringer med andre badmintonspillere og badmintoninteresserte.

## 6.3 Brukermanual for sluttbrukere

I denne brukermanualen vil du få en gjennomgang av hvordan du bruker webapplikasjonen, fra start til slutt. Etter du har lest denne skal du ha en grunnleggende forståelse av dens funksjoner og egenskaper.

### Innlogging og registrering



Figur 6.3.1a: Innloggingssiden

På innloggingssiden finner du to felt for e-postadresse/brukernavn og passord. For å logge inn, må du skrive inn ditt registrerte brukernavn eller e-postadresse og tilhørende passord i disse feltene. Hvis brukernavn/e-postadresse og/eller passord er feil, får du en feilmelding om dette. Hvis du har glemt innloggingsdetaljene, kan du følge instruksjonene under passordfeltet. Dersom du ikke har en bruker, trykker du på "Registrer" til høyre. Da skal du se figur 6.3.1b.



Figur 6.3.1b: Registreringssiden

På registreringssiden ser du 6 forskjellige inputfelt. Dette er for brukernavn, e-post, fornavn, etternavn, passord og bekreft passord. For å registrere en ny bruker, må du først lage et brukernavn. Dette må være minst 5 og maks 30 karakterer langt. Dersom brukernavnet overskridet disse grensene, eller noen andre har det som brukernavn, vil du få melding om

dette. Deretter fyller du ut e-postadresse, fornavn og etternavn. Etter det fyller du ut passord-feltet. Passordet må være minst 8 karakterer langt, inneholde minst én liten bokstav og én stor bokstav, samt minst ett tall. Dette er for å lage et sikkert passord. Det er viktig at du ikke gir bort ditt passord til noen. Passordet kan endres på senere ved behov. Til slutt fyller du ut bekreft passord-feltet, det må være identisk med passord-feltet. Ved å bekrefte passordet kan du være sikker på at du ikke har tastet det inn feil første gang. Dersom noen av feltene ikke oppfyller kriteriene, vil du få melding om dette, slik at du kan rette det opp, se figur 6.3.1c.

<b>Brukernavn</b>	<b>E-post</b>
Pia	pia@gmail.com
Brukernavn må være minst 5 tegn	
<b>Fornavn</b>	<b>Etternavn</b>
Pia	Karoline
<b>Passord</b>	<b>Bekreft passord</b>
*****	*****
Passord og bekreft passord er ikke like	

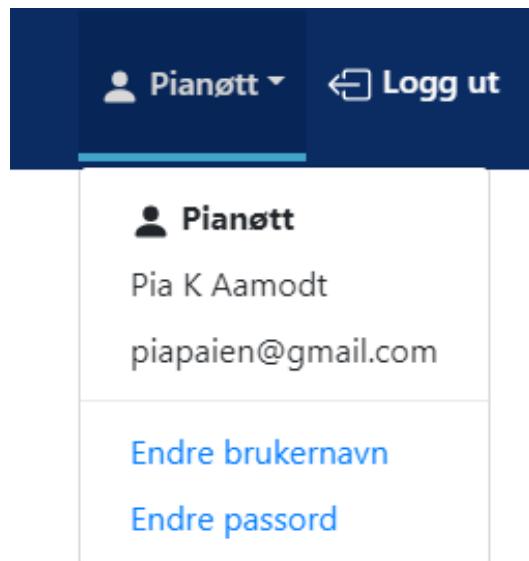
Figur 6.3.1c: Eksempel på inputvalidering

## Navigasjon



Figur 6.3.1d: Navigasjonsbar

Øverst i webapplikasjonen etter å ha logget inn, finner du navigasjonsbaren. Til venstre i denne ser du Norges Badmintonforbund sin logo. Ved siden av denne finner du knapper som tar deg med til hjem (forsiden), forumet og kunnskapsportalen. Til høyre i navigasjonsbaren kan du trykke på knappen med brukernavnet ditt. Denne åpner en nedtrekksmeny som lar deg se nøkkelinformasjonen du har oppgitt om deg selv som e-postadresse, fornavn og etternavn. Her har du også linker som gir deg mulighet til å endre brukernavn og passord. Disse vil se ganske like ut som da du først registrerte deg, se figur 6.3.1b.



Figur 6.3.1e: Nedtrekksmeny i navigasjonsbar

## Forside

### Diskusjoner i forumet

Sorter: ▾

Postet av **HansJarle** forrige mandag kl. 22:15

**Innhold i en time med skolebadminton**

Jeg har tidligere gjennomført skolebesøk, der elevene har fått prøvd seg med badminton i gymtimene. Mitt siste skolebesøk var tilbake i 2002(!) og jeg trenger derfor tips til innhø ...[Les mer](#)

Kompetanse - Skolebadminton      0 0

Postet av **CamillaStokkaune** 12.05.2021

**Jeg ønsker å bli trener**

Jeg har tidligere vært aktiv badmintonspiller, men ønsker å bidra i klubb/krets gjennom å være trener. Organiserer NBF slike kurs, og hvor ofte arrangeres de?

Kompetanse - Trenere      1 1

**Ønsker du å lære mer?**

I kunnskapsportalen finner du mange nyttige artikler og videoer som lærer deg korrekte treningssteknikker som er relevante for badmintonporten.

[Til kunnskapsportalen →](#)

Postet av **emiliestorvik** 12.05.2021

**Trening for å bli god**

Hvordan kan jeg trenere i en klubb for å nå maks potensiale? Hvordan trener landslaget i sine egne klubber? Noen som har tips?

Toppidrett - Trening      2 0

**Slå av en prat**

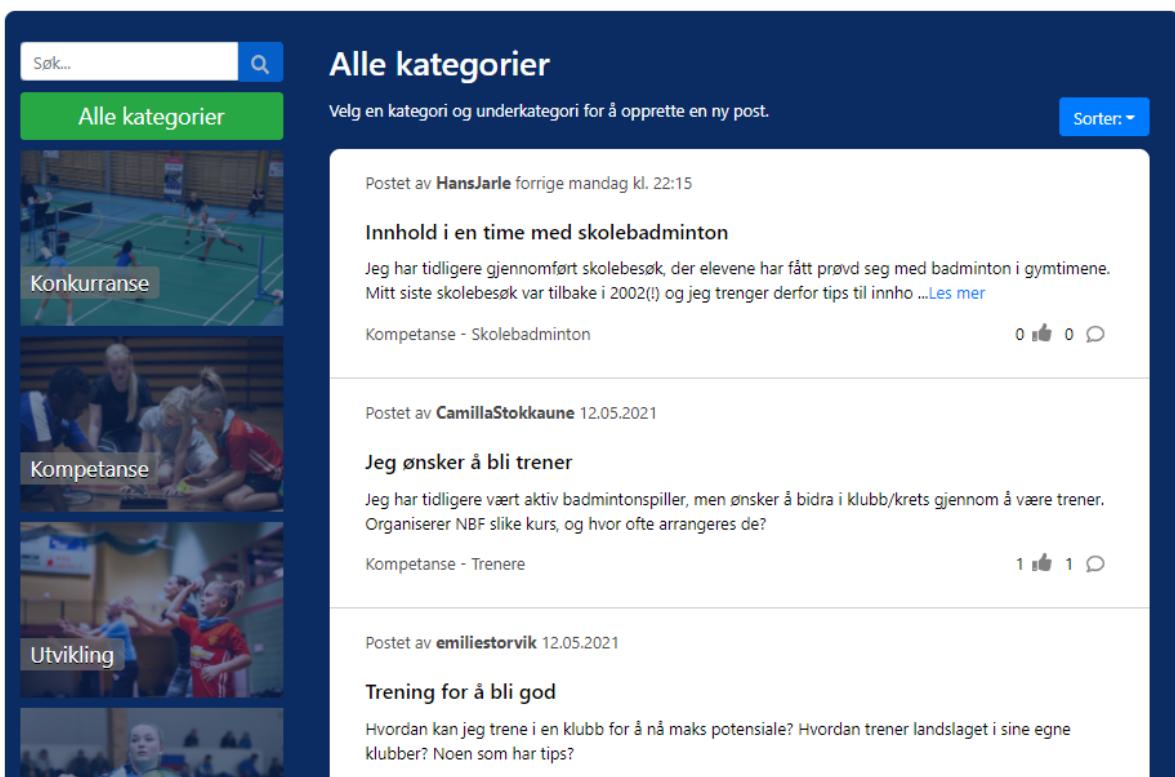
I forumet finner du mange likesinnede mennesker som ønsker å dele kunnskap, opplevelser og erfaringer fra badmintonporten.

[← Til forumet](#)

Figur 6.3.1f: Forsiden

Forsiden, som sett i figur 6.3.1f, er det første du ser etter du har registrert deg og/eller logget inn. I venstre kolonne kan du se et utdrag av diskusjoner i forumet. Disse kan du sortere med knappen til høyre over kolonnen. Klikker du på en av disse diskusjonene, går du inn på tråden for denne. Les mer om dette i neste under ”Forumpost”. I høyre kolonne finner du kort informasjon om webapplikasjonens hovedelementer, nemlig forumet og kunnskapsportalen. Klikker du på ”Til kunnskapsportalen”, vil du bli sendt videre til kunnskapsportalen. Se mer om kunnskapsportalen i ”Kunnskapsportalen”. Klikker du på ”Til forumet”, vil du bli sendt til forumet. Se mer om forumet i ”Forum”.

## Forum

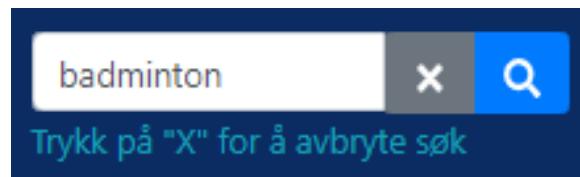


The screenshot shows the 'Alle kategorier' (All categories) page of the NBF forum. At the top, there is a search bar with the placeholder 'Søk...' and a magnifying glass icon. Below it, a green button says 'Alle kategorier'. To the right, a text box says 'Velg en kategori og underkategori for å opprette en ny post.' (Select a category and subcategory to create a new post.) and a 'Sorter:' dropdown menu. The main content area displays three forum posts:

- Innhold i en time med skolebadminton**  
Posted by HansJarle on May 22, 2021. The post content: Jeg har tidligere gjennomført skolebesøk, der elevene har fått prøvd seg med badminton i gymtimene. Mitt siste skolebesøk var tilbake i 2002(!) og jeg trenger derfor tips til innhø ... [Les mer](#).  
Category: Kompetanse - Skolebadminton. Likes: 0, Comments: 0.
- Jeg ønsker å bli trener**  
Posted by CamillaStokkaune on May 12, 2021. The post content: Jeg har tidligere vært aktiv badmintonspiller, men ønsker å bidra i klubb/krets gjennom å være trener. Organiserer NBF slike kurs, og hvor ofte arrangeres de?  
Category: Kompetanse - Trenere. Likes: 1, Comments: 1.
- Trening for å bli god**  
Posted by emiliestorvik on May 12, 2021. The post content: Hvordan kan jeg trenne i en klubb for å nå maks potensiale? Hvordan trener landslaget i sine egne klubber? Noen som har tips?  
Category: Utvikling. Likes: 0, Comments: 0.

Figur 6.3.1g: Forumet

Forumet (figur 6.3.1g) er der medlemmer av webapplikasjonen kan diskutere alt om badmintonsporten med hverandre. I venstre kolonne finner du et søkefelt. For å bruke denne, skriver du inn ditt ønskede søkeord og klikker Enter-knappen på tastaturet eller på det blå forstørrelsesglasset med musepekeren. Da vil du få opp relevante forumposter i høyre kolonne. For å avbryte søket, trykker du på den grå X-knappen som illustrert i figur 6.3.1.h. Forumpostene vil da returnere til sin opprinnelige stand slik det var før du søkte.



Figur 6.3.1h: Illustrasjon av søkefunksjon

Under søkefeltet ser du en grønn knapp der det står "Alle kategorier". Klikker du på denne, vil du se forumpostene i alle kategoriene som finnes. Hvis du vil se på en spesifikk kategori, klikker du på et av bildene under den grønne knappen. Du vil da få en rekke underkategorier å velge mellom. La oss ta utgangspunkt i at du klikker på "Kompetanse". Her vil du få velge mellom de underliggende kategoriene "Trening bredde", "Trenere" og "Skolebadminton".



Figur 6.3.1i: Illustrasjon av kategorier og underkategorier

Klikker du på én av underkategoriene, vil du se alle forumpostene som finnes i denne kategorien. I figur 6.3.1j er kategorien "Kompetanse" og dens underkategori "Skolebadminton" valgt, og posten som finnes i kategorien vises. Øverst i posten ser du hvem som er forfatteren og når den ble postet. Deretter får du postens tittel og innhold. Er posten lang, vil den bli kortet ned for å spare skjermpllass, men du kan lese mer av innholdet ved å trykke "Les mer". Nederst til venstre i posten ser du hvilken kategori og underkategori den er postet i. Til høyre i posten ser du antall "likes" (tomler opp) og antall kommentarer (snakkeboble) den har fått. Dersom du har "liket" posten, vil tommelen være blå.

## Kompetanse - Skolebadminton

Her kan du slå av en prat om skolebadminton

+ Ny post

Sorter: ▾

Postet av **HansJarle** forrige mandag kl. 22:15

### Innhold i en time med skolebadminton

Jeg har tidligere gjennomført skolebesøk, der elevene har fått prøvd seg med badminton i gymtimene.  
Mitt siste skolebesøk var tilbake i 2002(!) og jeg trenger derfor tips til innhø ...[Les mer](#)

Kompetanse - Skolebadminton

0 0

10 per side: ▾

*Figur 6.3.1j: Illustrasjon av sortering med kategorier og underkategorier*

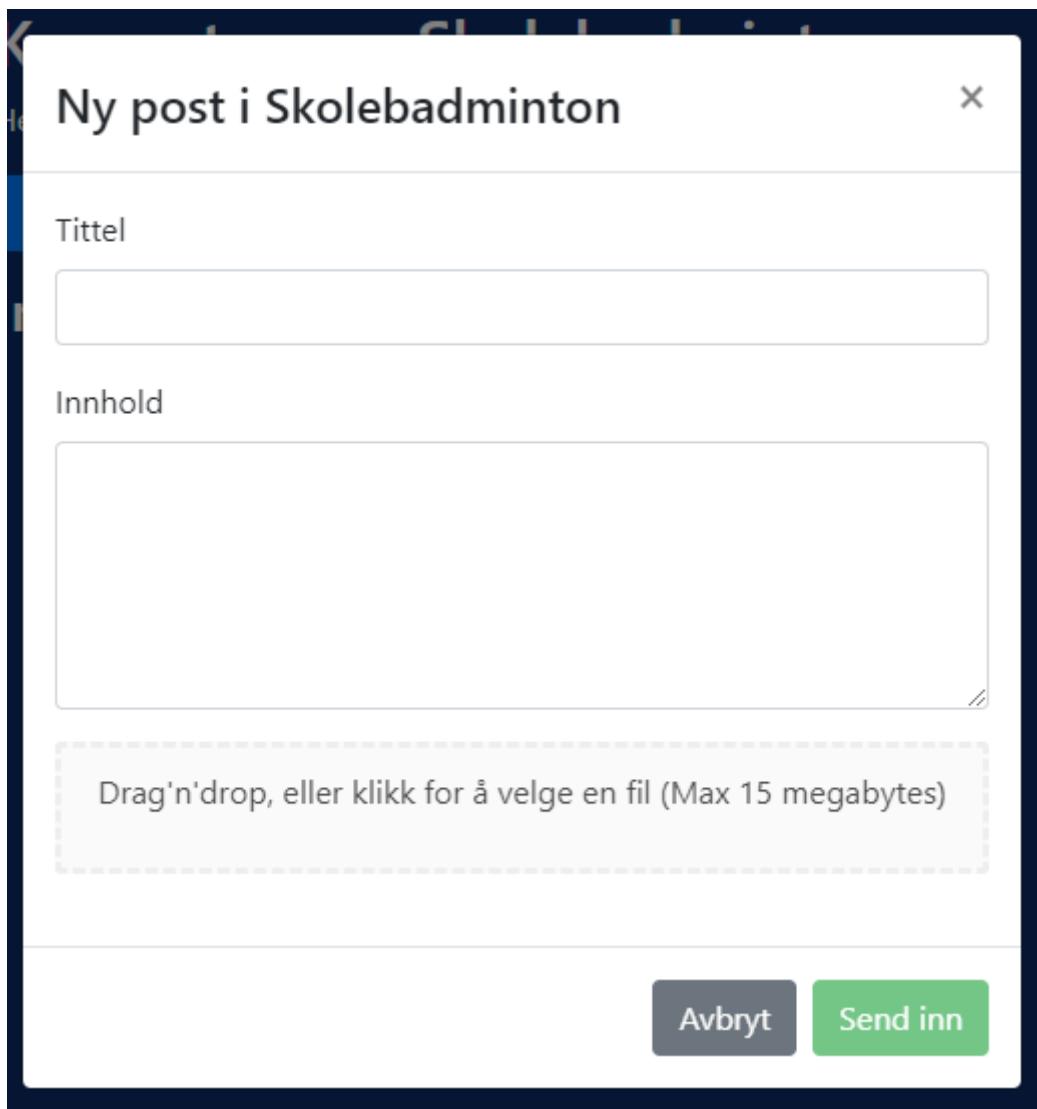
Har du valgt en kategori og underkategori, vil du få muligheten til å opprette en post ved å klikke på ”+ Ny post”-knappen til venstre, se mer informasjon om dette i neste avsnitt. Til høyre kan du sortere postene som vises etter ønske. Dersom det er mange poster i forumet, vil det etter hvert bli mange sider i forumet. Dette skal vises nede i venstre hjørne i figur 6.3.1j, men er ikke synlig fordi det ikke er nok poster i eksempelet. Figur 6.3.1k viser hvordan denne sidetallsfunksjonen ser ut. Nederst i høyre hjørne kan du velge hvor mange poster du vil se på én side. Hadde det vært 100 poster i forumet, ville du hatt muligheten til å se disse på én side istedenfor å navigere mellom disse med sidetallsfunksjonen.



*Figur 6.3.1k: Sidetallsfunksjon*

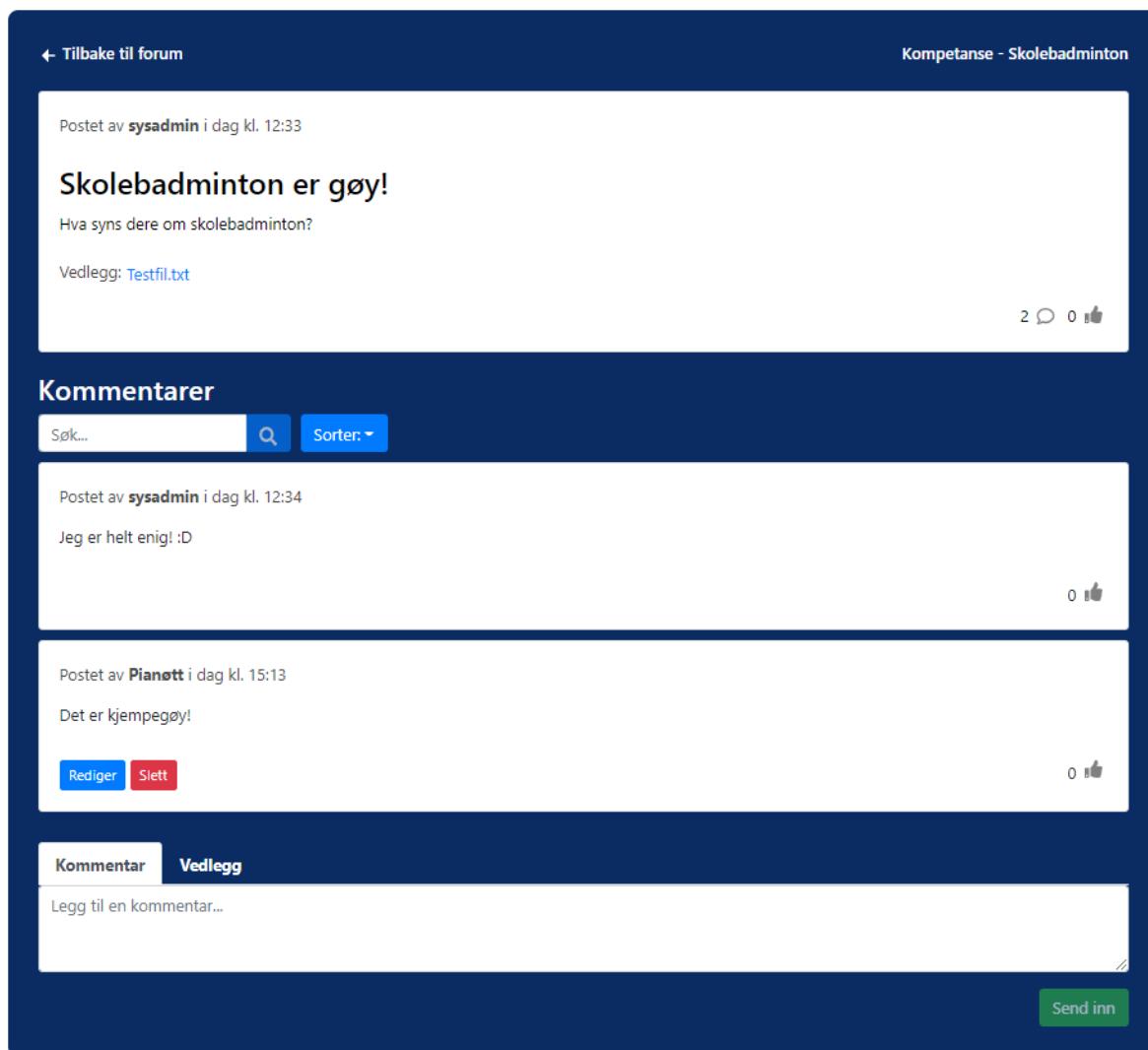
For å lage en ny post, klikker du som nevnt tidligere på ”+ Ny post”-knappen. Denne vises kun når du har valgt en kategori og tilhørende underkategori. Etter å ha trykket på denne, vil et nytt vindu dukke opp på skjermen, se figur 6.3.1l. Her kan du definere en tittel for posten, skrive innholdet i posten, og ved ønske kan du også legge til en fil. Du kan kun legge til filer av typen .txt, .pdf, .doc, .docx, .xls og .xlsx. For å avbryte, kan du trykke på ”Avbryt”-knappen. Etter å ha fylt ut tittel, innhold, og eventuelt lagt ved en fil, kan du trykke på ”Send inn”-

knappen. Du har da laget en post, og andre brukere kan kommentere og ”like” den. Etter å ha laget posten, vil du bli sendt direkte til den. Se mer om dette i neste avsnitt.



Figur 6.3.1l: Opprettelse av ny post

## Forumpost



Postet av **sysadmin** i dag kl. 12:33

**Skolebadminton er gøy!**

Hva syns dere om skolebadminton?

Vedlegg: Testfil.txt

2 0

Postet av **sysadmin** i dag kl. 12:34

Jeg er helt enig! :D

0

Postet av **Pianøtt** i dag kl. 15:13

Det er kjempegøy!

Rediger Slett 0

**Kommentarer**

Søk... Sorter: ▾

**Kommentar** **Vedlegg**

Legg til en kommentar...

Send inn

Figur 6.3.1m: Forumpost

Figur 6.3.1m viser et eksempel på en forumpost. På samme måte som i forumet, kan man se informasjon om hvem som har skrevet den og når, tittel, innhold og antall kommentarer og "likes". Ved å klikke på tommelen til høyre, kan du "like" posten. I tillegg er det vedlagt en fil i dette eksemplet. Ved å klikke på filnavnet, vil denne bli lastet ned til din enhet og du kan åpne den og se innholdet. Dersom du er forfatteren av posten, vil du få mulighet til å redigere eller slette posten. Redigerer du posten, vil du få opp et vindu nesten identisk med det som når du oppretter en ny post. Sletter du posten, vil du få opp et nytt vindu der du må bekrefte at du vil slette den, se figur 6.3.1n. Dette er for å unngå at du sletter den ved et uhell.



*Figur 6.3.1n: Sletting av post*

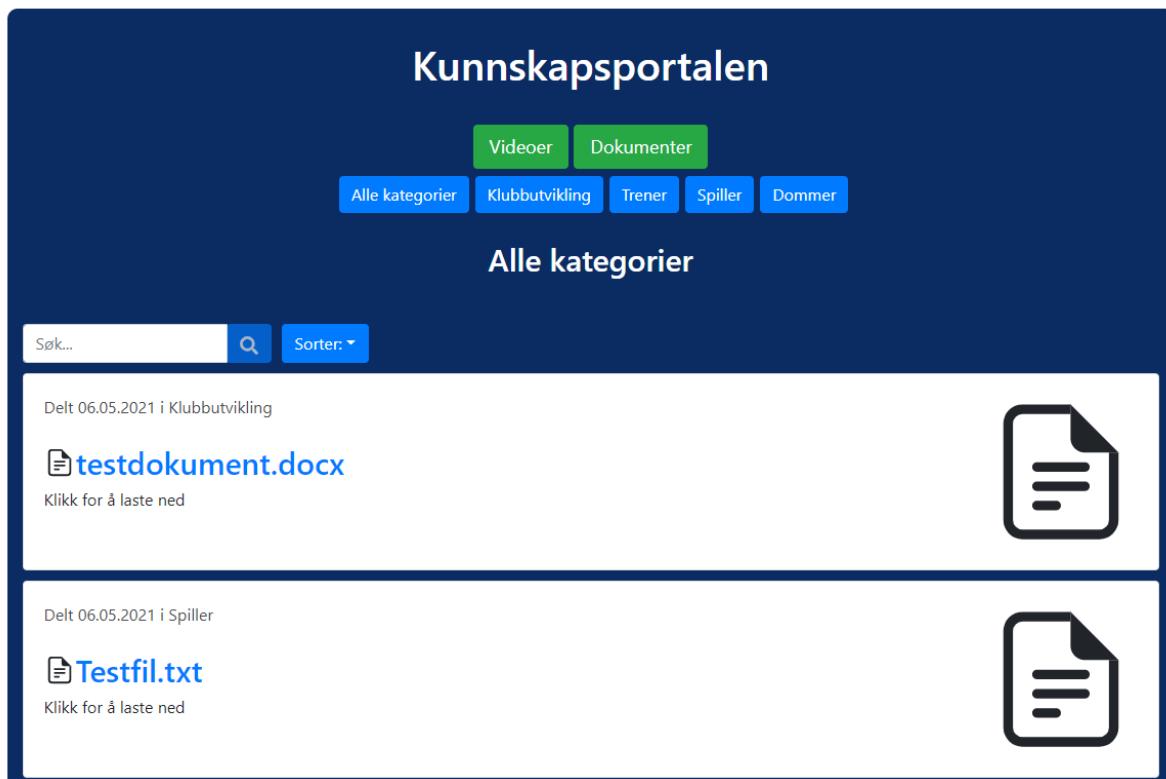
Kommentarer er i nesten likt format som en post. Kommentarer har ingen tittel, kun innhold. Dersom du er forfatteren av en kommentar, kan du også redigere og slette en kommentar på samme måte som om du var forfatteren av en post. Du kan også søke, sortere og navigere mellom flere sider i kommentarene, dersom det er mange nok, på lik linje som i forumet.

Vil du skrive en kommentar til en post, kan du gjøre det i feltet nederst i figur 6.3.1m. I nye kommentarer kan du også inkludere vedlagte filer ved å trykke på "Vedlegg"-knappen. Denne fungerer på samme måte som når man lager en post. Når du har skrevet en kommentar, og eventuelt lagt ved en fil, kan du trykke på "Send inn"-knappen nederst i figur 6.3.1o. Din kommentar vil da være synlig for andre som ser posten, og på denne måten kan du kommunisere med andre i posten.



*Figur 6.3.1o: Vedlegg til kommentar*

## Kunnskapsportalen



The screenshot shows the homepage of the Kunnskapsportalen. At the top, there is a dark blue header with the title "Kunnskapsportalen". Below the title, there are two green buttons labeled "Videoer" and "Dokumenter". Underneath these are five blue buttons labeled "Alle kategorier", "Klubbutvikling", "Trener", "Spiller", and "Dommer". A large heading "Alle kategorier" is centered below the buttons. Below this, there is a search bar with a placeholder "Søk..." and a magnifying glass icon. To the right of the search bar is a dropdown menu labeled "Sorter: ▾". The main content area displays two items. The first item is titled "Delt 06.05.2021 i Klubbutvikling" and has a file icon followed by the text "testdokument.docx". Below the title is the instruction "Klikk for å laste ned". To the right of this item is a document icon. The second item is titled "Delt 06.05.2021 i Spiller" and has a file icon followed by the text "Testfil.txt". Below the title is the instruction "Klikk for å laste ned". To the right of this item is a document icon.

Figur 6.3.1p: Kunnskapsportalen

Kunnskapsportalen er der administrator kan legge ut videoer og dokumenter som brukerne av webapplikasjonen kan dra nytte av. Kunnskapsportalens kategorier er bygget opp på en lignende måte som i forumet. Her er det to hovedkategorier, videoer og dokumenter, med 5 forskjellige underkategorier - "Klubbutvikling", "Trener", "Spiller" og "Dommer", eller man kan se alle kategoriene samtidig. Disse underkategoriene gjelder for både videoer og dokumenter. Man kan søke gjennom, sortere og navigere mellom sidene for videoer og dokumenter i kunnskapsportalen, på samme måte som i forumet og i en post.

Både videoer og dokumenter vises i et format lignende en post. Her vises datoen først og hvilken kategori den er postet i. Deretter vises tittelen. For dokumenter er dette navnet på filen, mens for videoer er dette tittelen administrator har angitt. Under tittelen er en kort instruksjon for hvordan man kan se på videoen eller hvordan man kan laste ned et dokument.

Delt 07.05.2021 i Trener

## Backhandgrep

Klikk for å se video



*Figur 6.3.1q: Eksempel for video*

For å laste ned et dokument, trenger man bare å klikke på dokumentnavnet, og det lastes ned til din enhet, akkurat som i en forumpost. For å se en video må man enten klikke på bildet eller tittelen. Da dukker det opp et vindu med videoen og en beskrivende tekst, se figur 6.3.1r. For å se på selve videoen klikker man på pilen midt i bildet. Under den beskrivende teksten finner du linken "Diskuter i forumet". Klikker du på denne linken, havner du i forumposten for denne spesifikke videoen, slik at du kan diskutere den med andre brukere. For å lukke vinduet, klikk på krysset øverst i høyre hjørne eller den grå "Lukk"-knappen nederst i høyre hjørne.



*Figur 6.3.1r: Vindu for visning av video*

## 6.4 Brukermanual for IT-ansvarlig

---

Denne webapplikasjonen består av to deler, et frontend-system og et backend-system. Begge disse systemene må konfigureres hver for seg. Etter ønske fra oppdragsgiver er systemene utviklet med publisering på Azure som et mål. Vi tar ikke for oss selve publiseringen til Azure, da det er mange måter å gjøre dette på, og IT-ansvarlig hos oppdragsgiver har egnet kompetanse på dette området.

### 6.4.1 Krav til tjenester på Azure Portal

For å fungere som tiltenkt, har denne webapplikasjonen noen krav til tjenester som er tilgjengelig via Azure Portal. Disse tjenestene omfatter en Azure SQL server til databasen, en Storage account til opplastede dokumenter og App Services for publisering av systemene. Ønskes en samlet oversikt over alle tjenestene som brukes av webapplikasjonen, anbefales det å opprette en egen Resource group for alle disse tjenestene.

### 6.4.2 Azure SQL server

Har oppdragsgiver en eksisterende Azure SQL server, kan denne benyttes. For å lage en ny server, åpnes SQL servers fra forsiden på Azure Portal. Når det opprettes en ny server settes brukernavn og passord her. Dette skal vi bruke senere under konfigurering av backend-systemet. Under SQL servers må det legges til en ny database. Det kan gjøres ved å gå inn på serveren som ble opprettet og trykke på «Create database». Når denne er lagt til, kan vi gå inn på den nye databasen og trykke på «Connection strings» under «Settings» i menyen på venstre side. I det første vinduet (ADO.NET), kan vi se denne strengen. Den skal vi også bruke til konfigurering av backend-systemet.



SQL servers

### 6.4.3 Storage account

Webapplikasjonen trenger en egen konto for lagring av alle dokumenter som lastes opp. For å lage en ny konto, åpnes Storage accounts fra forsiden på Azure Portal. Når vi har åpnet Storage accounts, er det bare å trykke på «Add» for å lage en ny konto. Når denne er laget, kan vi åpne den nye kontoen og velge «Access keys» under «Security + networking» i



Storage accounts

menyen på venstre side. Her kan vi se «Connection string» under «key1». Denne skal vi bruke til konfigurering av backend-systemet.

Alle filer som lastes opp i webapplikasjonen vil være tilgjengelig under “Containers” på kontoen som opprettes her. Dette gir tilgang til å finne igjen alle dokumenter om systemet skulle bli satt ut av drift. Det er viktig at ingen dokumenter slettes direkte i Azure Portal, da backend-systemet også benytter seg av en oppføring i databasen for å knytte dokumenter til poster, kommentarer og brukere. Strukturen på dokumenter under “Containers” er Dato -> Brukernavn -> Filnavn.

#### 6.4.4 App Services

Da vi publiserte en demo for oppdragsgiver, hadde vi en felles App Service plan for frontend- og backend-systemet. Denne opprettet vi med Windows som operativsystem, og F1 price tiers under «Dev/Test». Her bestemmer oppdragsgiver selv hva de ønsker å bruke, og systemressurser som vil være tilgjengelig for webapplikasjonen kan skaleres her. For publisering av systemet som skal brukes av sluttbrukere anbefales det en S1 price tier under «Production» for en sømløs brukeropplevelse. Dette er det første alternativet som er tilgjengelig her. Etterhvert som systemet vokser og antall brukere eskalerer, kan det bli nødvendig å endre dette senere.

Frontend- og backend-systemet skal opprettes som hver sin App Service. Backend-systemet skal publiseres først. Når dette er konfigurert (følg kapittel 6.4.5) og publisert, skal URL-linken dette har blitt publisert til benyttes under konfigurasjonen av frontend-systemet (følg kapittel 6.4.6). Når begge systemene er publisert, vil det være nødvendig å oppdatere innstillinger for CORS under backend-systemet sin App Service. Dette ligger under «API» og «CORS» i menyen på venstre side når tilhørende App Service åpnes i Azure Portal. Bruk URL-linken til frontend-systemet eller \* for å tillate alle. Husk å trykke på «Save».

Webapplikasjonen skal nå være klar til bruk, og åpnes med URL-linken til frontend-systemet.

#### 6.4.5 Konfigurering av backend-systemet

Alle nødvendige innstillinger som må settes i dette systemet skal gjøres i filen appsettings.json. Denne filen ligger under mappen API i Backend mappen. Filen kan åpnes med notepad. Eventuelt kan hele prosjektet åpnes i Visual Studio. Da skal filen webforum.sln som ligger i roten til backend-systemet åpnes. Skjermbildene vi har brukt her er fra Visual Studio. Figur 6.4.5a viser skjermbilde av hele appsettings.json filen.

```
{
  "ConnectionStrings": {
    "AzureDatabase": "Server", // Connection string for databasen skal brukes her
    "AzureStorageKey": "DefaultEndpointsProtocol" // Connection string for Storage account skal brukes her
  },
  "AuthSettings": {
    "Secret": "Skriv inn en unik nøkkel/hemmelighet som benyttes til generering av tokens her.",
    "TokenTimeout": 8 // Antall timer før bruker må logge inn på nytt
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

Figur 6.4.5a: appsettings.json

Det er tre ting som skal konfigureres før systemet publiseres. Dette er AzureDatabase, AzureStorageKey og AuthSettings.

### AzureDatabase

Dette er innstillingene for databasen. Da det ble opprettet en ny database i Azure Portal (kapittel 6.4.2), så vi også på hvordan vi kunne hente informasjon om «Connection strings». Den skal kopieres og limes inn på linjen vi ser i figur 6.4.5b.

`"AzureDatabase": "Server", // Connection string for databasen skal brukes her`

Figur 6.4.5b: Innstilling for databasen

«Server» skal da byttes ut med strengen vi har kopiert. Brukernavn og passord skal også fylles ut i denne strengen. En typisk streng kan se ut som dette:

`<Server=tcp:webforum.database.windows.net,1433;Initial Catalog=webforum;Persist Security Info=False;User`

`ID=BRUKERNAV;Password=PASSORD;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;>` hvor BRUKERNAV og PASSORD må endres til deres brukernavn og passord.

### AzureStorageKey

Dette er innstillingen for Storage account hvor dokumenter lastes opp. Når denne kontoen ble opprettet (kapittel 6.4.3), så vi også på hvordan vi kunne hente ut informasjon om «Connection string». Den skal kopieres og limes inn på linjen vi ser på figur 6.4.5c.

```
"AzureStorageKey": "DefaultEndpointsProtocol" // Connection string for Storage account skal brukes her
```

*Figur 6.4.5c: innstilling for Storage account*

«DefaultEndpointsProtocol» skal da byttes ut med strengen vi har kopiert. Det er ingen brukernavn eller passord som skal legges inn her.

### AuthSettings

Dette er innstillinger for JSON Web Tokens. Som vi kan se på figur 6.4.5d, er det to ting som kan endres her.

```
"AuthSettings": {  
    "Secret": "Skriv inn en unik nøkkel/hemmelighet som benyttes til generering av tokens her.",  
    "TokenTimeout": 8 // Antall timer før bruker må logge inn på nytt  
},
```

*Figur 6.4.5d: Innstillinger for JSON Web Tokens*

«Secret» er en streng som blir brukt når tokens generes i systemet, og denne skal holdes hemmelig. Strengen kan bestå av hva som helst, men for økt sikkerhet kan det være lurt å generere en tilfeldig streng som vil være umulig for utenforstående å gjette seg frem til.  
 «TokenTimeout» er antall timer en bruker kan være logget inn på systemet før sikkerhetsnøkkelen utløper. Når denne nøkkelen utløper blir brukeren nødt til å logge inn på nytt med brukernavn og passord.

Når disse innstillingene er lagt inn, må endringene lagres, og backend-systemet er nå klart til publisering i Azure Portal under App Services. I figur 6.4.5e har vi lagt ved et eksempel på hvordan appsettings.json filen kan se ut når alle innstillingene er lagt inn.

```
{  
    "ConnectionStrings": {  
        "AzureDatabase": "Server=tcp:webforum.database.windows.net,1433;Initial Catalog=webforum;Persist Security Info=False;  
        "AzureStorageKey": "DefaultEndpointsProtocol=https;AccountName=webforum;AccountKey=Q+MdjlKXXWm3IvEY3QmGkAvpsHrIuP3C1  
    },  
    "AuthSettings": {  
        "Secret": "51Hd9Bsh2ZDa7T3W564UP8ut4BeeD2ZFMXJ6nThUueVSPizsGdlgQ3eNMnsNbzsEMOJYvtK2yuCyx0zFqn8ryam1rx9cZvM2Dd",  
        "TokenTimeout": 8 // Antall timer før bruker må logge inn på nytt  
    },  
    "Logging": {  
        "LogLevel": {  
            "Default": "Information",  
            "Microsoft": "Warning",  
            "Microsoft.Hosting.Lifetime": "Information"  
        }  
    },  
    "AllowedHosts": "*"  
}
```

*Figur 6.4.5e: appsettings.json (merk at ikke hele teksten til Database og Storage har plass)*

#### 6.4.6 Konfigurering av frontend-systemet

Det er kun én ting som må gjøres i frontend-systemet før publisering, og dette er å lime inn URL-linken til backend-systemet i Apps.js filen. Denne filen ligger under mappen src i Frontend-mappen. Skjermbildet på figur 6.4.6a viser feltet som skal endres i denne filen.

```
// Her må linken til backend-systemet limes inn.  
export const host = "https://example.com/";
```

Figur 6.4.6a: App.js

Her skal https://example.com/ byttes ut med URL-linken til backend-systemet. Merk at denne linken må inneholde "/" på slutten. Endringen må lagres, og frontend-systemet er nå klart til publisering i Azure Portal.

#### 6.4.7 Ta i bruk webapplikasjonen

Når begge systemene er publisert skal webapplikasjonen være klar til bruk. For å åpne applikasjonen, skal URL-linken til frontend-systemet benyttes, og denne linken kan derfor betraktes som «nettsiden» en sluttbruker vil forholde seg til.

Det ligger en admin-bruker i systemet som kan brukes for å gi andre brukere adminrettigheter. Denne brukeren er opprettet med følgende:

Brukernavn: sysadmin

Passord: password

Merk at dette passordet kan og bør endres i systemet (se kapittel 4.2.1.3).

# Kapittel 7

# Referanser

Logoer i kapittel 2.1.3 og 6.4 er alle hentet fra Google Bilder.

Agile Alliance. (u.å.). *What is Scrum?* Hentet 21.04.2021 fra  
<https://www.agilealliance.org/glossary/scrum/>

Altvater, A. (2017). *What is N-Tier Architecture? How It Works, Examples, Tutorials, and More.* Hentet 12. mai 2021 fra <https://stackify.com/n-tier-architecture/>

Anderson, R., Dykstra, T., Smith, S. (2019). *App startup in ASP.NET Core.* Hentet 22. mai 2021 fra <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/startup?view=aspnetcore-5.0>

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D. (2001). *Agile Manifesto.* Hentet 23. april fra <https://agilemanifesto.org/>

Bekk Teknologiradar. (u.å.). *JWT (JSON Web Token).* Hentet 21. mai 2021 fra <https://radar.bekk.no/tech2017/arkitektur-og-plattform/jwt-json-web-token>

Booch G., Jacobson I., Rumbaugh J. (2005) (2. utgave), Addison Wesley, *The Unified Modeling Language User Guide.*

Entity Framework Tutorial. (u.å.). *Entity Framework Core: DbContext.* Hentet 23. mai 2021 fra <https://www.entityframeworktutorial.net/efcore/entity-framework-core-dbcontext.aspx>

Entity Framework Tutorial. (u.å.). *What is Code-First?.* Hentet 20. april 2021 fra <https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx>

Guru99. (u.å.). *MVC Tutorial for Beginners: What is, Architecture & Example*. Hentet 17.mai 2021 fra <https://www.guru99.com/mvc-tutorial.html>

Guru99. (u.å.). *UNIT TESTING in Asp.Net: Complete Tutorial*. Hentet 13. mai 2021 fra <https://www.guru99.com/asp-net-unit-testing-project.html>

Jones, J., Waddell, S. (2019). *The Cascading Costs of Waterfall* [Illustrasjon]. Medium.com. Hentet 23. april 2021 fra <https://medium.com/@joneswaddell/the-cascading-costs-of-waterfall-5c3b1b8beaec>

Kanjilal, Joydip. (2020). *How to use Data Transfer Objects in ASP.NET Core 3.1*. Hentet 12. mai 2021 fra <https://www.infoworld.com/article/3562271/how-to-use-data-transfer-objects-in-aspnet-core-31.html>

Kristoffersen, B. (2009). *Databasesystemer* (2. utg.). Universitetsforlaget.

Lindsjørn, Y. (2020). *Systemutvikling – Fra krav til modellering av objekter*

Lindsjørn, Y. (2020). *Systemutvikling - Modellering av krav*

Microsoft. (2020). *Common web application architectures*. Hentet 12. mai 2021 fra <https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures>

Microsoft. (2020, 21. juli). *Dependency injection in ASP.NET Core*. Hentet fra <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-5.0>

Node.js. (2011). *What is npm?* nodejs.org. Hentet 21. mai 2021 fra <https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/>

Oslo Metropolitan University. (2016). *Emneplan for DATS1500 Databaser*. OsloMet. Hentet 07. mai 2021 fra <https://student.oslomet.no/studier-/studieinfo/emne/DATS1500/2016/H%C3%98ST> Oslo

Metropolitan University. (2020a). *Emneplan for DAFE1200 Webutvikling og inkluderende design*. OsloMet. Hentet 07. mai 2021 fra <https://student.oslomet.no/studier-/studieinfo/emne/DAFE1200/2020/H%C3%98ST>

Oslo Metropolitan University. (2020b). *Emneplan for DAFE2200 Systemutvikling*. OsloMet. Hentet 07. mai 2021 fra <https://student.oslomet.no/studier-/studieinfo/emne/DAFE2200/2020/H%C3%98ST>

Oslo Metropolitan University. (2020c). *Emneplan for DATA1700 Webprogrammering*. OsloMet. Hentet 07. mai 2021 fra <https://student.oslomet.no/studier-/studieinfo/emne/DATA1700/2020/H%C3%98ST>

Oslo Metropolitan University. (2020d). *Emneplan for ITPE3200 Webapplikasjoner*. OsloMet. Hentet 22. mai 2021 fra <https://student.oslomet.no/studier-/studieinfo/emne/ITPE3200/2020/H%C3%98ST>

Rehkopf, Max. (u.å.). *What is a Kanban Board?* Atlassian. Hentet 20. april 2021 fra <https://www.atlassian.com/agile/kanban/boards>

REST API Tutorial. (u.å.). *REST Architectural Constraints*. Hentet 17.mai 2021 fra <https://restfulapi.net/rest-architectural-constraints/>

Schadler, Patrick. (2020). *C# Unit Tests with Mocks*. Hentet 13. mai 2021 fra <https://dev.to/patzistar/c-unit-tests-with-mocks-8i7>

TekTutorialsHub. (u.å.). *Dependency Injection Lifetime: Transient, Singleton & Scoped*. Hentet 17.mai 2021 fra <https://www.tektutorialshub.com/asp-net-core/asp-net-core-dependency-injection-lifetime/>

Telerik JustMock. (u.å.) *Arrange Act Assert*. Hentet 13. mai 2021 fra <https://docs.telerik.com/devtools/justmock/basic-usage/arrange-act-assert>

Tutorialspoint. (u.å.). *ASP.NET MVC - Data Annotations*. Hentet 20. april 2021 fra [https://www.tutorialspoint.com/asp.net\\_mvc/asp.net\\_mvc\\_data\\_annotations.htm](https://www.tutorialspoint.com/asp.net_mvc/asp.net_mvc_data_annotations.htm)

Tutorialspoint. (u.å.). *Entity Framework - Code First Approach*. Hentet 20. april 2021 fra  
[https://www.tutorialspoint.com/entity\\_framework/entity\\_framework\\_code\\_first\\_approach.htm](https://www.tutorialspoint.com/entity_framework/entity_framework_code_first_approach.htm)

Tutorialspoint. (u.å.). *Entity Framework - Fluent API*. Hentet 23. mai 2021 fra  
[https://www.tutorialspoint.com/entity\\_framework/entity\\_framework\\_fluent\\_api.htm](https://www.tutorialspoint.com/entity_framework/entity_framework_fluent_api.htm)

TutorialsTeacher. (u.å.). *ASP.NET Core - Program.cs*. Hentet 17.mai 2021 fra  
<https://www.tutorialsteacher.com/core/aspnet-core-program>

TutorialsTeacher. (u.å.). *Inversion of Control*. Hentet 17.mai 2021 fra  
<https://www.tutorialsteacher.com/ioc/inversion-of-control>

Ukjent forfatter. (2021). *What is Scrum? [Illustrasjon]*. Scrum.org. Hentet 20. april 2021 fra  
<https://www.scrum.org/resources/what-is-scrum>

Ukjent forfatter. (u.å.) *What is Kanban? [Illustrasjon]*. Digite. Hentet 21. april 2021 fra  
<https://www.digite.com/kanban/what-is-kanban/>

Ukjent forfatter. (2020). *Typical application layers [Illustrasjon]*. Microsoft.  
<https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures>

w3schools. (u.å.). *Responsive Web Design - Media Queries*. Hentet 30. april 2021 fra  
[https://www.w3schools.com/css/css\\_rwd\\_mediaqueries.asp](https://www.w3schools.com/css/css_rwd_mediaqueries.asp)

WebAIM. (u.å.). *WebAIM's WCAG 2 Checklist*. Hentet 14. mai 2021 fra  
<https://webaim.org/standards/wcag/checklist>

WebTrainingRoom. (u.å.). *Program.cs in ASP.NET Core*. Hentet 17.mai 2021 fra  
<https://www.webtrainingroom.com/aspnetcore/program>

# Kapittel 8

# Ordliste

## **API**

Et API (Application Program Interface (Applikasjon-program-grensesnitt)) er et grensesnitt som definerer samhandlinger mellom to eller flere applikasjoner. I dette systemet, og i webutvikling generelt, vil det i hovedsak si mellom frontend og backend.

## **Azure**

Azure er en skytjeneste av Microsoft.

## **Back end**

Applikasjonslaget brukeren/brukerne ikke har direkte tilgang på. Back end er typisk brukt til lagring og manipulering av data.

## **Brukergrensesnitt**

Et brukergrensesnitt er det en sluttbruker av et system ser og interagerer med.

## **C#**

C# er Microsoft sitt eget kodespråk som er basert på Java og C++. Dette kodespråket er en del av Microsoft sin .NET-platform, som er rammeverket vi har brukt i backend-systemet.

## **DOM (Document Object Model)**

Document Object Model er programmeringsinterfacet for HTML og XML-dokumenter på internett (dvs. ulike nettsider), og inneholder datarepresentasjonen av objektene som inneholder strukturen og innholdet til et disse dokumentene.

## **ER-modell**

ER-model (kort for entity–relationship model) er en modell som beskriver strukturen til en database ved å vise innhold, oppbygging og relasjoner mellom entiteter og deres attributter.

## Front end

Den delen av et nettsted som brukere kommuniserer direkte med, kalles frontend.

## Git

Git er en type versjonskontrollsysteem som gjør det lettere å holde rede på endringer i filer.

## JSX

React er et JavaScript-bibliotek for å bygge brukergrensesnitt. Men i stedet for å bruke vanlig JavaScript, bør React-kode skrives i noe som heter JSX og koden ser ut som HTML, og den bruker også en JavaScript-lignende variabel, men er verken HTML eller JavaScript. JSX er i en syntaks utvidelse (syntax extension) av vanlig JavaScript og brukes til å lage React-elementer.

## NuGet-pakker

NuGet-pakker er nedlastbare pakker for Visual Studio med løsninger basert på åpen kildekode. Med disse pakkene kan utviklere enkelt dele sin egen kode, og ta i bruk kode som er delt av andre utviklere.

## Prototype

Prototyping er en eksperimentell prosess der designteam implementerer ideer i håndgripelige former fra papir til digital.

## React

React er et JavaScript-bibliotek (library) oppfunnet av Facebook. React er kun et bibliotek som tar seg av tilstandskontroll (state management) og rendring, og ikke et fullt rammeverk (framework), så man er avhengig av andre rammeverk eller ren JavaScript for resten av funksjonalitetene.

## ReactDOM

ReactDOM er en pakke som gir DOM-spesifikke metoder som kan brukes på toppnivået i en webapplikasjon for å muliggjøre en effektiv måte å administrere DOM-elementer på websiden. ReactDOM gir utviklerne en API som inneholder følgende metoder og noen flere.  
render()  
findDOMNode()

## **React Hook**

Hooks er den nye funksjonen som ble introdusert i React 16.8-versjonen. State og andre React-funksjoner brukes uten å skrive en klasse.

## **Regex**

Regex står for regular expressions og omtaler mønsteret definert for spesifiserte tekststrenger. Det er vanlig at for eksempel passord følger dette da det er påkrevd med en viss mengde og type tegn.

## **Scrum**

Scrum er en smidig programvareutviklingsmetode som bruker en fleksibel strategi for å utvikle, levere og opprettholde komplekse produkter.

## **Skytjeneste (Cloud computing service)**

Skytjenester er en samlebetegnelse på ressurser og programvare som er tilgjengelig på eksterne servere. En typisk skytjeneste vi har brukt i bacheloroppgaven er Azure.

## **Standup**

En standup er en vanlig praksis i Scrum der teammedlemmene forteller kort om det de har gjort i et prosjekt og hva de skal gjøre videre, samt at eventuelle utfordringer i prosjektet tas opp. Her er det vanlig at medlemmene står oppreist, herav navnet standup.

## **Versjonskontrollsyste**m

Et versjonskontrollsyste

m er et system, typisk et dataprogram, som holder styr på ulike versjoner av ulike filer. Så ofte man ønsker, kan man legge til versjoner av filer og fritt gå mellom dem i ettertid.