



Læringsplattform for arbeidsinkludering

Bachelorprosjekt 2020

Gruppe 16



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

PROSJEKT NR.
16

TILGJENGELIGHET
Offentlig

Telefon: 22 45 32 00

BACHELORPROSJEKT

HOVEDPROSJEKTETS TITTEL Læringsplattform for arbeidsinkludering	DATO 23.05.2020
	ANTALL SIDER / BILAG 223 / 1
PROSJEKTDELTAKERE Alexandros Messaritakis Chousein Aga - s326156 Alina Zielinska - s330170 Amina Shahzad - s319898 Elias Pederstad - s326264 Silje Stephanie Gonçalves Bjørknes - s320752	INTERN VEILEDER Erika Gubrium erika.gubrium@oslomet.no / erikgu@oslomet.no

OPPDAGSGIVER Avdeling for samfunnsfag (SAM) ved OsloMet – Storbyuniversitetet	KONTAKTPERSON Erika Gubrium - erika.gubrium@oslomet.no / erikgu@oslomet.no
--	--

SAMMENDRAG
Dette bachelorprosjektet omhandler utvikling av en webapplikasjon for avdelingen SAM ved OsloMet og er gjennomført ved OsloMet – Storbyuniversitetet av studenter ved Informasjonsteknologi, anvendt data teknologi og Dataingenør.
Målet med dette prosjektet var å lage en enkel plattform der de ansatte kan legge til/redigere og/eller slette kurs. Denne plattformen skal også være brukervennlig for sluttbrukeren brukeren som skal ta kurs.
For å oppnå dette målet utviklet vi en webapplikasjon som skal være enkel å bruke for både de som skal ta kurs og de som kan opprette kurs. Det skal være mulig for ansatte som kan legge til kurs å kunne velge mellom video/bilde, tekst eller quiz for kurset. Denne webapplikasjonen er av enkel design.

3 STIKKORD
Læringsplatform
Webapplikasjon
React

Leserveiledning	6
1 Presentasjon	7
1.1 Innledning	7
1.1.1 Presentasjon av prosjektgruppen	7
1.1.2 Faglige forutsetninger	7
1.1.3 Presentasjon av oppdragsgiver	8
1.1.4 Presentasjon av kunde	9
1.2 Bakgrunn for prosjektet	9
1.3 Problemstilling	9
1.4 Konsept	10
1.5 Beskrivelse av løsning	10
1.5.1 Brukergrensesnitt	11
1.5.2 Administrasjonsgrensesnitt	12
1.6 Oppsummering	12
2 Prosessdokumentasjon	14
2.1 Innledning	14
2.2 Projektmetodikk - Scrum	14
2.2.1 Scrum Team og Scrum Master	15
2.2.2 Daily Scrum og Sprinter	15
2.3 Projektverktøy	15
2.3.1 Generelle verktøy	16
2.3.2 Utviklingsverktøy	17
2.3.3 Dagbok	19
2.3.4 GitHub	20
2.4 Prosjektfaser og arbeidsplan	20
2.4.1 Fase 1 - Læring, prosess og planleggingsfase	21
2.4.2 Fase 2 - Bygge Minste brukbare produkt (MBP) og stabilisere den	22
2.4.3 Fase 3 - Videreutvikling av webapplikasjonen	22
2.4.4 Ansvarsområder	22
2.4.5 Risikovurdering	23
2.4.6 Sprinter	24
2.5 utfordringer og refleksjoner	28
2.6 Oppsummering av prosessdokumentasjon	29
3 Produktdokumentasjon	30

3.1 Innledning	30
3.1.1 Introduksjon til applikasjonen	30
3.2 Funksjonell systembeskrivelse	30
3.2.1 Aktører	30
3.2.2 Brukergrensesnitt	31
3.2.3 Administrasjonsgrensesnitt	33
3.2.4 UI-utforming	35
3.3 Applikasjonens oppbygging	35
3.3.1 Frontend	36
3.3.1.1 Oppstart med React/ struktur av filer	36
3.3.1.2 Bootstrap	39
3.3.1.3 Side om prosjektet - Informasjonsside (About)	40
3.3.1.4 Hovedside (HomePage)	43
3.3.1.5 Modul hovedside	45
3.3.1.6 Modul lekse side med tekst	50
3.3.1.7 Modul lekse side med video	51
3.3.1.8 Modul lekse side med quiz	52
3.3.1.9 Admin hjemmesiden	53
3.3.1.10 Admin create modul side	55
3.3.1.11 Admin leksjonsside med tekst	56
3.3.1.12 Admin leksjonsside med video	56
3.3.1.13 Admin leksjonside side med quiz	58
3.3.1.14 Login side	64
3.3.1.14 Registrering Side	66
3.3.1.15 Ruter	66
3.3.1.16 Samspil med BackEnd delen	68
3.3.2 Backend	74
3.3.2.1 Database	74
3.3.2.2 Arkitektur	80
3.3.2.3 Publisering av APlet	87
3.3.2.4 Autentisering og autorisasjon	87
3.4 Testing	94
3.4.1 Brukertesting av prototype	94
3.4.2 Integrasjonstesting	96

3.4.3 Systemtesting	97
3.5 Utfordringer og refleksjoner	98
3.6 Oppsummering av produktdokumentasjon	98
4 Brukerveiledning	100
4.1 Pålogging	100
4.2 Opprettelse av en ny modul av administrator	101
4.3 Lage innhold	103
4.4 Gjennomførelse av modul	106
5 Referanseliste	109
6 Vedlegg	113
6.1 Terminologi	113
6.2 Nåværende WIX webside	116
6.3 Kravspesifikasjon	119
6.4 Dagbok	121
6.5 Sprinter Jira board	136
6.6 Wireframe	141
6.7 Figma prototype	149
6.8 Vedlegg for login funksjonalitet til Facebook og Google - første forsøk.	154
6.8 Vedlegg for backend	157
6.9 GitHub	201
6.10 Integrasjonstesting	211
6.11 Systemtesting	211

Leserveiledning

Denne rapporten er skrevet i forbindelse med et bachelorprosjekt i avdelingen for informasjonsteknologi (TKD) ved OsloMet - Storbyuniversitetet. Denne leseveiledning gir en oversikt over innholdet i rapportens hoveddeler.

1. **Presentasjon s.7-14**

Presentasjonen er en sammenfatning av hele prosjektet. Leseren av presentasjonen skal uten forkunnskaper og teknisk kompetanse kunne lese og forstå hva bachelorprosjektet går ut på. I denne delen presenteres bachelorgruppens medlemmer, oppdragsgiver, kunde, bakgrunn, samt en beskrivelse av løsningen.

Hver del av rapporten kan leses hver for seg, men det anbefales å lese dette dokumentet først og i sin helhet.

2. **Prosessdokumentasjon s.15-31**

Prosessdokumentasjonen er et dokument som beskriver bachelorgrouppe [HØ1] bruk av metodikk, verktøy og fremgangsmåte for gjennomføring av bachelorprosjektet. Her skildres det blant annet hvordan bachelorguppen har arbeidet, loggføring av arbeidet og inndeling av sprinter.

3. **Produkdokumentasjon s.32-105**

Produkdokumentasjonen beskriver den tekniske løsningen og gir en total fremstilling av applikasjonen. Det er en detaljert beskrivelse av hvordan backend- og frontend-løsningen er bygget opp

4. **Brukerveiledning s.106-114**

Brukerveiledningen gir en beskrivelse av all funksjonalitet applikasjonen har som sluttbrukeren kan benytte seg av. Dokumentet tar for seg hver nettside og gir en kort forklaring av knapper og funksjonalitet. Brukerveiledningen skal fungere som en manual for kunden som overtar løsningen, men kan også leses av interessenter for å få en helhetlig oversikt over funksjonaliteten.

5. **Referanseliste s.115-118**

Samtlige referanser i rapporten er samlet her.

6. **Vedlegg s.119-230**

1 Presentasjon

1.1 Innledning

Hensikten med presentasjonen er å gi et helhetlig bilde av oppgaven. I denne delen blir vi først kjent med prosjektgruppen, oppdragsgiver, kunde og bakgrunn for prosjektet. Deretter forklare vi oppgavens problemstilling og en beskrivelse av løsningen.

1.1.1 Presentasjon av prosjektgruppen

Denne prosjektgruppen, som består av fem studenter ved OsloMet, er svært mangfoldig og tverrfaglig. To av prosjektdeltakerne går på Dataingeniør, to går på linjen for Anvendt datateknologi og den siste går Informasjonsteknologi linjen. I tillegg snakker vi til sammen 11 forskjellige språk og har 5 etniske opprinnelser.

Amina Shahzad studerer Ingeniørfag-data. Hun har fagbrev i kontor- og administrasjonsfaget og to års arbeidserfaring fra statlig virksomhet. Gjennom studietiden har hun jobbet som studentassistent i webutvikling og inkluderings-design-faget, har vært studentambassadør for OsloMet og jobbet med promotering av IT-studiene.

Alex Mesaritakis Chousein Aga studerer Informasjonsteknologi (IT). Han er interessert i å lage programmer i forskjellige språk. Har tatt C++/C# emne og kan flere programmeringsspråk. I bachelorprosjektet jobbet han med backend-delen, herunder å lage databasediagrammet.

Silje Bjørknes studerer Ingeniørfag-data. Hun har en tverrfaglig bakgrunn som består av en bachelorgrad i juss fra Brasil, en mastergrad i sjørett fra Universitetet i Oslo og arbeidserfaring fra flere ulike bransjer. Hun er svært interessert i informasjonssikkerhet og har i løpet av prosjektet jobbet med implementasjon av databasen og backend.

Alina Zielinska studerer Anvendt Datateknologi. For 5 år siden flyttet Alina fra Ukraina der hun fullførte en mastergrad i økonomi og management. Fra oktober 2019 har hun jobbet 50 prosent som QA i et IT-firma. Hun er også Universitetslektor på OsloMet for studenter fra estetiske fag (HTML, CSS, vårsemesteret 2020) og har erfaring som lærerassistent for "Web-utvikling" høsten 2019. Våren 2019 var Alina en del av teamet som reiste til Genève for å delta på WISI konferansen.

Elias Pederstad studerer Anvendt Datateknologi. Elias har et internship i LØPE, et digitalt produkt og designstudio, der han har fått erfaring med prototyping, brukertesting og designmetodikk. Gjennom studietiden har Elias vært studentrepresentant og ledet Promoterings-teamet for IT-studier ved OsloMet. Der fikk han lede samarbeidet mellom IBM og OsloMet sitt bidrag i Oslo som Europeisk Miljøhovedstad. Elias var også en del av teamet som reiste til Genève for å delta på WISI konferansen.

1.1.2 Faglige forutsetninger

Anvendt Datateknologi – faglige forutsetninger:

Gjennom studiet har «anvendt-studentene» hatt spesialiserte fag som har rettet fokuset mot universell utforming i utvikling av digitale løsninger. Erfaring fra fagene *Universell utforming av*

IKT, Menneske maskin interaksjon og Prototyping, har lært studentene å utvikle brukervennlig programvare.

Følgende fag har også vært spesielt relevante for dette prosjektet: *Testing av programvare, DATA1700, Webapplikasjoner, ADTS2310*.

Dataingeniør og informasjonsteknologi – faglige forutsetninger:

Dataingeniørene har gjennom studiene hatt mer dyptgående programmeringsfag. I løpet av bachelorprosjektet har de brukt mye av kompetansen de har opparbeidet seg i disse fagene. Viktigst av alt har fagene lagt til rette for ny læring. Ved hjelp av online-kurs har de raskt opparbeidet seg ny kunnskap i andre kodespråk og utviklingsplattformer.

Følgende fag har vært spesielt relevante for dette prosjektet: *DATA1500 - Database, DAPE1400 - Programmering, DATA1600 - Programutvikling, ITPE3200 - Webapplikasjoner, ADTS2310 - Testing av programvare, DATA1700 - Webprogrammering, DAVE3605 - Effektiv kode med C og C++*.

I tillegg har faget *DAFE2200 - Systemutvikling* vært grunnleggende for en helhetlig gjennomføring av prosjektet.

1.1.3 Presentasjon av oppdragsgiver

Oppdragsgiveren for bachelorprosjektet er Erika Gubrium, professor ved fakultet for samfunnsvitenskap (SAM) ved OsloMet. For tiden jobber hun og kollegene hennes på et prosjekt som heter Work Inclusion of Migrants, forkortet som «WIX».

WIX er et samarbeidsprosjekt mellom forskere fra tre universiteter; Oslo Metropolitan University, National Law School of India University i Bangalore og Higher School of Economics i Moskva. Prosjektet er utviklet og ledet av eksperter innenfor sosialpolitikk, sosialt arbeid og urbane studier. Resultatet av WIX-prosjektet skal være flere online MOOCs (massive open online course), som omhandler arbeidsinkludering av migranter i sammenlignbare urbane kontekster.

Sammen med hennes medarbeidere har Erika jobbet med utvikling av moduler for online-kurs som fokuserer på politiske og sosiale velferdsmessige sammenhenger i de tre byene Oslo, Bangalore og Moskva. Hun har hatt tre finansierte pedagogiske samarbeidsprosjekter tidligere. I disse prosjektene har de vært opptatt av å sende studenter til og fra OsloMet sin samarbeidsinstitusjon i Bangalore. De har erfart at å rekruttere studenter til å besøke Oslo har vært enkelt, men å rekruttere masterstudenter til å reise til Bangalore har derimot vist seg å være mer utfordrende. Det har gjort at de har utforsket andre muligheter for å samarbeide om utdanning på tvers av institusjonene.

Opprinnelig var modulene planlagt som MOOCs der studenter på tvers av institusjoner gjennomførte modulene samtidig. De beveget seg etterhvert vekk fra den strategien og mot en modell der modulene ble integrert i pågående masterprogram. Dette gjør det mulig for “flipped learning”, som vil si at modulene ikke erstatter ordinære forelesninger, men heller er et supplement.

Erika og teamet hennes ønsker å sammenligne fattigdom og arbeidsinkluderingstiltak i tre veldig forskjellige miljøer (Oslo, Bangalore, Moskva). I tillegg ønsker de å tilby en plattform for

co-teaching og læring på tvers av de tre institusjonene. Fremover er de interessert i å skalere opp bruken av webapplikasjonen til flere land og institusjoner.

1.1.4 Presentasjon av kunde

Prosjektet er finansiert av **Forskningsrådet og Direktoratet for Internasjonalisering og Kvalitetsutvikling i høyere Utdanning** (DiKU). Forskningsrådet er et norsk organ som ligger under kunnskapsdepartementet. De investerer og finansierer forskning og innovasjonsprosjekter på vegne av regjeringen. DiKU er et direktorat som også ligger under kunnskapsdepartementet. De jobber for å styrke kvaliteten i norsk utdanning og fremmer utvikling, nyskapning, internasjonalt samarbeid og digitale læringsformer.¹

1.2 Bakgrunn for prosjektet

WIX-teamet har til nå prøvd mange forskjellige løsninger for å opprette og behandle moduler. I begynnelsen opprettet de modulene i EdX, en MOOC laget av Harvard University og MIT. Formatet var fint og brukervennlig for sluttbrukeren, men det viste seg etter hvert at programmet var vanskelig å lære og at prisen ikke rettferdiggjorde funksjonaliteten.

WIX-teamet prøvde deretter å bruke *Bokskapet*, OsloMet sin egen kursplattform, som også bygger på EdX plattformen. Denne plattformen hadde flere utfordringer, deriblant mangel på teknisk support, publisering av innhold, opprettelse av nye passord og bruk av plattformen fra samarbeidsinstitusjonene i andre land.

Da Bokskapet heller ikke var en god nok løsning flyttet de innholdet over i en bloggplattform, kalt Wordpress, som var tilgjengelig gjennom OsloMet. Det viste seg at denne plattformen også var vanskelig å bruke, noe som resulterte i moduler som verken var brukervennlig eller hadde tilstrekkelig visuell kvalitet (Se vedlegg 6.2).

Erika og teamet i WIX var derfor fremdeles på utkikk etter en plattform til å dele og opprette moduler da de kom i kontakt med oss. De uttrykte også at de ønsket seg ansikt-til-ansikt opplæring av plattformen de skal bruke, slik at de selv kan ha opplæring for andre.

Dette er bakgrunnen for vårt prosjekt.

1.3 Problemstilling

Problemstillingen som dette prosjektet skal løse er å utvikle en skreddersydd webapplikasjon der teamet i WIX kan opprette og redigere moduler med både brukervennlighet og god visuell kvalitet. De som eier innholdet til modulen har stor variasjon av teknisk kompetanse som gjør at vi blir nødt til utvikle en løsning som er enkel og intuitiv for alle å bruke.

Gjennom samtale med oppdragsgiver ble det tidlig opprettet en liste med kravspesifikasjoner på bakgrunn av problemstillingen. Følgende spesifikasjoner omhandlet alle ønskede formater til læring (se liste over formater under kapittel 1.5.1), enkle pålogging- og skaleringsmuligheter

¹ Forskningsrådet, Diku, no date

(videreføring av administrasjonsrettigheter) og brukervennlighet. Problemstillingen er løst dersom vi klarer å møte alle avtalte kravspesifikasjoner.

1.4 Konsept

I startfasen ble det tydelig konkretisert hva bachelorprosjektet skulle inneholde gjennom samarbeid med oppdragsgiver. I denne fasen gikk vi systematisk gjennom konkurrerende plattformer (Coursera, Udemy), samt erfaringer fra tidligere brukte plattformer (EdX, Bokskapet, Wordpress). Målet var å plukke ut det beste, eller den mest prefererte, løsningen basert på noen spesifiserte krav og samle det til en skreddersydd løsning. Basert på disse kravene utviklet prosjektgruppen prototyper laget av ark og post-it-lapper. På den måten fikk vi illustrert en samling av de prefererte funksjonene og det visuelle brukergrensesnittet oppdragsgiver ønsket seg. Prototypen ble lagt frem for oppdragsgiver slik at vi kunne få en god diskusjon om endelig løsning.

Konseptet og løsningen på problemstillingen ble å utarbeide en todelt webapplikasjon med en administrasjonsside og en brukerside. Administrasjonssiden skal være det området der forskere i WIX-teamet kan opprette og redigere moduler bestående av PowerPoint-slides, tekst, video, bilder, pdf, flervalgsspørsmål, samt ha et forum for diskusjoner seg imellom.

Administrasjonssiden skal være enkel og intuitiv å bruke, etterligne brukersiden og inneholde forskjellige hjelpebidrifter og informasjon for administrator slik at den støtter selvstendig arbeid. Den andre delen av webapplikasjonen er brukersiden som er tilgjengelig for alle uten å måtte logge seg inn. Her skal brukerne kunne gjennomføre modulene ved å se video, svare på spørsmål, diskutere med hverandre og lære.

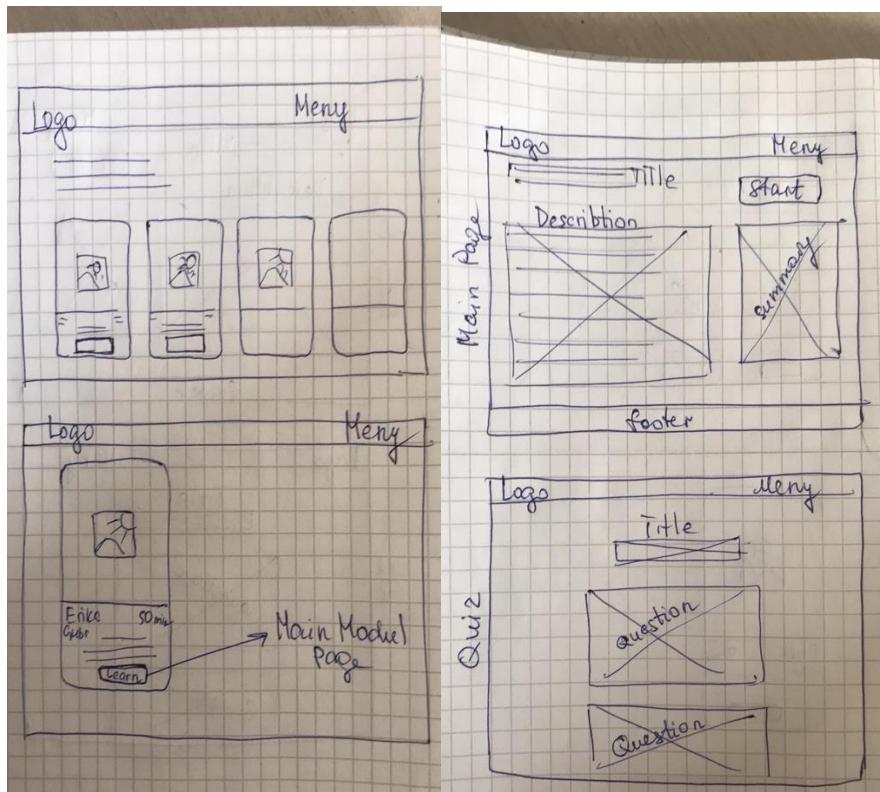
Begrepet "What you see is what you get" (WYSIWYG) påvirket designet til konseptet: Når en administrator oppretter en modul skal vedkommende være sikker på at den blir seende ut akkurat slik den ble laget. Utenom dette prinsippet var ikke prosjektgruppen begrenset til én spesifikk designprofil, men det ble tidlig diskutert om plattformen skulle ligne OsloMet sin profil eller ikke.²

1.5 Beskrivelse av løsning

Denne beskrivelsen av plattformen gir en kort forklaring og oversikt av webapplikasjonen. Beskrivelsen er en del av presentasjonen og kan derfor bli lest som en avslutning og oppsummering av første del av rapporten. For en mer nyansert forklaring og gjennomgang av applikasjonen anbefaler vi å lese brukerveiledningen som har som formål å hjelpe nye brukere i gang med applikasjonen.

Applikasjonen vi har laget er en kursplattform hvor administrator kan opprettes kurs og studenter kan gjennomføre ulike moduler. Modulene består av en rekke sider med forskjellige innhold. Ved å trykke seg til neste side avanserer brukeren videre i modulen. Modulene kan bestå av tekst, bilde, video, PDF, PowerPoint eller flervalgsspørsmål. Hver side i en modul inneholder kun ett av disse formatene om gangen.

² Bernheim, 2019.



Figur 1 – Papirprototype

Det er to hoveddeler i applikasjonen; brukersiden og administrasjonssiden. Bachelorgruppens mål var å lage disse to sidene så like som mulig, slik at administratoren som oppretter en modul kan forvente å se hvordan modulen ser ut i brukersiden (se kapittel 1.4).

1.5.1 Brukergrensesnitt

Det første brukeren blir møtt med på brukersiden er en oversikt over alle modulen. De er stilt opp ved siden av hverandre som kort (cards). Hvert kort inneholder et bilde som er representativt for temaet, tittelen til modulen, informasjon om hvem som har laget modulen, hvor lenge modulen varer og en "lær mer"-knapp. I menylinjen øverst på siden er det knapper som tar deg til en informasjonsside og en logg-inn/«registrer deg»-side.

Bachelorgruppen valgte å bruke modulsiden som landingsside fordi vi ønsket å bringe brukeren rett til hovedfunksjonaliteten. Ved å ha store innbringende bilder skal det også være fristende å utforske modulen videre.

Trykker brukeren videre på en modul kommer vedkommende til informasjon om modulsiden. Her står det skrevet en oppsummering av modulen, en innholdsfortegnelse for modulen og informasjon om lengde, hvilken institusjon modulen kommer fra, pris og språk. Det er også et bilde og informasjon om kurslederen. Øverst til høyre på siden er det en stor og synlig knapp der det står "Start the course".

Når brukeren starter en modul er det en gjennomgående progresjons-bar øverst på siden. Her får brukeren oversikt over antall sider det er totalt for det spesifikke kurset. Hvis brukeren er

logget inn får vedkommende en beskjed om hvilke sider hun har gjennomført og hvilken side hun er på.

Gjennom modulen er det fem forskjellige formater. De er listet under med tilhørende informasjon:

1. **Tekst:** I det tekstlige formatet er det enkel tekst, kulepunkter og lignende som brukeren kan forvente å se. I tillegg kan det bli vedlagt bilder.
2. **Video:** Med videoformat kan brukeren se videoer i nettleseren enten gjennom youtube eller ved opplastede filer.
3. **PDF:** Gjennom en PDF viewer kan brukeren bla seg gjennom et pdf-dokument. Dette er for å kunne utnytte eksisterende materiale uten å måtte endre på filformatet.
4. **PowerPoint:** Gjennom en PowerPoint viewer kan brukeren bla seg gjennom slides. Her er det lurt å utnytte eksisterende innhold.
5. **Flervalgsspørsmål:** Gjennom flervalgsspørsmål kan brukeren bli testet i det han eller hun nettopp har lest. Målet er ikke å samle inn resultatdata, men å bidra til brukerens læring gjennom en slik funksjonalitet.

Nederst på hver side i modulen er det mulig å legge ved et diskusjonsforum. Hvis dette er valgt kan innloggede brukere kommentere og diskutere innholdet på akkurat den siden i modulen.

1.5.2 Administrasjonsgrensesnitt

For å komme til administratorgrensesnittet må administrator logge seg på gjennom nettsiden. Her kan vedkommende logge på med e-post og passord eller via Facebook-kontoen sin. Når administrator har logget på møter vedkommende det samme brukergrensesnittet som brukeren, men med mulighet til å redigere innholdet. Administratoren møter modulsiden der alle tilgjengelig og utilgjengelige moduler er stilt opp ved siden av og under hverandre. Ved å trykke på en modul kommer administrator inn i informasjonsdelen av den valgte modulen. Her har administrator mulighet til å redigere tittel, innholdsfortegnelse, informasjon om kurset og annet innhold. Videre har administrator mulighet til å legge til eller redigere innholdet. På samme måte som brukeren nавigerer seg videre inn til kurset, trykker administrator på den samme knappen som tar han eller hun til kurset. Her kan administrator legge til innhold i fem forskjellige formater som nevnt ovenfor (se kapittel 1.5.1).

1.6 Oppsummering

Da vi kom i kontakt med Erika og WIX-teamet ble vi raskt interesserte i arbeidet de holdt på med. Arbeidsinkludering for migranter er ikke bare nødvendig, men det stod også nært for flere av bachelorgruppens medlemmer. Work Inclusion of Migrants (WIX) har som mål å forske på, publisere og utdanne studenter om fattigdom og arbeidsinkludering hos migranter i forskjellige urbane kontekster. Prosjektet ga bachelorgruppen en unik mulighet til å starte fra bunn av og gå igjennom alle fasene i systemutviklingsprosessen. Videre fikk vi svært relevant

arbeidserfaring relatert til våre forskjellige arbeidsområder, herunder utviklingsmetodikk, frontend, backend eller prosjektledelse.

En utfordringene for WIX-teamet var å identifisere og etablere en kursplattform som de kunne bruke til å utdanne og informere interesserter og andre aktører i det arbeidet de gjør. Hovedfunksjonaliteten skulle omhandle flere MOOCs som skulle bli brukt løpende i masterprogrammer. Dette viste seg å være vanskelig å finne en plattform med de kravene og spesifikasjonene teamet i WIX har.

Dette ble bachelorgruppens problemstilling og løsningen ble å utvikle en egen plattform skreddersydd for behovet til WIX. Gjennom mye samtaler tidlig i prosjektfasen lagde vi digitale prototyper basert på testing av konkurrerende plattformer, feedback på vår egen prototype og krav fra oppdragsgiveren.

Applikasjonens hensikt er å enkelt og brukervennlig opprette, redigere og dele kurs. Det skal gjøres av mennesker med en bred variasjon av datakyndighet. Innholdet i kursene gir den informasjonen som allerede eksisterte hos innholdseieren i WIX teamet. For oss innebar det at vi måtte utvikle en plattform som mottok informasjon fra et tekstlig format, sammen med video, PowerPoint, pdf og flervalgsspørsmål. Videre fikk applikasjonen et forum der studenter som gjennomfører kursene kan kommunisere og debattere rundt innholdet i kurset.

Bachelorgruppen bestående av fem mennesker med variert kompetansefelt og bakgrunn gjennomførte denne våren utviklingen av denne plattformen til stor glede fra oppdragsgiver. Utbytte gruppen fikk av prosjektet var utviklet problemløsingsevne, erfaring ved å raskt ta til seg ny læring og anvende det i prosjektet, og ikke minst nye, gode bekjentskap. Vi er takknemlig for muligheten til å løse dette oppdraget og ønsker å takke for støtten og engasjementet vi fikk fra Erika, både som oppdragsgiver og veileder i dette prosjektet.

2 Prosessdokumentasjon

2.1 Innledning

Hensikten med prosessdokumentasjonen er å beskrive ulike faser av prosjektarbeidet og hvordan disse har blitt gjennomført. Først redegjør vi for hvilken arbeidsmetodikk og prosjektverktøy gruppen har brukt for å gjennomføre prosjektet. Videre beskriver vi utviklingsfasene i form av sprinter.

Bachelorgruppen fikk muligheten til å gjennomføre et prosjekt fra en oppdragsgiver internt i OsloMet. Oppdragsgiver utarbeidet og forsket på arbeidsinkludering i urbane miljøer og trengte en skreddersydd webapplikasjon for kurs til utdanning av studenter.

I likhet med produktdokumentasjonen er prosessdokumentasjonen et viktig bidrag til å muliggjøre videreføring av prosjektet. Det er ikke uvanlig at nye studenter eller utviklere ved OsloMet bygger videre på eller gjør endringer i applikasjoner utviklet av bachelorstudenter. Da vil de nyttiggjøre seg av god dokumentasjon av arbeidet.

Dokumentet kan leses av alle type interesserter, også de uten teknisk bakgrunn, men vi anbefaler og lese Del 1 Presentasjon, samt terminologilisten for å få en fullstendig forståelse av prosessen gruppen har gjennomgått.

2.2 Prosjektmetodikk - Scrum

Høsten 2018 hadde alle i prosjektgruppen faget *Systemutvikling (DAFE2200)*. Her fikk vi en gjennomgang av en klassisk utviklingsprosess og anbefalte prosjektmetodikker.

I vårt prosjekt brukte vi en smidig, inkrementell tilnærming fremfor fossefall. Det vil si at vi ikke lagde detaljerte planer for hele prosjektet, men delte det opp i flere små «prosjekter» kalt iterasjoner eller «sprinter». Det finnes utallige metoder og rammeverk som bygger på smidige prinsipper. Scrum og Kanban er blant de mest populære.³

I begynnelsen prøvde vi å jobbe mest med Kanban, men etter en kort stund tok vi Scrum-rammeverket i bruk. Ifølge Schwaber og Sutherland (2017) er Scrum et rammeverk for å utvikle, levere og opprettholde komplekse produkter. Det er ikke én prosess, ett verktøy eller én bestemt metode, men du kan benytte deg av mange ulike prosesser og teknikker innenfor dette rammeverket. Et viktig prinsipp ved Scrum er å kontinuerlig gjøre hyppige og inkrementelle endringer i produktet, teamet og arbeidsmiljøet. Målet ved Scrum er å effektivt levere produkter av mest mulig verdi.

For oss har bruken av Scrum synliggjort alle stadier av prosessen gjennom utviklingen. Det er lettere å håndtere endringer som oppstår underveis da Scrum-rammeverket består av korte tidsintervaller eller sprinter med fokus på kontinuerlig læring gjennom regelmessige tilbakemeldinger. For å lykkes med Scrum består dette rammeverket også av en rekke komponenter som Scrum-teamet, Scrum master, Sprinter og Daily Scrum.⁴

³ Wålberg, 2018.

⁴ Schwaber og Sutherland, 2017.

2.2.1 Scrum Team og Scrum Master

Et Scrum team består av tre komponenter; en produkteier, et utviklingsteam og en Scrum master⁵. I vårt prosjekt er produkteieren vår oppdragsgiveren Erika. Hun er ansvarlig for å få størst mulig verdi ut av produktet som prosjektgruppen lager. Produkteier har vanligvis ansvar for å styre backloggen, men av praktiske årsaker har dette ansvaret ligget hos Scrum masteren.

Scrum master er en fasilitterende rolle i et Scrum team. Scrum masteren er ansvarlig for å promotere Scrum og hjelpe gruppen å forstå teorien, praksisen og reglene rundt Scrum. I vårt prosjekt har Alina hatt rollen som Scrum master. Hun har satt opp tydelige rollefordeling, håndtert verktøy for logging, arrangert sprinter, Daily Scrum og andre møter. Scrum master er også en del av utviklingsteamet.

Utviklingsteamet er siste komponent i Scrum teamet. Det er en selvorganisert og tverrfaglig gruppe med utviklere. Prosjektgruppen vår er i dette tilfelle utviklingsteamet som består av både frontend- og backend-utviklere.

2.2.2 Daily Scrum og Sprinter

En sprint er en tidsbegrenset periode, innenfor en måned eller mindre, hvor det i slutten av hver sprint skal leveres et ferdig eller leveringsklart produkt. I vårt tilfelle hadde vi en sprint på to uker med et bestemt mål for hver sprint som var å levere fungerende funksjoner. Sprintene består av planlegging, "Daily Scrum", utvikling, presentasjon av funksjon og et retrospektiv av Sprinten.⁶

Planleggingsmøter ble gjennomført i begynnelsen av hver Sprint. Da planla vi arbeidet for de neste to ukene og ble enige om hva slags resultat vi ønsket å oppnå.

Daily Scrum er et daglig møte som blir gjennomført på kort tid, gjerne innenfor 15 minutter. Hensikt med møtet er å synkronisere gruppens progresjon og diskutere det som vanskelig gjør gruppens evne til å nå Sprintens mål. Selv om vi ikke alltid hadde behov for daglige møter, fant vi metoden veldig nyttig. Til å følge progresjon og koordinere oppgaver brukte vi Toggle og Jira som verktøy (Se vedlegg 6.4 og 6.5).

Sprint retrospektiv er et møte vi hadde etter hver sprint. Dette ga bachelorgruppen muligheten til å se tilbake på det arbeide som hadde blitt gjort og identifisere de aktivitetene vi gjorde bra og de vi gjorde mindre bra, slik at vi kunne forberede oss i neste sprint.

2.3 Prosjektverktøy

I dette delkapittelet beskriver vi de ulike verktøyene bachelorgruppen benyttet seg av under prosjektperioden.

⁵ Schwaber og Sutherland, 2017.

⁶ Schwaber og Sutherland, 2017.

2.3.1 Generelle verktøy

	Verktøy	Beskrivelse/brukes til	
	LucidChart	LucidChart er en webbasert plattform som tillater brukere å samarbeide om tegninger og diagrammer. Vi brukte verktøyet til å lage sitemap, tilstandsdiagram og aktivitetsdiagram (Se vedlegg 6.6).	https://www.lucidchart.com
	Jira	Jira er et prosjektstyringsinstrument. Vi brukte Jira til å planlegge og følge Sprint-progresjon basert på Scrumrammeverket. (se kapittel 2.2.2)	https://www.atlassian.com
	Trello	Trello er et webbasert oversiks- og prosjektstyringsverktøy. Vi brukte Trello i begynnelsen av prosjektet som plannleggingsinstrument. Fra midten og til slutt av prosjektet brukte vi bare Jira og Excel som hovedinstrumenter til planlegging og oppfølging av arbeidsprosessen. Jira er mer systemutvikling fokusert og egnet seg bedre enn Trello.	https://trello.com
	Toggl	Toggl er en tidssporings-app som ble brukt til å ha oversikt over gruppens arbeid.	https://toggl.com/
	Google Meet	Google Meet er en video kommunikasjonstjeneste utviklet av Google. Vi brukte denne tjenesten for møter og annen kommunikasjon i gruppen.	https://meet.google.com/
	Adobe Photoshop	Adobe er et bilderedigeringsprogram som vi brukte til å lime printscreen av sider med «scrolling».	http://editor.0lik.ru/en/photoshop-online-new.html
	Lightshot	Lightshot er en app for å ta «screenshots» og som gir oss muligheten til å redigere skjermbilder ved å peke på	https://app.prntscr.com/en/in

		viktige deler umiddelbart etter at vi har tatt bildet.	dex.html
	Draw	Draw er et gratis online diagramprogramvare for å lage flytdiagrammer, prosessdiagrammer, org-diagrammer, UML, ER og nettverksdiagrammer.	https://app.diagrams.net/

2.3.2 Utviklingsverktøy

	Figma	Vi brukte Figma for design og prototype.	https://www.figma.com
	React	React er et JavaScript-bibliotek for å lage/bygge brukergrensesnitt.	https://reactjs.org/
	Visual Studio Code	Dette er et verktøy for å bygge WebApplikasjon både for Frontend og Backend, men egner seg best for Frontend.	https://code.visualstudio.com
	Visual Studio	Verktøyet ble brukt for å bygge applikasjonen på Backend.	https://visualstudio.microsoft.com
	GitHistory	Utvidelser for Visual Studio Code. Se og søk i historien til en eller alle grener (git log), til en fil, linje i en fil (Git Blame). Se historien til en forfatter. Sammenlign grener, forpliktelser, filer på tvers av forpliktelser.	https://githistory.xyz
	React Native Tools	Denne utvidelsen gir et utviklingsmiljø for React Native prosjekter. Ved hjelp av denne utvidelsen kan vi feilsøke koden og raskt kjøre reaksjons-native kommandoer fra kommando paletten.	https://reactnative.dev

	ReactBootstrap	Front-end komponentbibliotek som ble brukt til å utvikle WebApplikasjon.	https://react-bootstrap.netlify.com/
	Visual Paradigm Online	Et online-verktøy som vi brukte til å bygge klasseDiagram.	https://online.visual-paradigm.com/
	Swagger	En programvare med åpen kildekode støttet av et stort økosystem av verktøy som hjelper utviklere med å designe, bygge, dokumentere og konsumere RESTful web-tjenester. Vi brukte Swagger-verktøyet for automatisert dokumentasjon og kodegenerering.	https://swagger.io/
	Postman	Samarbeidsplattformen for API-utvikling ble brukt ved integrasjonstesting.	https://www.postman.com/
	.ASP .Net Core	.Net Core er en åpen kildekode utviklingsplattform.	https://docs.microsoft.com/en-us/dotnet/core/
	Azure	Azure er en cloud computing-plattform og infrastruktur skapt av Microsoft.	https://azure.microsoft.com/nb-no/
	SQL Server Management Studio (SSMS)	En programvare program som brukes til å konfigurere, administrere og administrere alle komponenter i Microsoft SQL Server.	https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15

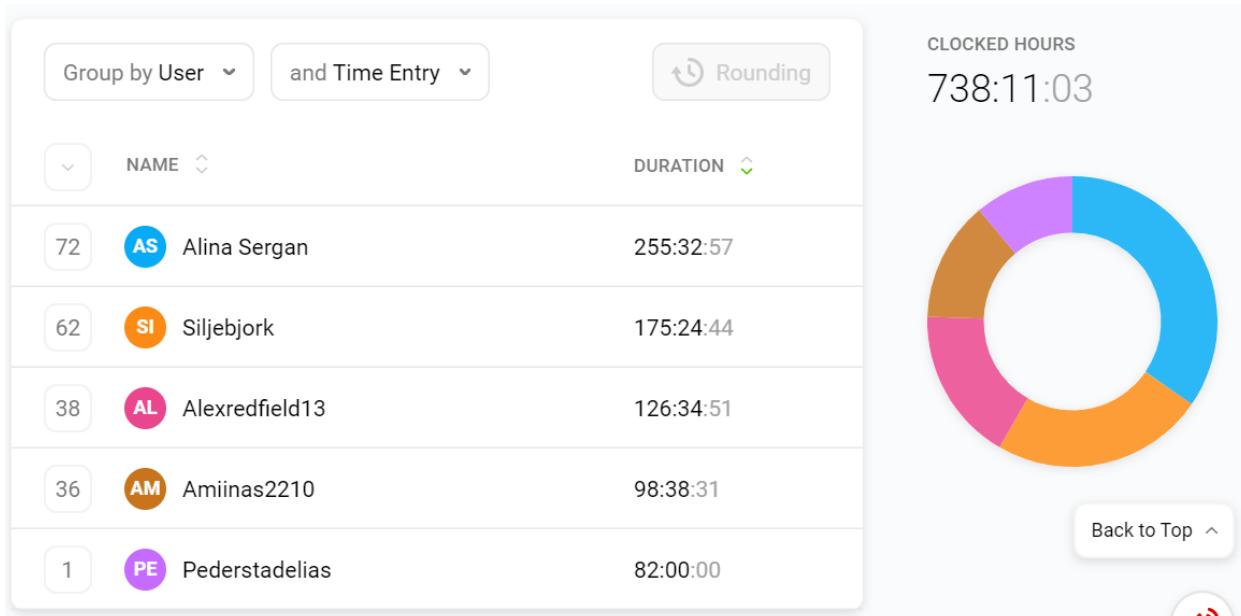
2.3.3 Dagbok

For å følge gruppemedlemmene s hverdagsaktiviteter, og å ha oversikt over oppgaver progresjon, ble det bestemt å fylle ut en dagbok underveis. Først brukte vi Google Docs, men etter en kort periode ble det synlig at det er ikke ga oss god nok oversikt og rapportering. Derfor begynte vi å bruke Toggle. Toggle er en populær tjeneste som lar oss føre en logg over medlemsaktivitet med manuell tidssporing. Spesielt under COVID-19 og manglende mulighet å jobbe sammen i ett rom, så vi at dette var svært viktig.

Det var hvert enkelt gruppemedlems ansvar å beskrive oppgaver i Toggle slik at det ble hjelpsomt for alle å ha oversikt om hva hver av oss gjorde, hvilken oppgave og hvor lenge. Da kunne vi forstå hverandres bidrag og eventuelle problemer som oppstod relatert til bestemte oppgaver (Se vedlegg 6.4).



Figur 2 – Oversikt over timer arbeidet i Toggle



Figur 3 - Oversikt over timer arbeidet i Toggle

2.3.4 GitHub

I løpet av prosjektet brukte vi GitHub for å utvikle programvare. Alle filene for prosjektet vårt ble lagret i repository. Vi kan få tilgang til dem ved bruk av en unik URL. Vi hadde to repository; en for frontend og en for backend. I frontend repository lagde vi dev (develop) og “master brancher”. I dev branch “committed” vi alle endringer fra Visual Studio Code. Dersom disse “commitments” ikke hadde synlige “regressions” eller “bugs” slo vi de sammen i master branch.

GitHub gir et nettbasert grafisk grensesnitt, tilgangskontroll og flere samarbeidsfunksjoner. På backend brukte vi en repository som inneholder branches. På starten jobbet alle med sine oppgaver på forskjellige prosjekter i GitHub. Etterhvert skjønte vi at vi må slå sammen alle prosjektene til ett prosjekt. Hver gang vi ville lage en ny funksjon benyttet vi branch sin funksjonalitet. Når funksjonen ble ferdigkrevet og fungerte, så slo vi sammen den branchen med en master. Dersom vi gjorde en feil kunne vi gjøre en rollback og bruke den siste commit-en som fungerte. Vi hadde også gruppesamtaler før vi sammenslo, slik at vi ble enige om det på forhånd (Se vedlegg 6.9).

2.4 Prosjektfaser og arbeidsplan

Bachelorgruppen kom frem til at det var lurt å dele prosjektet i tre faser. En prosjektfase er en gruppe med relaterte operasjoner som resulterer i en eller flere leveranser. Vi jobbet kun med de to første fasene som varierte i stor grad med tanke på kompleksitet. Den tredje fasen er en videreutviklingsfase der vi kan plassere funksjoner og ideer som hadde vist frem potensielle i applikasjonen, og som andre kan jobbe videre med på sikt.

Figur 4 – Fremdriftsplan for prosjektet

2.4.1 Fase 1 - Læring, prosess og planleggingsfase

I begynnelsen av prosjektet tas det mange viktige beslutninger: Hvordan lagre data, hvilket programmeringsspråk skal vi velge, hvilke rammeverk skal vi bruke, og så videre.

Det var i første fase at vi skulle samle kravspesifikasjoner for prosjektet. I denne fasen jobbet bachelorgruppen med å opprette et design for prosjektet og bygge prototypen. Vi bestemte oss også for hvilket verktøy vi skulle jobbe med for utvikling og planlegging.

Gruppen bestemte seg å bruke React fremfor Angular, selv om ingen av oss hadde erfaring med noen av dem. Vi leste om begge rammeverkene før vi kom til en beslutning. Valget vi tok var basert på etterspørsel av bestemt teknologi, arkitektur, læringskurve, template og obstruksjon. React har dobbelt så mange søker på Finn.no enn Angular. React bruker JavaScript som vi er kjent med, mens Angular følger med Typescript. Angular har en bratt læringskurve sammenlignet med React. Men React lar oss enkelt lære og lage en app i React økosystemet hvis en mestrer JavaScript.

Videre kom gruppen også frem å bruke .Net Core for backend og ikke .Net. Det var ikke alle i gruppen som var kjent med .Net Core så vi kjøpte et kurs på Udemy og tok det sammen.

Andre oppgaver som ble gjennomført i denne fasen var å lage design for webapplikasjonen, wireframe og flytskjema, brukerteste prototypen, lage UML - diagrammer og sekvensdiagrammer.

2.4.2 Fase 2 - Bygge Minste brukbare produkt (MBP) og stabilisere den

I den andre fasen av prosjektet jobbet vi med selve webapplikasjonen både på frontend og backend. Gruppen jobbet blant annet med å lage databasen og forskjellige funksjonaliteter for applikasjonen. Noen av funksjonalitetene omhandlet pålogging for administrasjon, registrering av brukere og lage, oppdatere og slette kurs.

Andre funksjonaliteter vi jobbet med var å lage Quiz til kurs, opprettelse av diskusjon på en bestemt side av kurset og nedlastning av video og tekst til kurs. I denne fasen jobbet vi også med testing (unit, integrasjon, system) og stabilisering av minste brukbare produkt (MBP).

2.4.3 Fase 3 - Videreutvikling av webapplikasjonen

Webapplikasjonen vår kan videreutvikles. Den kan få lagt inn flere funksjonaliteter som å spore og lagre brukerens progresjon i en kurs. Andre kan videreutvikle rettighetene til en administrator (admin), for eksempel at en admin kan samarbeide med en annen admin og lage et felles kurs. Andre funksjonaliteter som kan videreutvikles er blant annet å ha en admin- eller en brukerprofil, og laste ned presentasjonene til PowerPoint.

2.4.4 Ansvarsområder

Da vi begynte prosjektet i januar hadde vi en idé om hva prosjektet skulle handle om, men det tok noen møter før vi bestemte hvordan prosjektet skulle se ut og hva som skulle bli inkludert i hovedfunksjonaliteten. På dette tidspunktet ble det også relevant å sette tydelige ansvarsområder på høyere nivå for å gjøre arbeidet enklere. Elias ble ansvarlig for design, Alina for FrontEnd og Alex, Amina og Silje for Backend. Underveis i prosjektet ble også ansvarsområdene fordelt slik:

- Elias: Wireframes, brukertesting, prototyping og design, hovedansvar for rapporten.
- Alina: Frontend, risikovurdering, fremdriftsplan, wireframe, Scrum master, rapporten (frontend, prosessdokumentasjon).
- Silje: Databasen, publisering av API-en i Azure, autentisering og autorisasjon, rapporten (backend).
- Amina: Database, CRUD funksjonalitet, rapporten (backend)
- Alex: Skissering av database diagram og deretter implementere den i API, rapporten (backend)

2.4.5 Risikovurdering

Enhver prosjektaktivitet kan ha flere risikoer. Teamets evne til å forutsi hvilke trusler som kan oppstå, iverksette risikoreduserende tiltak, og planlegge det nødvendige arbeidet i tilfelle noe skulle skje, er veldig viktig for prosjektets fremdrift og suksess.

Risikovurderingen av prosjektet ble gjennomført når vi oppretter [HØ1] et prosjekt og hver gang det var betydelige endringer i omfanget. Vi bruker ikke kostnader for dette prosjekt. Derfor er risiko for oss, først og fremst, en ikke planlagt hendelse som påvirker varigheten av et prosjekt for å utvikle et system.

For å vurdere konsekvenser og sannsynlighet brukte vi “planning poker”.⁷ Planning Poker er en konsensusbasert teknikk for estimering. Det er en variant av Wideband Delphi-metoden. Med risiko-poker bruker vi listen med risiko og hver av gruppemedlemmene estimerer risiko basert på sin egen ekspertise. Etter første runde forklarer vi bakgrunn for estimering og spiller en gang til. Slik prøver vi å finne konsensus for et estimeringsnivå.

Riskopoeng og klassifikasjon på forskjellige nivåer fremstilles i tabellen nederst.

		Konsekvenser				
		1 Ubetydelige	2 Mindre	3 Alvorlige	4 Meget alvorlige	5 Katastrofale
Sannsynlighet	5 Ofte	5	10	15	20	25
	4 Sannsynlig	4	8	12	16	20
	3 Sjeldent	3	6	9	12	15
	2 Usannsynlig	2	4	6	8	10
	1 Meget usannsynlig	1	2	3	4	5

Figur 5 – konsekvensanalyse

I risikovurderingstabellen beskriver vi hva slags risiko som kunne oppstå, samt konsekvenser og sannsynlighet for hver av dem. Deretter satt vi opp aksjoner som vi burde gjøre for å unngå beskrevet risikoer.

Risikovurdering

⁷ Visual Paradigm, no date

Risiko	Konsekvens	Sannsynlighet	Risikopøeng	Aksjoner
Uklar oppgave fra klient	Alvorlig	Sjeldent	9	Tydelig overordnet mål
Mangel på motivasjon	Alvorlig	Sjeldent	9	Kommunisere med hverandre kontinuerlig, ba om hjelp, ta pauser
Gruppefravær	Meget alvorlig	Sjeldent	12	Være strenge på fravær
Dårlig planlegging	Meget alvorlig	Sannsynlig	16	Være realistiske, dokumentert planlegging, gjennomførbare oppgaver
Sykdom	Alvorlig	Sjeldent	9	
Begrensninger pga rammeverk/ tekniske løsninger	Katastrofal	Sannsynlig	20	Research på det som skal tas i bruk
Gruppekonflikt	Meget alvorlig	Sannsynlig	16	Holde på god stemning, høre på det som man har å si
For høye ambisjoner	Mindre	Sannsynlig	8	Ikke være altfor optimistisk, naiv. Gjøre research før man starter implementasjon
Utilstrekkelig kompetanse	Katastrofal	Sannsynlig	20	Kartlegge hva man kan. Lære seg nye ting
Oversikt over oppgaver prosgresjon	Alvorlig	Sannsynlig	12	Følge scrum rutiner
Dårlig kommunikasjon	Meget alvorlig	Sjeldent	12	Kommunisere med hverandre kontinuerlig, respektere hverandre, være tilgjengelig

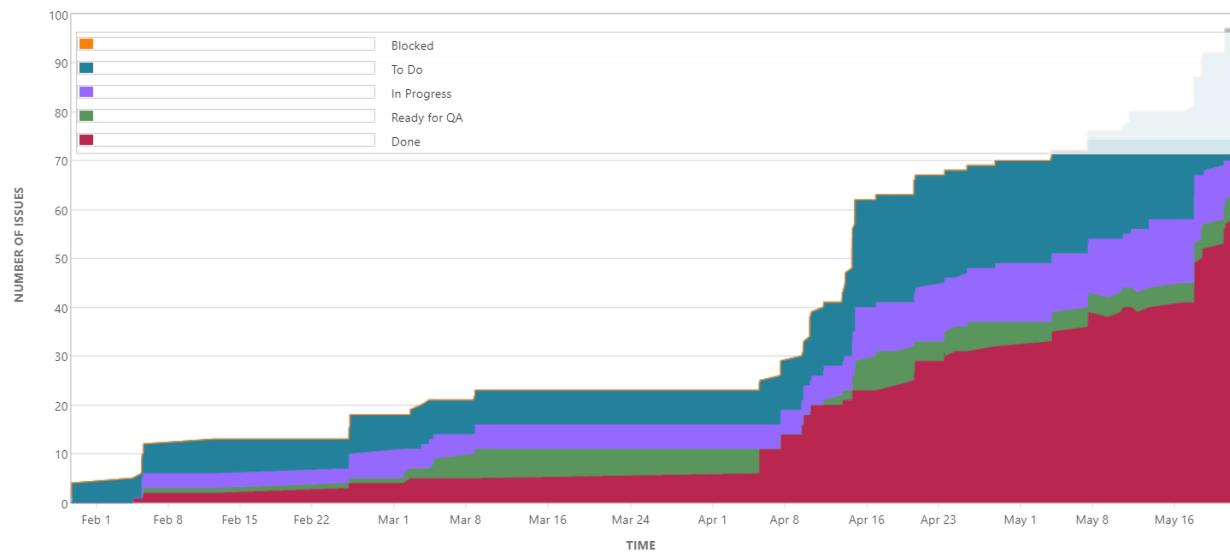
2.4.6 Sprinter

Siden skolen stengte i mars på grunn av korona krisen, så måtte vi omstrukturere de sprintene. Vi kunne ikke møtes lenger for å jobbe sammen, så måtte vi jobbe hver for oss og hjelpe hverandre remotely. Vi begynte å ha daglige møter via google meet for å vise hverandre hva vi hadde gjort og for å få hjelp dersom vi satt fast.

Cumulative Flow Diagram



29/Jan/20 to 21/May/20 (All Time) ▾ Refine report ▾



Figur 6 – Flytdiagram i Jira

Med Cumulative flow diagram er hovedformålet å vise hvor stabil flyten er. Det hjelper å forstå hvor vi trenger å fokusere for å gjøre prosessen mer forutsigbar. Det gir både kvantitativ og kvalitativ innsikt i både tidlige og eksisterende problemer. Diagrammet spører det totale antall oppgaver for hele prosjektet. Den horisontale aksen til CFD representerer tidsrammen som kartet visualiserer data for. Den vertikale aksen viser antall kort som er i arbeidsflyten på de forskjellige tidspunktene. De forskjellige fargede båndene som deler diagrammet av den oppadgående strømmen er de forskjellige stadiene i arbeidsflyten. Vi kan se om prosessen er stabil i bare ett øyeblikk ved å se på hvordan topp- og bunnlinjen i hvert bånd i det kumulative flytskjemaet skrider frem.

På den diagrammet kan vi se at nye oppgaver (to do) går inn i arbeidsflyten parallelt med de som forlater den (done). Dette viser at vi kan fokusere innsatsen på å forkorte syklustidene for oppgavene våre.

I begynnelse var det planlagt ni sprints. Hver sprint varer i 2 uker. På grunn av dårlig planlegging og mangel på erfaring, hadde vi avvik fra starts planen.

Sprint 1

Første sprint varte i 5 uker. Den var dedikert til læring av nødvendig base for å starte på prosjekt. Grunnen til at denne sprinten varte så lenge var blant annet at vi gikk gjennom noen kurs på Udemy som var veldig omfattende og tidkrevende. Det var masse som vi ikke lærte på studiet som vi måtte lære på egenhånd. Hovedmål for sprint 1 var å være klar med alle forberedelser og kurs for å begynne utviklingsprosess fra Sprint 2. I tillegg var det første sprint når begynte vi å bruke Jira som prosjektstyringsverktøy. Derfor var det ikke alle oppgaver rapportert eller beskrevet tydelig. I første sprint hadde vi bare ni oppgaver på board som var beskrevet ganske generelt. Fire av dem hadde medium prioritering, fire hadde den høyeste og en oppgave av en høy prioritet. Gruppen sluttet sprint med fire oppgaver som Done, tre

oppgaver var fortsatt i progress, wireframe var forberedt for QA og en oppgave som gjeld om separert JavaScript opplæring var i status “To do”.

Sprint 2

Sprint nummer to gjaldt begynnelsen på design og Frontend. I denne Sprinten begynte vi å jobbe også med Backend. Det var start på arbeid med rettigheter til admin og login funksjonalitet. Disse oppgavene var veldig omfattende derfor overførte vi dem videre til neste sprint. Det var masse som vi jobbet med her, men endret videre på neste sprint. Vi startet med å bruke identity for logg inn, men endret dette i neste sprint. Generelt ble det bedre med oppgave mengde, men gruppen fortsatt måtte lære seg å beskrive og å dele oppgaver i subtask slik at vi kunne ha bedre oversikt over utført arbeid. I sprint to hadde vi 14 oppgaver til sammen. Åtte av dem hadde den høyeste prioritet, 2 - høy og 4 - medium. Her forstår vi at videre bør vi dedikere en ticket bare til en oppgave.

Sprint 3

Tredje Sprint inkluderer oppgaver tilknyttet Frontend sider som er relatert til Admin. Det var også oppgaver som gjaldt å knytte API til FrontEnd. I denne sprinten hadde vi veldig mye fokus på logg inn funksjonalitet og admin rettigheter. Det var en ganske stor og omfattende oppgave. Vi måtte å gå gjennom kurs på Udemy igjen for logg inn, for å kunne lage denne funksjonaliteten. I tillegg begynte vi med rapporten. Det var arbeid tilknyttet til å finne god struktur til rapporten vår slik at vi kan passe på forskjellige arbeidsområder og starte med rapportskriving i neste sprint.

Sprint 4

I fjerde Sprint begynte vi å skrive mer på rapporten. Vi gikk fram med å beskrive løsningen, faser, verktøy og andre deler av prosjekt. Men stort sett rapport arbeid ble overført til neste sprint med status “in progress”. Vi utviklet også selveste kurs siden, altså oppretting av en kurs. Andre ting som ble gjort i denne sprinten var å bytte databaser for kurs fra Entity Framework til Dapper, lagde API for å se kurs, altså “lære mer”, API for sletting av kurs, lagde database for admin og brukere. For frontend delen ble endret login side, gjort funksjonalitet med google og facebook, bygget modul hovedside for Admin, tilsatt breadcrumbs.

Alt i alt hadde vi 29 oppgaver for sprint nummer fire. Selv om hadde vi så mye å gjøre, sprinten var ikke forleng og været 2 uker som det var planlagt. Men åtte av de oppgavene var overført til den neste sprint in “In progress” status. Og fem oppgaver var fortsatt i “to do” status på slutten av sprint.

I tillegg til 28 oppgaver ble det 7 oppgaver i sprinten som vi tok bort tilbake til backlog. Det var oppgaver tilknyttet til login funksjonalitet på frontend og backend delen ved hjelp av Microsoft Outlook og Google. Vi bestemt at den funksjonalitet har ikke høy prioritet for prosjekt og vi kan overføre den til det neste fase. Det var klart at Scopet var for stort. Så mange oppgaver som måtte vi overføre til neste sprint indikerte at det ble dårlig evaluering av innsats for hver oppgave. Men det er helt naturlig på grunn av mangel på erfaring med implementering av slike oppgaver. På retrospektivmøtet ble vi enige om at vi bør prøve å planlegge så mye oppgaver at det blir mulige å bli ferdig med de nesten alle planlagte i løpet av sprinten.

Sprint 5

Sprint 5 var vår siste utviklings sprint. Denne sprinten tok for seg fire uker. Til sammen hadde vi 38 oppgaver. Mesteparten av de var fullført.

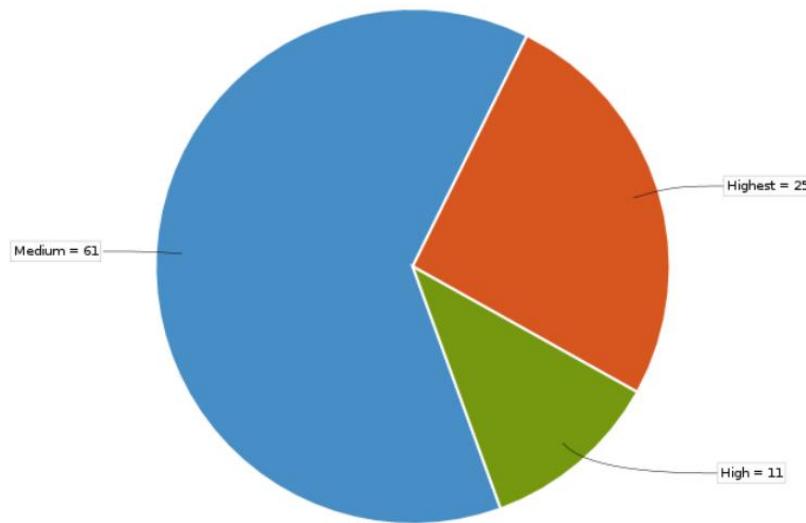
Oppgavene som hadde status “done” på slutten av sprinter er:

- Frontend (Authorisation, delete course, edit course, create course, admin video side, lage pop-up vindu med svarer for spørsmål, tilsetting data istedenfor statik),
- Report (om samarbeid mellom frontend og backend, oppdatering doc for databases i SQL server, beskrivelse hjemmeside, about side, database for bruker og kurs),
- Backend (lage roller, oppdatering av stored procedures i SQL server, publisering av API i Azure, implementasjon av Database Diagram to SQL Server Management Studio og Visual Studio, fixed database slik at alle kan ha tilgang til den, API for oppdatering av kurs, Create Course, lagde diagram for databasene).

I tillegg til det ble det gjennomført React kurs. Ingen av de oppgavene var i status “To do” på slutten av sprint. Seks oppgaver var “in progress”. Blant annet er det oppgaver som er tilknyttet rapportskriving. For eksempel, beskrivelse av løsningen, login-funksjonalitet, publisering i Azure, presentasjon av kunde og gruppe. En rekke oppgaver var ferdig for testing (API på frontend, Router).

Sprint 6

Sprint nummer seks var dedikert til rapportskriving og testing. I løpet av den sprinten hver av oss prøvde å skrive sin egen del. Etterpå hadde vi lange felles møter for å gå gjennom hvert eneste kapittel. Systemtesting var en del av hoveddeler av den sprint. Gjennom testing fant vi forskjellige problemer som gruppe måtte fikse fortløpende. Det var databasesøk og samspill mellom frontend og backend. Heldigvis løste vi det ganske fort.



Figur 7 – Opgavefordeling basert på prioritet

På dette kakediagrammet kan man se fordelt oppgaver basert på prioritet. Da har vi 62% av de oppgavene som ble estimert som medium, 25% som høy og 11% som har den høyeste prioritet. Prioritering var estimert for noen oppgaver av selve utviklere baseres på sin egen erfaring og forståelse. For de andre oppgaver satt vi prioritet sammen med tanke på nødvendighet for det prosjektet.

Man kan se Sprint Jira board i vedlegg.

2.5 Utfordringer og refleksjoner

Det finnes mange forskjellige verktøy for planlegging og prosjektledelse. Vi kunne ikke velge med en gang hva skal vi jobbe med, men vi begynte med Trello og Jira, og Excel ble også brukt til dagboksnotater. Etterhvert i prosjektet droppet vi Trello. Da fortsatte vi med dagboka i Excel og sprint-planning i Jira. Nesten alle gruppemedlemmer har lite erfaring med å jobbe med sprinter og å følge et Jira board. Derfor hadde vi problemer med estimering og arbeidsflyt.

I starten hadde vi ikke avklart hvem som skulle være prosjektleder og hva prosjektlederen skulle gjøre. Vi tok ikke på alvor at prosjektlederens ansvar var viktig. Derfor ble det vanskelig med kommunikasjon og generelt planlegging av hele prosjektet. Vi forstod at vi heretter måtte definere bestemte roller og ansvarsområder fra begynnelsen og følge de.

I Risikovurdering delen beskrevet vi hva slags risiko kan oppstå, konsekvenser og sannsynlighet for hver av dem. Også aksjoner som vi bør gjøre til å unngå beskrevet risikoer. Men den vanskeligste delen ble det å følge aksjoner med hver av teammedlemmene ellers er det ikke hjelpsom for prosjektet.

Det som vi har fått tak i er tydelig overordnet mål fra klient. Med de andre risiko hadde vi problemer. Selvfølgelig situasjonen med pandemi spilte stor rollen først og fremst med motivasjon. Vi hadde ikke gruppekonflikter men det var vanskelig å følge på gruppemedlemmer fravær. Vi planlagte med en tanke på å være realistisk men likevel hadde vi for høye ambisjoner, særlig i begynnelsen av web-applikasjon utvikling. Hadde vi også begrensninger på grunn av rammeverk eller tekniske løsninger, men det er grunn av den andre risiko som er utilstrekkelig kompetanse.

Problemene hadde vi også med oversikt over oppgaver progresjon og dårlig kommunikasjon. For mange var det vanskelig å følge Scrum rutiner og dagbok som omgjort til mangel på kontinuerlig kommunikasjon for hele teamet.

Første sprinter varte for lenge og ble dårlig beskrevet. Alle oppgavene var ikke registrert i Jira, og vi hadde ikke definert Scrum master. Dette førte til at vi hadde ikke Scrum rutiner i begynnelsen. I tillegg estimerte vi ikke hver oppgave. Derfor noe sprinter inneholder for mye oppgaver for to uker. Og flyttet vi de til nye sprinter. Det er ikke kritisk men det er tegn på dårlig planlegging som gikk ikke bra oversikt over velocity.

2.6 Oppsummering av prosessdokumentasjon

Utviklingsprosessen har vært preget av en bratt læringskurve i det bachelorgruppen satt seg ut for å innhente kunnskap om en rekke nye teknologier, kodespråk, metodikker og utviklingsplattformer. Gruppen har vært drevet av å lære så mye som mulig og har sett på det som god reell arbeidserfaring.

Prosessen har også vært preget av at rammeverket rundt systemutviklingen har blitt gjort internt i gruppen. Det betyr at vi ikke har hatt et konsulenthus, oppdragsgiver, veileder eller lignende som har hjulpet oss med å integrere en utviklingsmetodikk, viktige dokumenter, nødvendig tekniske plattformer eller lignende kompetanse som hadde kommet godt med. På grunn av dette gikk mye av tiden til å forberede utviklingsperioden som igjen påvirket hvor mye tid som gjensto til systemutviklingen. Bachelorgruppen sitter likevel igjen med en stor mestringsfølelse og mye god erfaring.

Gjennom utviklingsprosessen har gruppen omfavnet Scrum som utviklingsmetodikk. Gjennom metodikken har gruppen utdelt roller som Scrum master, produkteier og utviklingsteam. Gruppen har utviklet applikasjonen i intervaller kjent som sprinter der vi har gjennomgått metoder som Daily Scrum og sprint retrospektiv.

Bachelor gruppen har lært seg en rekke nye verktøy, der noen har blitt brukt til dokumentering av prosessen. Det har blitt skrevet dagbøker, registrert timer, laget fremdriftsplaner og risikovurderinger. En av de største jobbene har vært å holde styr på sprintene, her har det vært mye lerdøm, utfordringer og refleksjoner.

3 Produktdokumentasjon

3.1 Innledning

Formålet med produktdokumentasjonen er å gi leseren en nøyaktig og teknisk gjennomgang av hvordan applikasjonen er bygd opp. I dokumentasjonen blir de tekniske aspektene beskrevet i tre deler.

- I den funksjonelle systembeskrivelsen blir wireframes, flytskjema og prototypene beskrevet.
- I frontend delen blir applikasjonens strukturer, funksjoner, komponenter, samarbeid med backend beskrevet.
- I backend delen blir applikasjonens arkitektur, database, APIer og lignende beskrevet.

Bachelorgruppen har valgt å dokumentere nøye på grunnlag av videreførelse av prosjektet. Potensialet i applikasjonen mener vi er stort, spesielt ved videreutvikling. Derfor er det tenkt at produktdokumentasjonen legger til rette for nye utviklere å ta over produksjonen.

Produktdokumentasjon kan leses som et eget dokument, men det anbefales å lese del 1 (Presentasjonen) først. Dokumentasjonen er også ment å lese av fagkyndige da det er mye tekniske begreper. Vedlagt ligger en terminologiliste som kan hjelpe til på veien.

3.1.1 Introduksjon til applikasjonen

Applikasjonen er bygd opp i to deler som ser like ut, men har forskjellige funksjonaliteter. Kjerneprogrammet er en kursplattform der brukere kan gå igjennom kurs i form av tekst, video, bilde, PDF, PowerPoint og quiz. Brukeren har mulighet til å velge mellom ulike tilgjengelige kurs på landingssiden som tar brukeren videre til informasjonssiden til valgt kurs. Ved valg om å starte kurs vil brukeren bli vist informasjon i en av de nevnte formatene over om gangen. Dette betyr at på en kursside kan det bare være et format, men et kurs består ofte av mange kurssider.

Den andre delen er administrator delen. Her kan en administrator opprette, redigere og slette kurs. Administratoren møter de samme sidene som brukeren, men med mulighet til å redigere landingssiden, informasjonssiden og selve kurset. Når en administrator oppretter et kurs kan vedkommende velge en av de nevnte formatene over til en kursside om gangen.

3.2 Funksjonell systembeskrivelse

3.2.1 Aktører

Administrator

Administrator har rettigheter til å opprette, redigere og slette kurs.

Produkteier/ content owner

Produkteier har administratorrettigheter og har innholdet til kursene.

Foreleser/ Lærer/ Instruktør

en annen person som driver med utdanning kan ha administratortilgang, men også brukertilgang og rettigheter til distribusjon og registrering av studenter.

Sluttbruker/ Elev

Sluttbrukeren er interessenter som skal ta kursene. Det kan være studenter, forelesere i de andre institusjoner som er interessert i temaet, migranter eller lignende.

Utviklere

Utviklere er aktører som skal vedlikeholde og videreutvikle webapplikasjonen.

3.2.2 Brukergrensesnitt

Brukergrensesnittet er sammensatt av sider for modul, innlogging, kursinformasjon, og selvet kurset.

På innloggingssiden kan brukeren logge seg inn. Brukere trenger ikke logge seg inn for å ta kurs, men det tillater systemet og spore fremgangen til brukere. Brukeren kan registrere seg eller logge seg inn med mail og passord eller Facebook.

The screenshot shows the 'Log in' page of a website. At the top, there is a dark header bar with the 'OSLOMET' logo on the left and 'Modules', 'About', and 'Login' links on the right. Below the header, the main content area has a light gray background. It features a 'Log in' heading at the top center. Below it are two input fields: 'Email' and 'Password', each with its own input box. Underneath the password field is a small checkbox labeled 'Check me out'. At the bottom of this section are two yellow buttons: 'Submit' on the left and 'Register' on the right. A horizontal line separates this from the next section. Below the line, the text 'or signup with' is followed by a blue button with the white text 'SIGN IN WITH FACEBOOK'.

Figur 8 – innloggingssiden

Det første brukeren ser er brukersiden og en oversikt over alle moduler. De er stilt opp ved siden av hverandre som kort (cards). Hvert kort inneholder et bilde som er representativt for temaet, tittelen til modulen, informasjon om hvem som har laget modulen, hvor lenge modulen varer og en "learn more"-knapp. I menylinjen øverst på siden er det knapper som tar deg til en informasjonsside og en logg-inn/ registrer deg-side.

Select a module you would like to learn more about.

Erika Gubrium 60 min It will be name of the module Learn more	Aadne Gorstad 160 min It will be next name of the module Learn more	Alina Zielinska 120 min It will be next next name of the module Learn more	Alina Zielinska 120 min It will be next next next name of the module Learn more

About us | Contacts | Something | Personal info

Figur 9 - Modulsiden

Trykker brukeren videre på en modul kommer vedkommende til informasjon om modulsiden. Her står det skrevet en oppsummering av modulen, en innholdsfortegnelse for modulen og informasjon om lengde, hvilken institusjon modulen kommer fra, pris og språk. Det er også et bilde og informasjon om kurslederen. Øverst til høyre er den en stor og synlig knapp der det står "Start the course".

Header for the module

SContrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32. The standard chunk of Lorem Ipsum used since the 1500s is reproduced below for those interested. Sections 1.10.32 and 1.10.33 from "de Finibus Bonorum et Malorum" by Cicero are also reproduced in their exact original form, accompanied by English versions from the 1914 translation by H. Rackham.

[Start the course](#)

Duration	30 min
Institution	OsloMet
Subject	Social welfare
Price	Free
Language	English

Course topics

Number	Name	Duration
1	The Norwegian welfare state	4 min
1	The Norwegian welfare state	4 min
1	The Norwegian welfare state	4 min

Elias
lector
OsloMet

Alina Zielinska
lector
OsloMet

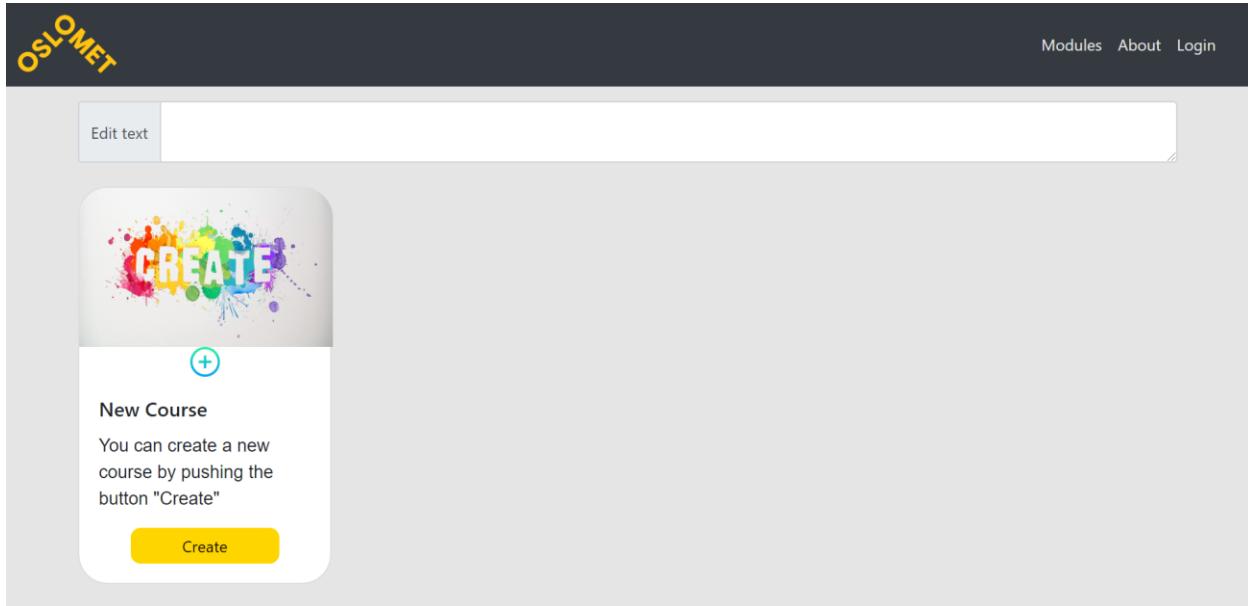
Bob Marley

Figur 10 – Hovedsiden for modul

Når brukeren starter en modul så er det en pagination øverst på siden. Her får brukeren oversikt over antall sider det er totalt for kurset. Dette kan videreutvikles med at brukeren kan logge inn og ha oversikt over hvilke sider hen har gjennomført og hvilken side vedkommende er på. Gjennom modulen er det fem forskjellige formater (tekst, video, pdf, PowerPoint og flervalgsspørsmål).

3.2.3 Administrasjonsgrensesnitt

Administratoren møter modul-siden der alle tilgjengelig og utilgjengelige modulene er stilt opp ved siden og under hverandre. Ved å trykke på en modul kommer administrator inn i informasjonsdelen av den valgte modulen.



Figur 11 - Hovedside administrator

Her har administrator mulighet til å redigere tittel, innholdsfortegnelse, informasjon om kurset og annet innhold.

The screenshot shows the 'Modulside administrator' interface. At the top left is the 'OSLOMET' logo. At the top right are links for 'Modules', 'About', and 'Login'. Below the header, there's a form for adding a course name and a large text area for a brief description. To the right of the description area are fields for 'Duration', 'Institution', 'Subject', 'Price', and 'Language', each with a corresponding input field. A yellow 'Add a content' button is located at the top right of the description area. Below the description area is a table for the course schedule with columns for 'Number', 'Name', and 'Duration'. At the bottom of the schedule table are three buttons: 'Publish', 'Delete', and 'Save'. A yellow 'Download schedule' button is positioned above the schedule table.

Figur 12 – Modulside administrator

Videre har administrator mulighet til å legge til eller redigere innhold. På samme måte som brukeren nавigerer seg videre inn til kurset, trykker administrator på den samme knappen som tar vedkommende inn til kurset. Her kan administrator legge til innhold i fem forskjellige formater som nevnt ovenfor (tekst, video, pdf, PowerPoint og flervalgsspørsmål).

The screenshot shows the 'Quiz-side administrator' interface. It features two identical question-and-answer entry forms. Each form has a top section for the question text ('Type in your question') and a bottom section for the answer and explanation ('Type in answer for the question and explanation'). Between the two forms is a small blue plus sign icon. The background of the page is light grey.

Figur 13 Quiz-side administrator

3.2.4 UI-utforming

Ved start av prosjekt begynte vi med tre hovedstadier i design av brukergrensesnitt:

- Brukertesting
- Design og prototyping
- Evaluering

Brukertesting beskrives i kapittel 3.4.1. Delkapittelet beskriver brukertesting fra et teoretisk perspektiv og våre valg som viker fra teorien.

Design og prototyping begynte vi med sketch - et enkelt og raskt skjema til å se hvordan denne eller den funksjonaliteten vil se ut. Dette lagde vi ved hjelp av en blyant og et papir. Etter det startet vi på wireframe (vedlegg 1.6). Det er utformingen av grensesnittet ved hjelp av enkle og grunnleggende elementer. Wireframe knyttet vi med flytskjema. Flytskjema er en måte å vise alle algoritmer eller gjensidighet prosesser for hele designet. Her brukte vi lucidchart som verktøy. Denne grunnleggende struktur og hovedfunksjonene ble fulgt av Frontend Utviklere.

Flytskjema med Wireframe av Hjemmeside og Modul Hovedside.

I Figma lagde vi wireframe-prototype (se vedlegg 6.7). Slik kunne vi vise simulering av den endelige gjensidigheten mellom brukeren og grensesnittet.

I Figma man kan også se mockups som vi lagde for prosjekt. Det ble en faktisk grafisk design som skulle brukes for applikasjonen. Men på grunn av at oppdragsgiver hadde ikke strikt design visjon følger vi ikke mockups detaljert.

En evaluatingsdesign er en struktur laget for å produsere en objektiv vurdering av fordelene med et program. Gjennom evaluatingsdesign gjorde vi avgjørelsen baseres på ressursene som er tilgjengelige og av grad av presisjon som er nødvendig.

3.3 Applikasjonens oppbygging

For å gjøre prosessen med å utvikle og vedlikeholde en webapplikasjon enklere og mer effektivt bygger vi et prosjekt med tanke på en arkitektur som er lettere å utvide og endre, samt teste, feilsøke og forstå. Som grunnlaget for arkitekturen i vårt prosjekt valgte vi å dele den i flere deler, det vil si at vi jobbet med forskjellige delsystemer som funksjonelle moduler, tjenester, lag og underprogrammer, samtidig som vi organiserte interaksjonen mellom disse.

Hovedmål for arkitekturen vår er:

- Skalerbarhet - muligheten til å utvide systemet og øke produktiviteten ved å legge til nye moduler.
- Vedlikehold - å endre en modul krever ikke endring av andre moduler og er lettere å forstå og vedlikeholde
- Modulbytte - modulen er enkel å bytte ut med en annen
- Mulighet til å teste (Unit Testing) - modulen kan kobles fra alle andre og testes / repareres
- Gjenbruk - modulen kan gjenbrukes i andre deler av programmet

Først delte vi systemet i store funksjonelle moduler / delsystemer (Brukeren og Admin) som beskriver business logikken. Oppdelingen i moduler / delsystemer gjøres basert på oppgavene som systemet løser.

Hovedoppgaven er delt inn i mindre under oppgaver, som kan løses/ utføres uavhengig av hverandre. Hver modul er ansvarlig for å løse et underproblem og utføre den tilhørende funksjonen. I tillegg til det funksjonelle formålet er modulen også preget av et sett med data, som er nødvendige for at den skal utføre sin funksjon. Det vil si: *Modul = Funksjon + Data* som er nødvendige for implementering.

3.3.1 Frontend

3.3.1.1 Oppstartning med React/ struktur av filer

Frontend delen skrives i Visual Studio Code ved hjelp av React og Bootstrap komponenter.

Vi brukte JSX, en dialekt av JavaScript utvidet med syntaktiske konstruksjoner. Først lagde vi en ny prosjektkatalog med tre filer: App.css, App.js og Index.html-fil. I index.html brukte vi ReactDOM for å vise hva koden fra <App/> representerer på siden. Deretter laster vi komponenten som en egenskap av React, så vi trengte ikke lenger utvide React.Component.

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import { BrowserRouter } from 'react-router-dom';
//testing

ReactDOM.render(<BrowserRouter basename="http://localhost:3000"><App /> </BrowserRouter>, document.getElementById('root'));
```

Vi har laget en komponent - App-komponenten. Nesten alt i React består av komponenter, som kan være klassekomponenter eller funksjonellekomponenter. Vi har mange små komponenter og alt lastes inn i hoved App-komponenten. Komponenter får også ofte sin egen fil.

The screenshot shows a code editor with two panes. The left pane displays a portion of a React component's code, specifically the `App` component. The right pane shows the project structure of a folder named `BACHELOR_2`. The structure includes `.vscode`, `node_modules`, `public`, and a `src` directory containing various files and components like `AboutPage`, `Assets`, `Data`, `Homepage`, `Layout`, `LoginPage`, `ModulPage`, `ModulPageAdmin`, `Services`, `utils`, and several CSS files. It also includes configuration files like `.gitignore`, `package-lock.json`, `package.json`, and `README.md`.

```

export default class App extends Component {
  render() {
    const sessionToken = localStorage.getItem("sessionToken");
    return (
      <Layout>
        <Switch>
          <Route exact path="/" component={CardModules} />
          <Route path="/login" component={LoginPage} />
          <Route path="/modules" exact component={CardModules} />
          <Route path="/about" component={AboutPage} />
          <Route
            exact
            path="/modules/:moduleId"
            render={(props) => <ModulForm {...props} />}
          ></Route>
          <Route exact path="/modules/:moduleId/lessons/:lessonId">
            <ModulDetailLesson />
          </Route>
          <Route exact path="/modules/:moduleId/finish">
            <Finish />
          </Route>
          <Route path="/admin" component={CardModulesAdmin} />
          <Route path="/create" component={AdminModulMainPage} />
          <Route exact path="/modules/:moduleId/edit" component={AdminModulMainPage}>/>
          <Route path="/textAdmin" component={AdminModulDetailPageText} />
          <Route path="/videoAdmin" component={AdminModulDetailPageVideo} />
          <Route path="/quizAdmin" component={AdminModulDetailPageQuiz} />
        </Switch>
      </Layout>
    );
  }
}

```

I prosjektstrukturen kan vi se en `/public` og `/src`-katalog, sammen med de vanlige `node_modulene`, `.gitignore`, `README.md`, `package-lock.json` og `package.json`.

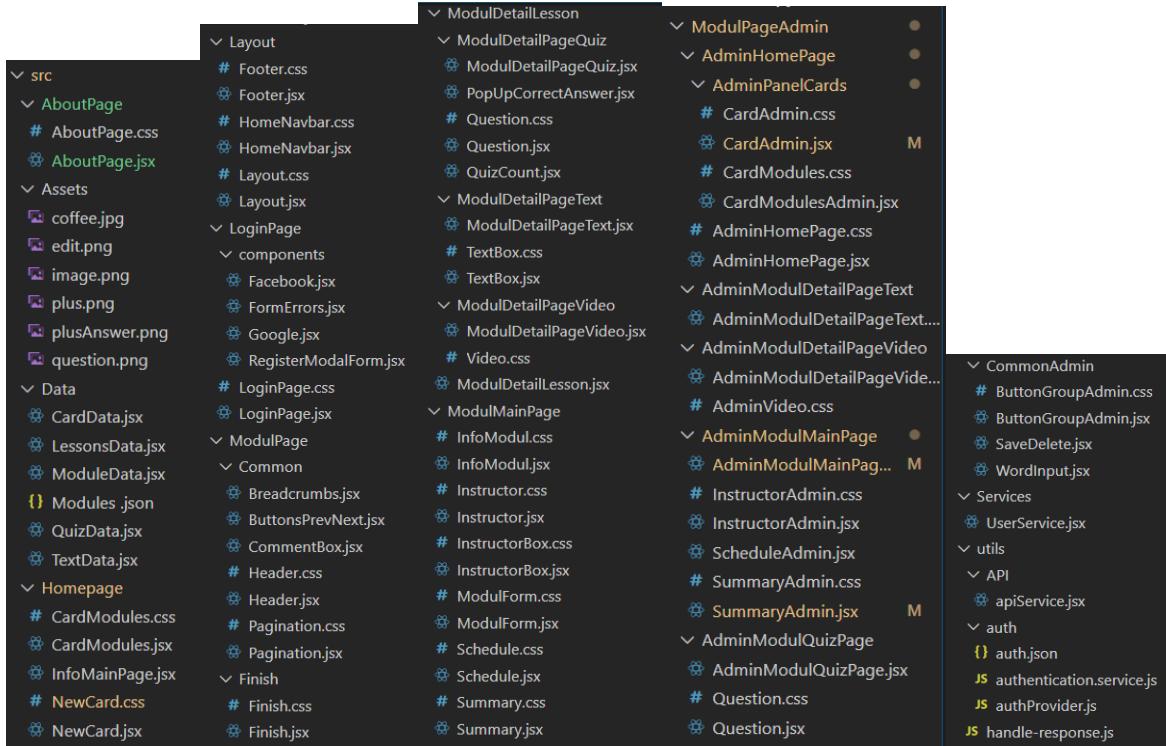
`package.json` er filen som brukes av npm til å administrere pakkeavhengigheter og versjoner for lokalt installerte pakker.

```

{} package.json > {} scripts
1  {
2    "name": "wmx-bachelor",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "formik": "^2.1.4",
7      "yup": "^0.28.4",
8      "rxjs": "",
9      "bootstrap": "^4.3.1",
10     "connected-react-router": "6.5.2",
11     "history": "4.10.1",
12     "jquery": "^3.5.0",
13     "merge": "1.2.1",
14     "popper.js": "^1.16.0",
15     "react": "16.11.0",
16     "react-bootstrap": "^1.0.1",
17     "react-dom": "16.11.0",
18     "react-facebook-login": "^4.1.1",
19     "react-google-login": "^5.1.10",
20     "react-redux": "7.1.1",
21     "react-router": "5.1.2",
22     "react-router-dom": "5.1.2",
23     "react-scripts": "^3.4.1",
24     "reactstrap": "8.1.1",
25     "redux": "4.0.4",
26     "redux-thunk": "2.3.0",
27     "svgo": "1.3.0"
28   },
29   "scripts": [
30     "start": "react-scripts start",
31     "startHTTPS": "HTTPS=true react-scripts start",
32     "build": "react-scripts build",
33     "test": "react-scripts test",
34     "eject": "react-scripts eject",
35     "auditfix": "npm audit fix"
36   ],
37   "eslintConfig": {
38     "extends": "react-app"
39   },
40   "browserslist": {
41     "production": [
42       ">0.2%",
43       "not dead",
44       "not op_mini all"
45     ],
46     "development": [
47       "last 1 chrome version",
48       "last 1 firefox version",
49       "last 1 safari version"
50     ]
51   }
52 }
53

```

Src inneholder alle filene som inneholder komponenter til forskjellige deler av applikasjonen, data som ble brukt før API var klar for bruk og bilder.



3.3.1.2 Bootstrap

I dette prosjektet brukte vi React Bootstrap for Frontend delen. Alle komponenter ble stilisert med CSS hvis det var nødvendig. For hver eneste fil ble bestemte Bootstrap APIer importert.

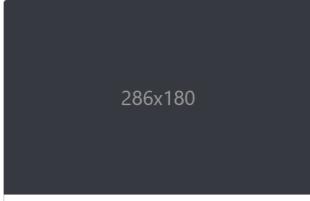
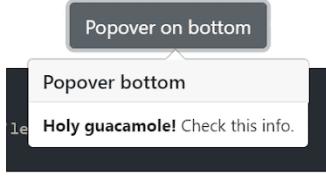
```

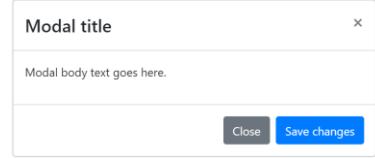
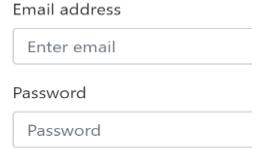
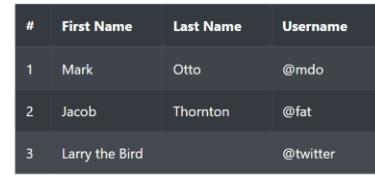
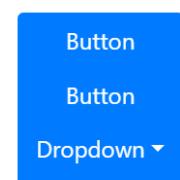
1 import React, { Component } from "react";
2 import "./NewCard.css";
3 import Card from "react-bootstrap/Card";
4 import Button from "react-bootstrap/Button"; | API fra Bootstrap
5 import {
6   BrowserRouter as Router,
7   Switch,
8   Route,
9   useParams,
10  useRouteMatch,
11 } from "react-router-dom";
12

```

For å gjøre design responsive brukte vi Grid system.

I tabellen fremstiller vi mesteparten av elementer som ble brukt i prosjektet fra Bootstrap, i tillegg til navbar og knapper:

<p><Card.Body> hovedelement på HomePage som inneholder kort informasjon om Modul</p>  <p>Card Title Some quick example text to build on the card title and make up the bulk of the card's content. Go somewhere</p>	<p><Popover> brukes som hjelpemiddel på AdminSide</p>  <p>Popover on bottom Popover bottom Holy guacamole! Check this info.</p>	<p><Form.Check> ble brukt på Quiz side</p> <ul style="list-style-type: none"> <input type="radio"/> first radio <input type="radio"/> second radio <input type="radio"/> third radio
<p><Breadcrumb> brukes til å ha brukervennlig navigasjon på nettsiden</p>  <p>Home / Library / Data</p>	<p><Form.File> gjennom den formen kan brukeren laste fil opp på video AdminSide</p>  <p>Custom file input <input type="button" value="Browse"/></p>	<p><Pagination> brukes for navigasjon inn i Modul</p>  <p>« < 1 ... 10 11 12 13 14 ... 20 > »</p>
<p><InputGroup> gjennom input group får vi navn og beskrivelse av Modul fra admin</p>	<p><Modal> brukes til registrerings form</p>	<p><Form.group> blir brukt for loggin side</p>

																		
<ListGroup> brukes for fremstille informasjon om forelesere på ModulHovedSide	<Table> for å gruppere timeliste	<ButtonGroup> for å fremstille forskjellige alternativer for å lage side for Moduler																
	 <table border="1"> <thead> <tr> <th>#</th> <th>First Name</th> <th>Last Name</th> <th>Username</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Mark</td> <td>Otto</td> <td>@mdo</td> </tr> <tr> <td>2</td> <td>Jacob</td> <td>Thornton</td> <td>@fat</td> </tr> <tr> <td>3</td> <td>Larry the Bird</td> <td></td> <td>@twitter</td> </tr> </tbody> </table>	#	First Name	Last Name	Username	1	Mark	Otto	@mdo	2	Jacob	Thornton	@fat	3	Larry the Bird		@twitter	
#	First Name	Last Name	Username															
1	Mark	Otto	@mdo															
2	Jacob	Thornton	@fat															
3	Larry the Bird		@twitter															

3.3.1.3 Side om prosjektet - Informasjonsside (About)

Denne siden er den enkleste i appen. Her bruker vi bare HTML og CSS uten noe ytterligere funksjonalitet. Ikoner til universiteter viderekobler til universitets nettsider. Det er en statisk side, men hvis kunden har lyst til å ha en permanent mulighet til å endre det, da kan vi lage den tilleggsfunksjonalitet for admin.

About the Project

Developed and led by experts in the fields of social policy, social work, and urban studies, this course provides an introduction to work and social inclusion of migrants in the comparative urban contexts of Bangalore, Moscow and Oslo.

In the course, we will use a visual and interactive approach to explore theories and methods for evaluating work inclusion measures, as well as the crucial strategies and factors that are often not considered. You will gain knowledge of how poverty may be understood and compared across the three settings and will learn alternative conceptual approaches to the more traditional income-focused one.



We will focus on:

- methods and theories for understanding and measuring poverty in comparative settings
- policy strategies and necessary considerations developing inclusion measures
- evaluation of current work inclusion measures using alternative approaches to a focus on income

The course will move between different levels of experiences shaping the success of work and social inclusion measures, including a biographical approach to the everyday life of the target group individuals, the factors of changing place and space before and after migration, as well as the structural and institutional factors shaping inclusion experiences. The concepts and ideas you will learn will help you to reflect upon current practices and improve practices for often marginalized communities.

You will learn new ways to evaluate the work inclusion strategies beyond those currently used. Our focus is on moving beyond an income-only focus in order to improve the social inclusion of target groups through work. Using a "situated" work inclusion approach, the course will provide an appreciation of the historical, social-material and psychological factors shaping the success of inclusion methods targeted to diverse and often marginalized populations and identities.



[About us](#) [Contacts](#) [Something](#) [Personal info](#)

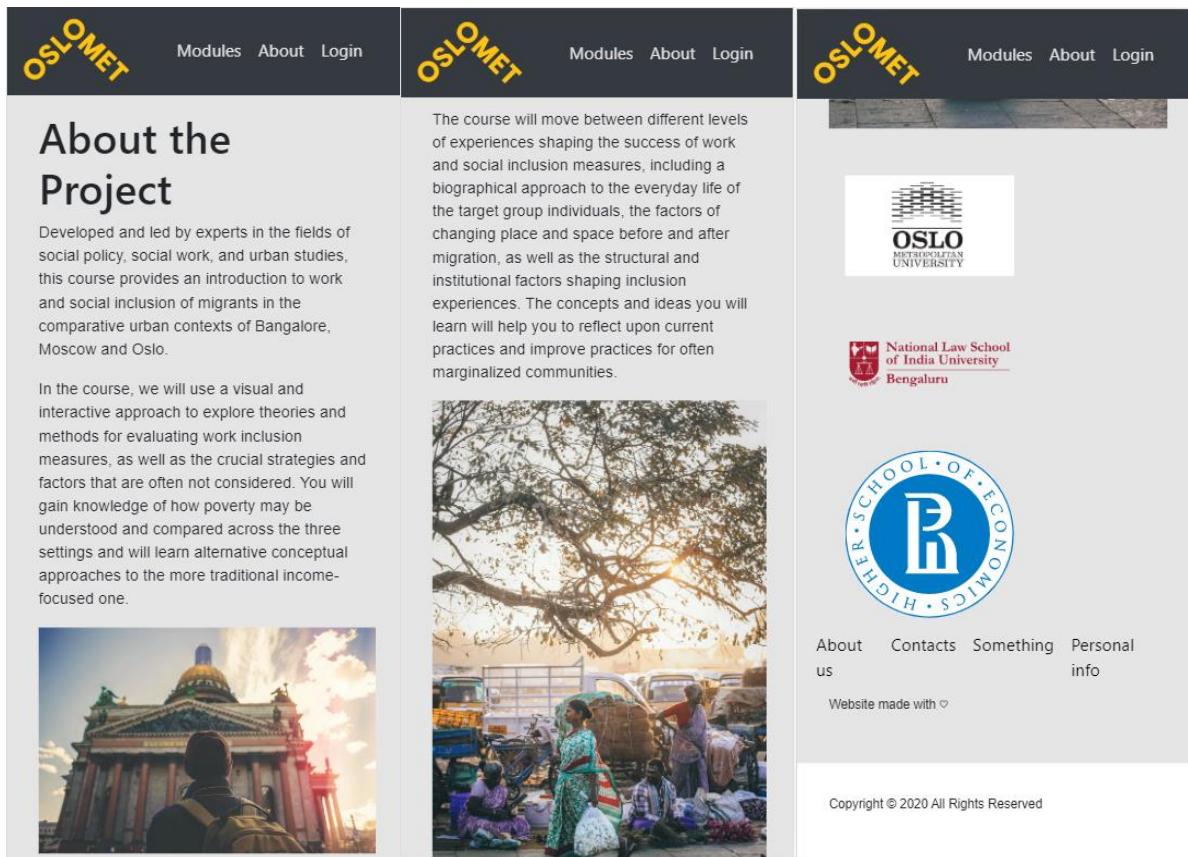
Figur 14 - Om-siden

Her bygget vi komponentet AboutPage i form av JavaScript-klassen som kalles komponentklassen. Den må inneholde en funksjon som kalles render (). Denne funksjonen

returnerer JSX-koden til komponenten.

```
export class AboutPage extends Component {
  render() {
    return (
      <Container>
        <Row>
          <Col xs={12} md={12}>
            <h1>About the Project</h1>
          </Col>
        </Row>
        <Row>
          <Col xs={12} md={8}>
            <p>
              Developed and led by experts in the fields of social policy, social work, and urban studies, this course provides an introduction to work and social inclusion of migrants in the comparative urban contexts of Bangalore, Moscow and Oslo.
            </p>
          </Col>
        </Row>
      </Container>
    );
  }
}
```

Bruk av Bootstrap Grid system gjør siden responsive.



Figur 15 – Om-siden mobilformat

Footer skrev vi i form av funksjonen (funksjonelle komponenter) fordi i dette tilfelle er vi interesserte i komponenter uten state (tilstand).

```

export function Footer(){
  return(
    <div className='footer'>
      <Navbar id="footer">
        <Nav className="mr-auto" >
          <Nav.Link href="#home">About us</Nav.Link>
          <Nav.Link href="#features">Contacts</Nav.Link>
          <Nav.Link href="#pricing">Something</Nav.Link>
          <Nav.Link href="#pricing">Personal info</Nav.Link>
        </Nav>
      </Navbar>

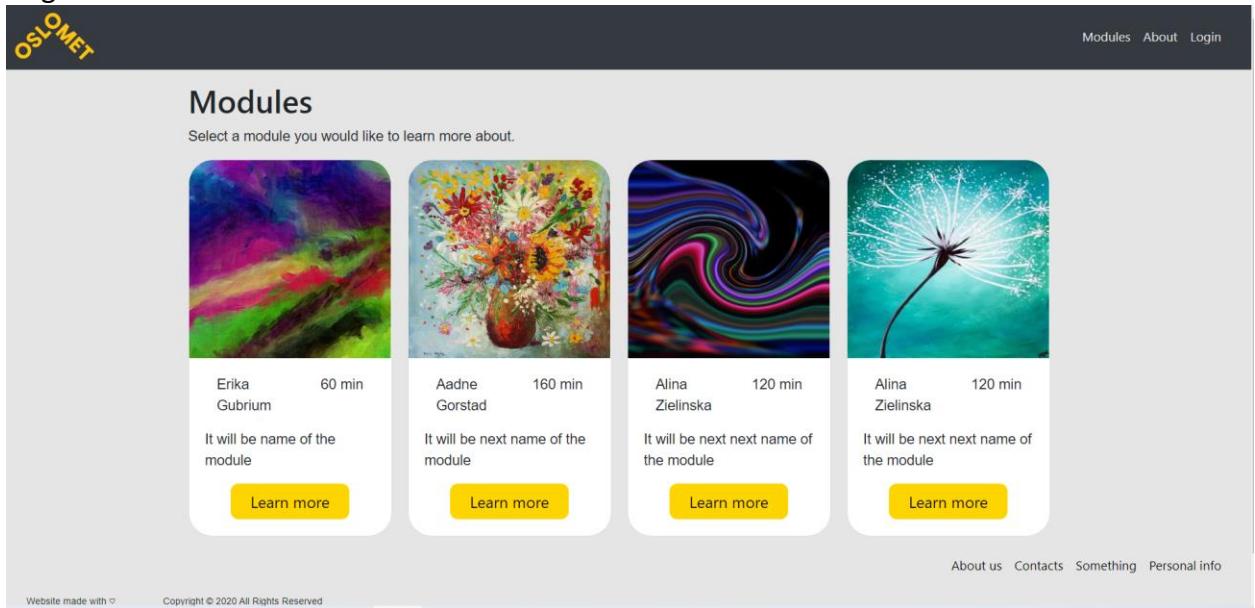
      <div className="personalInfo">
        <p>Website made with ♡</p>
        <p>Copyright © 2020 All Rights Reserved </p>
      </div>
    </div>
  );
}

```

3.3.1.4 Hovedside (HomePage)

Hovedside (HomePage) består av navbar på toppen med lenker til relevante sider, kort statisk informasjon for brukere, footer med lenker til sider som er ikke planlagt ennå. Derfor står det bare eksempel på noe som det kan være. For eksempel personvern eller kontakter eller info om utviklingsgruppe. Moduler i form av kort er de hovedelementene på siden.

På kortet står det informasjon om personen som er ansvarlig for et bestemt kurs (modul), varighet og moduls temaet.



Figur 16 – Modul hovedside

Informasjonen som står på kortene kommer fra databasen. I løpet av utviklingsfasen, før API ble ferdig, brukte vi håndlaget database.

```

src > Data > CardData.jsx ...
1  export const cardData =
2  [
3    {image: "https://upload.wikimedia.org/wikipedia/commons/thumb/5/52/Abstract_Art_2002.jpg/330px-Abstract_Art_2002.jpg",
4 lecturer: "Erika Gubrium",
5 timing:"60 min",
6 name: "It will be name of the module"},
7
8 {image: "https://i.etsystatic.com/17072610/r/il/4dc22f/1427703510/il_570xN.1427703510_kqct.jpg",
9 lecturer: "Aadne Gorstad",
10 timing: "160 min",
11 name: "It will be next name of the module"},
12
13 {image: "https://cdn.pixabay.com/photo/2015/12/20/00/32/art-1100519_1280.jpg",
14 lecturer: "Alina Zielinska",
15 timing: "120 min",
16 name: "It will be next next name of the module"},
17
18 {image: "https://i1.wp.com/loveyogastudios.com/wp-content/uploads/2019/12/dandelion.jpg?fit=480%2C600&ssl=1",
19 lecturer: "Alina Zielinska",
20 timing:"120 min",
21 name: "It will be next next name of the module"} ]

```

Den informasjonen ble sendt til et bestemt kort ved hjelp av mapping. For å iterere over en liste slik at den gjengir mer enn en komponent eller implementerer en betinget logikk, bruker vi rent JavaScript. Det meste av tiden, `map()` dekker behovene våre.

```

src > Homepage > CardModules.jsx > render
1  import React, { Component } from "react";
2  import { InfoMainPage } from './InfoMainPage';
3  import { NewCard } from './NewCard';
4  import { cardData } from '../Data/CardData';
5
6  export class CardModules extends Component{
7
8    render() {
9
10      return(
11        <>
12          <InfoMainPage/>
13          <div className='cards'>
14            {
15              cardData.map((x,y)=>{
16                return <NewCard cardData=[x]/>;
17              })
18            }
19          </div>
20        </>
21      );
22    }
23  }

```

Her bruker vi også mest populært metoden `render()` som er påkrevd og brukes i hver klasse. `render()` metoden sender HTML til DOM.

- Homepage
 - CardModules.jsx
 - InfoMainPage.jsx
 - NewCard.css
 - NewCard.jsx

Hovedsiden (HomePage) består av komponenter som ligger i tre filer. CardModules er ansvarlig for samling av alle komponenter, InfoMainPage inkluderer informasjon som står øverst ved moduler og NewCard er selv modul som

aksepterer kort info om kurs. Her bruker vi bootstrap komponent `<Card.Body>`.

- Layout
 - Footer.css
 - Footer.jsx
 - HomeNavbar.css
 - HomeNavbar.jsx
 - Layout.css
 - Layout.jsx

Navbar og footer ble lagret i en separat "Layout" mappe, fordi disse komponentene er felles for alle sider. Vi kaller på Layout gjennom App.jsx.

```

23  export default class App extends Component {
24    render() {
25      const sessionToken = localStorage.getItem("sessionToken");
26      return (
27        <Layout>
28          <Switch>

```

Ved å trykke på "Learn more" knappen, går man til hovedsiden av et bestemt kurs. Her bruker vi Router med Modul Id på App.jsx: <Route exact path="/modules/:moduleId">

Det får vi til ved å bruke useRouteMatch() funksjonen som matcher Url for side, og href hvor vi bruker url til å danne en ny lenke.

3.3.1.5 Modul hovedside

Number	Name	Duration
1	The Norwegian welfare state	4 min
1	The Norwegian welfare state	4 min
1	The Norwegian welfare state	4 min

Figur 17 – Modul hovedside

<pre> ▼ ModulMainPage # InfoModul.css ⚑ InfoModul.jsx # Instructor.css ⚑ Instructor.jsx # InstructorBox.css ⚑ InstructorBox.jsx # ModulForm.css ⚑ ModulForm.jsx # Schedule.css ⚑ Schedule.jsx # Summary.css ⚑ Summary.jsx </pre>	<p>Modul MainPage består av forskjellige komponenter som ligger i seks filer. ModulForm samler alle komponenter til å danne en side. InfoModul inneholder tekstdelen av Modul hvor man beskriver temaet av modul og alle momenter som er nødvendige for studenter å vite før de starter kurset.</p>
--	---

```

1 import React, { Component } from 'react';
2 import Listgroup from 'react-bootstrap/ListGroup';
3 import { InstructorBox } from './InstructorBox';
4 import './Instructor.css';
5
6
7 export class Instructor extends Component{
8   render (){
9     return(
10       <ListGroup>
11         {this.props.dataInstructors.map((item)=>{
12           return(<ListGroup.Item><InstructorBox item={item}/></ListGroup.Item> );
13         })}
14
15       </ListGroup>
16     );
17   }
18 }
19 
```

Instructor filen segregationer informasjon om forelesere som tar del i kurset og videresender til InstructorBox.

```

7 export class InstructorBox extends Component{
8 render (){
9   return (
10     <Container>
11       <Row>
12         <Col xs={5} md={4}>
13           <img className='instructorImg' src={this.props.item.img}/>
14         </Col>
15         <Col xs={7} md={8}>
16           <div className='instructorInfo'>
17             <p>{this.props.item.name}</p>
18             <p>{this.props.item.title}</p>
19             <p>{this.props.item.institution}</p>
20           </div>
21         </Col>
22       </Row>
23     </Container>
24   );
25 }
26 
```

InstructorBox tar inn dataene og fremstiller dem på side i browser.

```

5 export class Schedule extends Component {
6   render() {
7     return (
8       <Table responsive variant="dark">
9         <thead>
10           <tr>
11             <th>Number</th>
12             <th>Name</th>
13             <th>Duration</th>
14           </tr>
15         </thead>
16         <tbody>
17           {this.props.dataSchedule.map((item) => {
18             return (
19               <tr>
20                 <td>{item.number}</td>
21                 <td>{item.name}</td>
22                 <td>{item.duration}</td>
23               </tr>
24             );
25           .... });
26         </tbody>
27       </Table>
28     );
29   }
30 }
```

Schedule filen inneholder et komponent som tar inn data om temaer som skal presenteres i kurset, lengden og temaene nummer.

```

5  export class Summary extends Component {
6    render() {
7      console.log("propsSummary", this.props);
8      return (
9        <Table responsive>
10       <tbody className="summary">
11         <tr>
12           <td>Duration</td>
13           <td>{this.props.dataSummary.duration} min</td>
14         </tr>
15         <tr>
16           <td>Institution</td>
17           <td>{this.props.dataSummary.institution}</td>
18         </tr>
19         <tr>
20           <td>Subject</td>
21           <td>{this.props.dataSummary.subject}</td>
22         </tr>
23         <tr>
24           <td>Price</td>
25           <td>{this.props.dataSummary.price}</td>
26         </tr>
27         <tr>
28           <td>Language</td>
29           <td>{this.props.dataSummary.language}</td>
30         </tr>
31       </tbody>
32     </Table>
33   );
34 }

```

Klasse Summary tar inn data om kurs og fremstille den på siden i form av en tabell.

Før APlet ble ferdig, bruker vi dataene fra håndlaget kilde av data.

```

3  export const ModulData = {
4    title: "Header for the module",
5    description:
6      "Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature",
7    duration: "30",
8    institution: "OsloMet",
9    subject: "Sosial",
10   price: "free",
11   language: "English",
12   lectors: [
13     {
14       img: coffeeImg,
15       name: "Elias",
16       title: "lector",
17       institution: "OsloMet",
18     },
19     {
20       img:
21         "https://2w6kxc2rrr9mabqt1mg1gait6-wpengine.netdna-ssl.com/wp-content/uploads/2019/08/man-shrug-beard-1024x580.jpg",
22       name: "Alina Zielinska",
23       title: "lector",
24       institution: "OsloMet",
25     },
26     {
27       img:
28         "https://2w6kxc2rrr9mabqt1mg1gait6-wpengine.netdna-ssl.com/wp-content/uploads/2019/08/man-shrug-beard-1024x580.jpg",
29       name: "Bob Marley",
30       title: "lector",
31       institution: "OsloMet",
32     },
33   ],
34   topics: [
35     { number: 1, name: "The Norwegian welfare state", duration: "4 min" },
36     { number: 1, name: "The Norwegian welfare state", duration: "4 min" },
37     { number: 1, name: "The Norwegian welfare state", duration: "4 min" },
38   ],
39 };
40 };
41 };
42 
```

De dataene sender vi på nivå av InfoModul:

```
59          <h4>Course topics</h4>
60          <Schedule id="schedule" dataSchedule={ModulData.topics} />
61      </Col>
62  |  <Col xs={12} md={4}>
63      <Container>
64          <Row>
65              <Button type="button" href={`${url}/lessons/001`}>
66                  Start the course
67              </Button>
68          </Row>
69          <Row>
70              <Summary dataSummary={ModulData} />
71          </Row>
72          <Row>
73              <Instructor dataInstructors={ModulData.lectors}/>
```

På denne siden ble bruket av Grid system fra Bootstrap veldig hjelpsom. Med dette systemet var det ganske enkelt å sette alle elementer på plass og å ordne responsive design.

OSLOMET

Modules About Login

Home / Library / Data

Header for the module

SContrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum

Course topics

Number	Name	Duration
1	The Norwegian welfare state	4 min
1	The Norwegian welfare state	4 min
1	The Norwegian welfare state	4 min

[Start the course](#)

Duration	30 min
Institution	OsloMet
Subject	Social welfare
Price	Free
Language	English

 Elias
lector
OsloMet

 Alina Zielinska
lector
OsloMet

 Bob Marley
lector
OsloMet

About us Contacts Something Personal info

Website made with  Copyright © 2020 All Rights Reserved

Figur 18 – Hovedside modul mobiloptimalisert

Ved å trykke på "Start the course" knappen omdirigerer vi brukeren til innholdet i det bestemte modulet (første side for kurset). For å gjøre det, slik at brukerne går til innholdet til det bestemt Modulet, så må vi gjenbygge

```
<Route exact path="/modules/:moduleId/lessons/:lessonId">
  <ModulDetailLesson />
</Route>
```

Router i App.jsx.

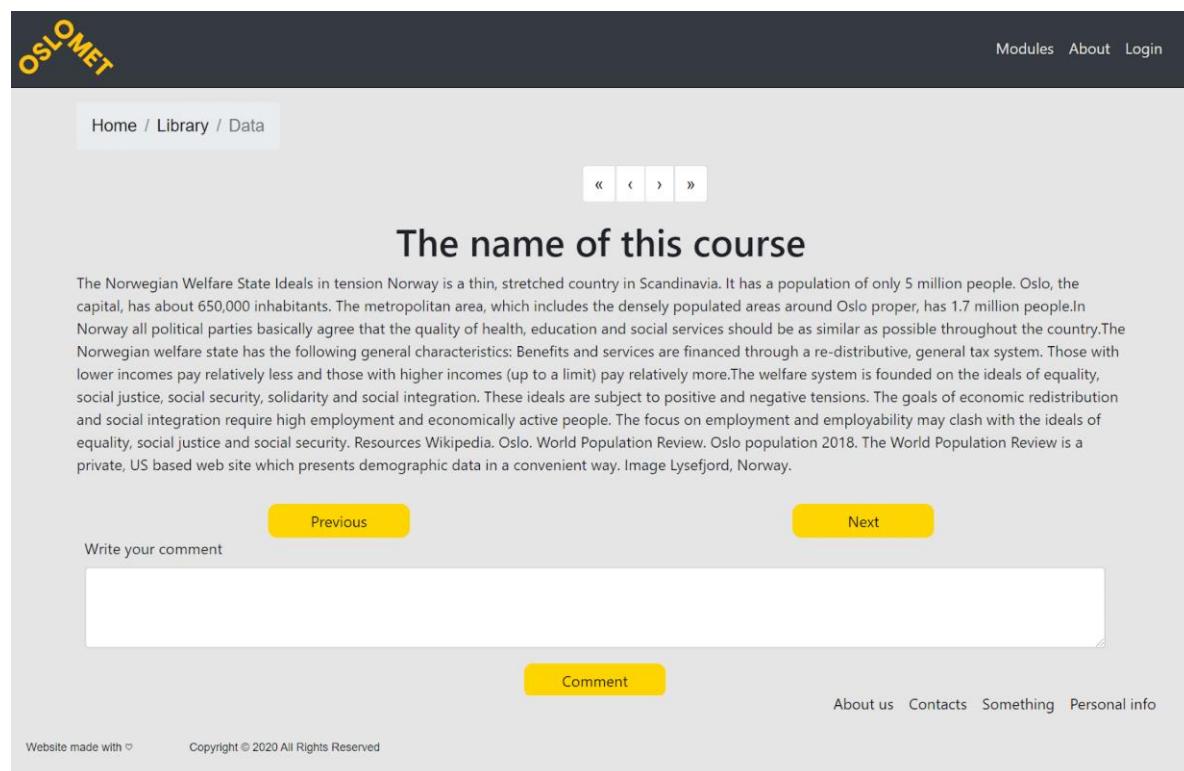
```
24 function ModulLessonContent(param) {
25   switch (param.type) {
26     case "text":
27       return <ModulDetailPageText info={param} />;
28     case "video":
29       return <ModulDetailPageVideo info={param} />;
30     case "quiz":
31       return <ModulDetailPageQuiz info={param} />;
32     default:
33       return "Modul not found";
34   }
35 }
```

Slik lager vi path til en bestemt leksjon av en bestemt modul, basert på deres Modul Id og Lesson Id. Det fikk vi til ved å bygge filen for ModulDetailLesson hvor vi bruker switch() funksjon til å sende brukeren til en bestemt side avhengig av bestemt case som tilsvarer en definert innholdstype (text, video eller quiz). Switch bruker vi til å utføre forskjellige handlinger basert på

forskjellige innhold. Når vi blir gjennom forskjellige sider av Modul (kurs) viser vi forskjellige sider avhengig av nøkkelen (course type).

Disse parametrene sender vi med funksjon withRouter const ShowTheLocationWithRouter = withRouter(ModulDetailLesson) - det vil si at vi lager en ny komponent som er "tilkoblet" til ruteren.

3.3.1.6 Modul lekse side med tekst



Figur 19 – Kurs tekstformat

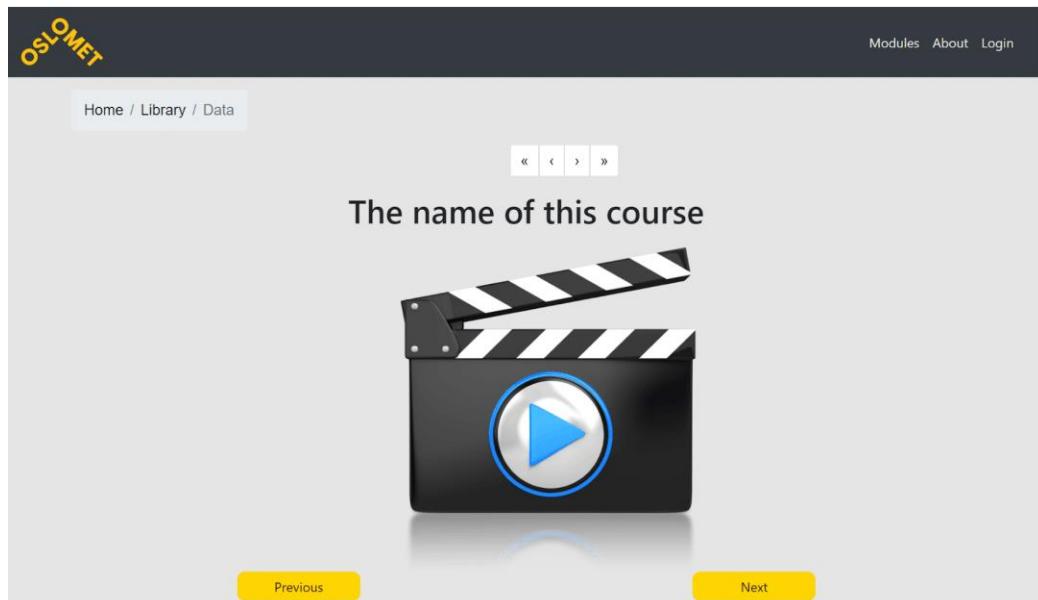
Modul sider har enkelt design. Hver side inneholder en leksjon eller et tema som fremstilles i et av de formatene (tekst, video eller quiz). Vi også foreslo til prosjekteieren å ha mulighet til å laste PowerPoint og PDF filer, men det var ikke prioritert. Derfor kan det gjøres i fase tre av utviklingsprosessen.

Det som må endres er formatering av teksten, som kommer fra backend. Det skal vi gjøre ved hjelp av Draft.js rammeverk. Om dette rammeverket snakker vi mer [her](#).

Ved hjelp av Next og Previous knapper kan man navigere til neste eller forrige sider i modul. Man kan også navigere i modul ved hjelp av pagination. Slik kan man gå til en av de sidene av Modul og se antall sider for denne.

Her kan dere også se kommentarfeltet. Dette feltet er valgfri og kan tilsettes eller tas bort. Generelt er kommentarfeltet nødvendig dersom det er ønsket for diskusjon i en bestemt leksjon.

3.3.1.7 Modul lekse side med video



Figur 20 – Kurs videoformat

Common
• Breadcrumbs.jsx
• ButtonsPrevNext.jsx
• CommentBox.jsx
Header.css
• Header.jsx
Pagination.css
• Pagination.jsx

På video siden kan man se på video som er lastet opp av kurs forfatter eller video som kan spilles direkte fra Youtube. Det er enkel interface hvor man har et hoved video element og elementer som er felles for innhold sider. Disse felles elementene er Breadcrumbs, pagination, previous,

```
4 export class Header extends Component {  
5   render() {  
6     console.log(this.props);  
7     return [  
8       <div className="header">  
9         <h1>{this.props.title}</h1>  
10        </div>  
11    ];  
12  }  
13}  
14
```

next knapper, header og kommentarfeltet. De er lagret i en separat mappe som heter "Common".

Header endres for hver side, og det er avhengig av hvilken element ID som kalles. Det kalles i ModulDetailLesson ved hjelp av Lesson ID.

```
45 let { moduleId, lessonId } = useParams();
46 lessonId = lessonId * 1;
47 console.log(moduleId, lessonId);
48
49 if (lessonsData.length >= lessonId) {
50   let lesson = lessonsData[lessonId - 1];
51   console.log(lesson);
52   return (
53     <Container>
54       <Row>
55         <Breadcrumbs />
56       </Row>
57       <Row>
58         <Col></Col>
59         <Col xs={4} md={1}></Col>
60         <PaginationRow />
61         <Col></Col>
62       </Row>
63       <Row>
64         <Header title={lesson.title} />
```

3.3.1.8 Modul lekse side med quiz

The name of this lection

Here should be a question. Very interesting question about smith important

Choose one correct answer

first radio
 second radio
 third radio

Here should be a question. Even more interesting than the previous one

Choose one correct answer

first radio
 second radio
 third radio

Here should be a question. As you understand, this one is the most exciting questions that you can imagine

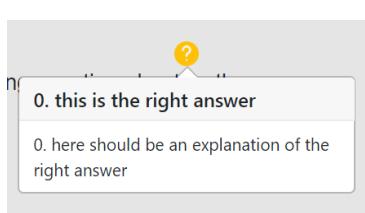
Choose one correct answer

first radio
 second radio
 third radio

Previous Next

Figur 21 – Kurs quizformat

Quiz siden fremstiller en rekke spørsmål med svar varianter. Det kan bli bare en eller flere varianter av riktig svar for konkrete spørsmål. Ved å trykke på spørsmåltegn, får man riktig svar og begrunnelse.



Den siden består av komponenter som lagres i tre filer. ModulDetailPageQuiz samler nødvendige komponenter ved bruk av grid systemet.

```

    ModulDetailPageQuiz
      ModulDetailPageQuiz.jsx
      PopUpCorrectAnswer.jsx
      # Question.css
      Question.jsx

```

Dataene fra spørsmål og svar kommer gjennom sortering ved å bruke mapping som parameter til avklart komponent.

PopUpCorrectAnswer tar ansvar for <Popover> ved å ta inn dataene fra databasen og viser dem på onClick hendelse.

Question fil bruker også mapping til å fremvise dataene. Den jobber også med checkbox til å ta inn input fra brukerne.

```

{quizData.map((item) => {
  return (
    <Container>
      <Row>
        <Col></Col>
        <Col></Col>
        <Col xs={12} md={4}>
          <PopUpCorrectAnswer dataAnswers={item.tips} />
        </Col>
      </Row>
      <Row>
        <Col></Col>
        <Col xs={12} md={6}>
          <Question dataQuestions={item} />
        </Col>
      </Row>
    </Container>
  );
});

```

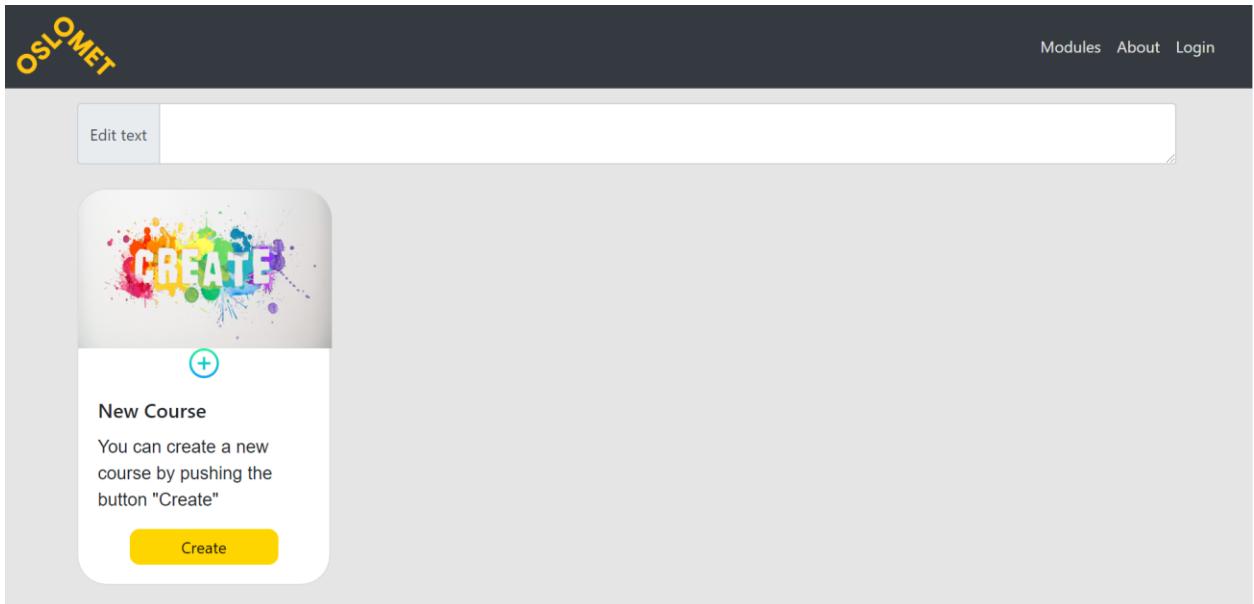
```

const handleClick = (event) => {
  setShow(!show);
  setTarget(event.target);
};

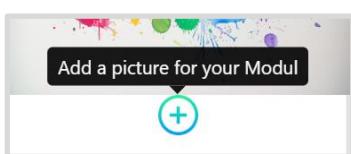
return (
  <div ref={ref}>
    <img onClick={handleClick} style={{ width: "20px" }} src={Question} />
    <Overlay
      show={show}
      target={target}
      placement="bottom"
      container={ref.current}
      containerPadding={20}
    >
      <Popover id="popover-contained">
        <Popover.Title as="h3">{text.rightAnswer}</Popover.Title>
        <Popover.Content>{text.explanation}</Popover.Content>
      </Popover>
    </Overlay>
  </div>
);
}

```

3.3.1.9 Admin hjemmesiden



Figur 22 – Administrator hovedside



På admin hjemmesiden kan man lage en ny modul. For å ha tilgang til denne siden trenger man de spesielle rettigheter (autorisasjon). Dem kan du lese mer om på login delen og database delen.

Her bruker vi <Card> fra bootstrap som hovedelement. I tillegg brukte vi også <OverlayTrigger> til å lage pop-up med instruksjoner for Admin. De brukes til å laste ned bildet for en modul.

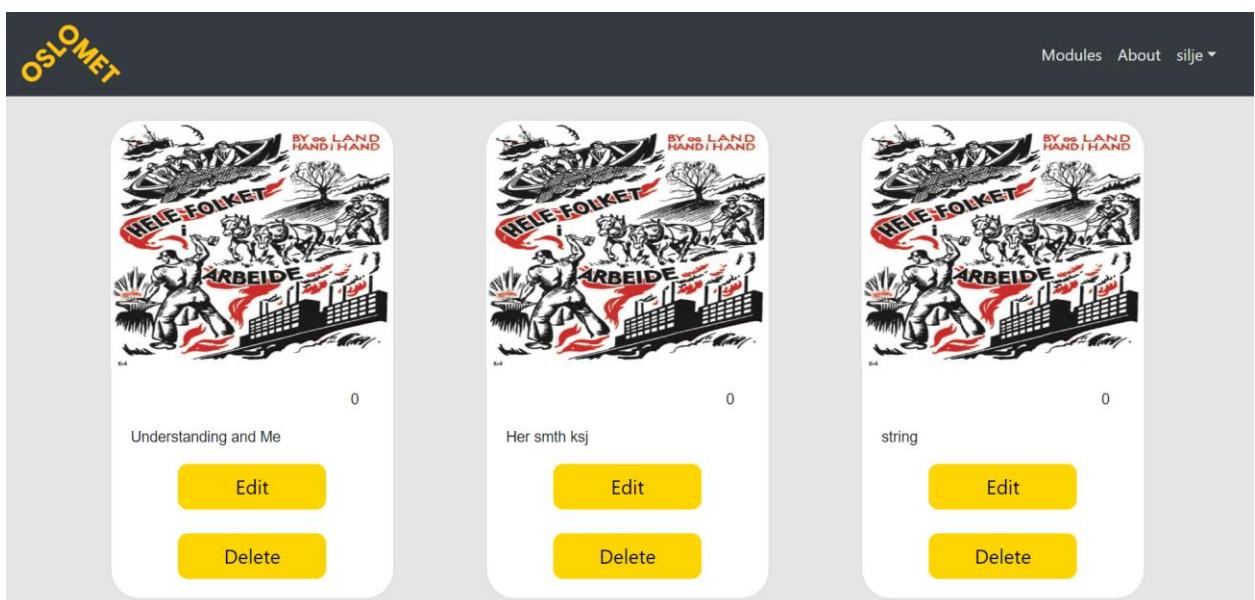


Etterhvert bestemte vi oss for å overføre denne funksjonen til "hovedModul" admin side. Slik blir det lettere å samle all info for en modul. Derfor endret vi pop-up meldingen og lagde ny input felt for bildet på Modul siden.

Ved å trykke på "Create" omdirigerte admin til hovedsiden for Modul. Her bruker vi routing. <Route path="/create" component={AdminModulMainPage} /> i App.jsx.

I tillegg til å lage Modul, kan man redigere text til hovedsiden der brukerne velger Modul.

I neste fase var det planlagt å jobbe blant annet med admin Profil. Derfor ble det bestemt i den fasen å tilsette moduler som er ferdiglagte på Admin hjemmesiden. De er ikke fordelt ifølge createBy, men det er mulig å redigere og slette dem på den siden. Interface for Admin hjemmesiden ser lik ut som User hjemmesiden. Beskrivelse av samspill mellom API og delete funksjonen kan man se [her](#).



Figur 23 – Administrator Hovedside redigering

State er et verktøy som lar oss oppdatere brukergrensesnittet basert på hendelser. Her bruker vi state til å endre informasjon på kortene, når man klikker på save knappen. Når brukeren klikker på knappen oppdateres brukergrensesnittet. Svaret på disse hendelsene gjøres gjennom en funksjon som håndterer hendelser onChange={this.changeState}.

I tillegg tar vi være på error som kan oppstå ved henting data fra database. Slik kan vi laste ned bildet på default hvis bildet fra backend kommer ikke.

```

<Card.Img
  variant="top"
  src={item.picture}
  onError={(e) => {
    e.target.onerror = null;
    e.target.src =
      "https://storage.googleapis.com/sn1-no-media/media/144223/standard_sosialdemokrati.jpg";
  }}
/>

```

3.3.1.10 Admin create modul side

The screenshot shows the 'Admin Modul redigeringsside' (Admin Module editing page). The top navigation bar includes 'OSLOMET', 'Modules', 'About', and 'Login'. The main form area starts with a text input field for 'Name of your course'. Below it is a larger text area for 'Describe briefly your course' with a placeholder 'Edit text'. To the right of this area are five input fields: 'Duration' (Whole course duration), 'Institution' (Presented institution), 'Subject' (Define subject), 'Price' (Price for the course), and 'Language' (Course's main language). Further down is a section for 'Describe lecturer of the course' with a small icon of a person. At the bottom left is a table for 'Schedule' with columns for 'Number', 'Name', and 'Duration'. At the very bottom are three buttons: 'Publish', 'Delete', and 'Save'.

Figur 24 – Administrator modul redigeringsside

Hovedmodul side for Admin består av tre hovedelementer som tilhører bare den siden og felles elementer som vi bruker på de andre sidene. Disse felles elementene gruppert i CommonAdmin mappe. Det er Save/ Delete knapper, text input. Hvor man kan skrive navn på Modul og beskrive innhold eller hoved ideen. Tekst input bruker vi videre på de andre sidene.

Under knapp "Add a content" finner man Modul sammendraget som er en del av hovedelementene på denne siden. Her kan man gi kort informasjon om kurset ved bruk av input feltene.

Neste element tar inn info om instruktør/foreleser. Ved å trykke på ikonet av bilde, kan man tilsette bilde av foreleseren. Ved siden av bilde, kan man redigere informasjon om foreleser (navn, institusjon, tittelen).

```

    < AdminModulMainPage
    < AdminModulMainPage.jsx
    # InstructorAdmin.css
    # InstructorAdmin.jsx
    # ScheduleAdmin.jsx
    # SummaryAdmin.css
    # SummaryAdmin.jsx
    > AdminModulQuizPage
    < CommonAdmin
    # ButtonGroupAdmin.css
    # ButtonGroupAdmin.jsx
    # SaveDelete.jsx
    # TextInput.jsx

```

3.3.1.11 Admin leksjonsside med tekst

The screenshot shows a web application interface for managing lesson content. At the top, there's a dark header with the 'OSLOMET' logo on the left and navigation links 'Modules', 'About', and a dropdown menu 'silje ▾' on the right. Below the header, the URL 'Home / Module / Lesson' is visible. A sidebar on the left contains three buttons: 'Text' (selected), 'Video', and 'Quiz'. To the right of the sidebar is a large text editor area with a vertical toolbar on the left labeled 'Edit text'. At the bottom of the text area are four yellow buttons: 'Delete', 'Previous', 'Next', and 'Save'. Above the text area, there are four small navigation icons: '<', '<<', '>', and '>>'. The main content area is currently empty.

Figur 25 – Administrator leksjonsside med tekst

Denne siden bygde vi for at Admin skal lage side for Modul, som inneholder tekst og/ eller bilder. Det er en side med enkelt interface som inneholder felles komponenter for lekser og felter hvor man kan skrive inn tekst.

Først lagde vi input feltet ved hjelp av bootstrap komponent `InputGroup.Text`. Med denne komponenten er det vanskelig å ta vare på text formatering. Derfor ble det bestemt å jobbe med `Draft.js`.

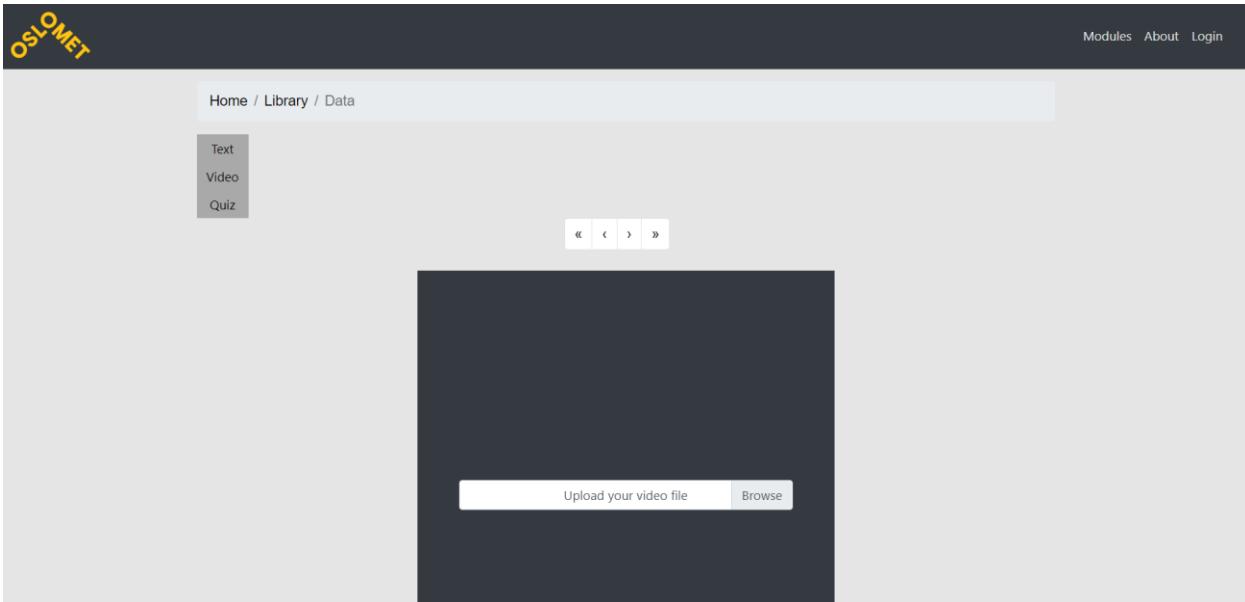
The screenshot shows a rich text editor interface. At the top, there's a toolbar with various text styling options: H1, H2, H3, H4, H5, H6, Blockquote, UL, OL, Code Block, Bold, Italic, Underline, and Monospace. Below the toolbar is a text input field containing the placeholder 'Tell a story...'. The entire interface is contained within a light gray box.

`Draft.js`-modellen⁸ er bygd med `immutable-js`, og tilbyr et API med funksjonelle tilstandoppdateringer og aggressivt utnytter data persistens for skalerbar minnebruk.

3.3.1.12 Admin leksjonsside med video

Neste type av lekseinnhold som en eier kan lage er en video. Det er en relativt enkelt design av siden, hvor man kan laste ned video fra sin PC-en eller sette inn lenke til video fra YouTube.

⁸ `Draft.js`. Rich Text Editor Framework for React.



Figur 26 – Administrator leksjonsside med video

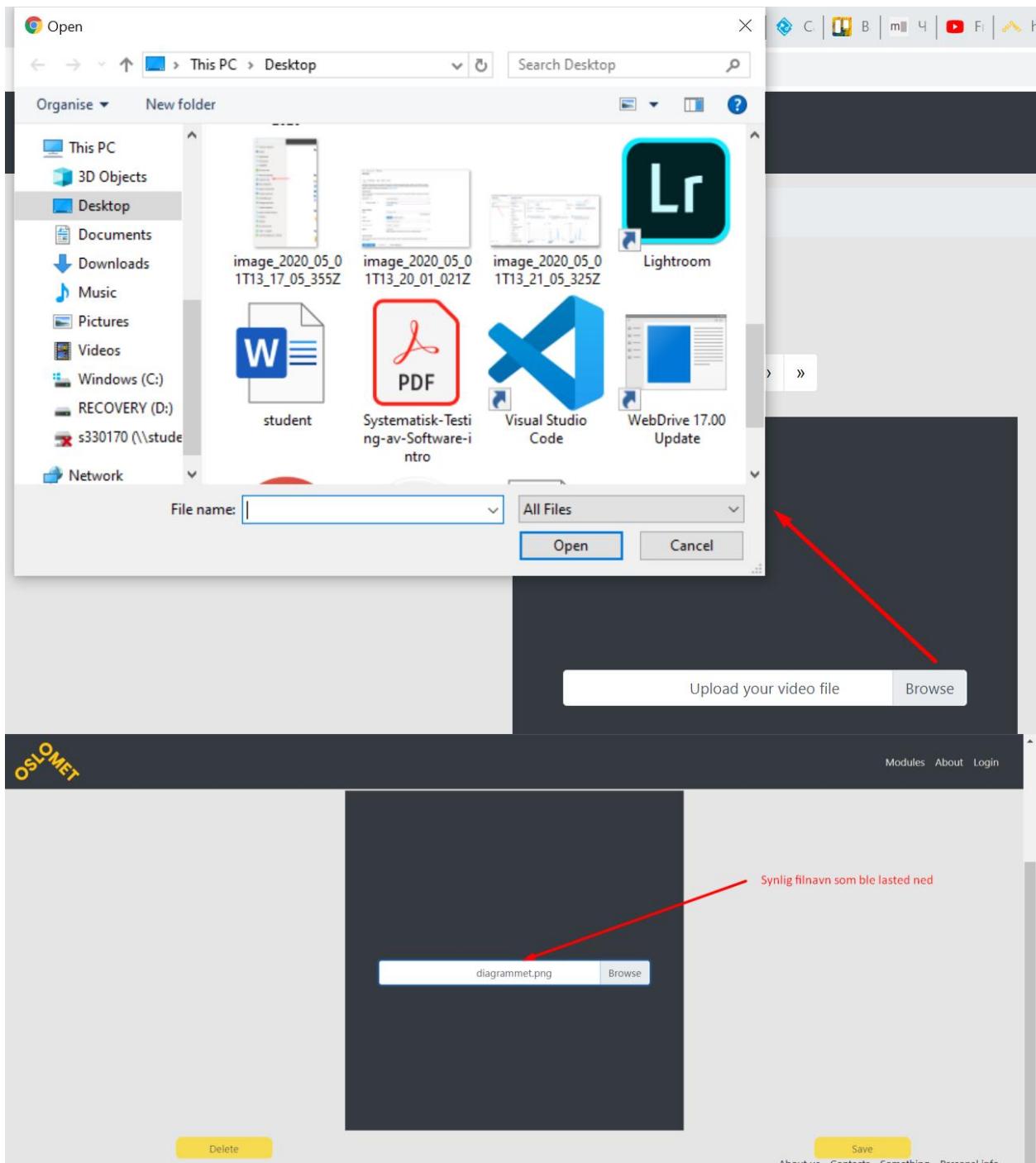
For å ha mulighet til å laste opp video fra PC-en, brukte vi Bootstrap komponent **FormFile** og plugin for å animere tilpasset filinput. Denne plug-in måtte vi installere på Visual Studio Code:

```
PS C:\Users\alina\Documents\OsloMet\webprosjekt\bachelor> npm install bs-custom-file-input --save
npm WARN react-scripts@3.4.1 requires a peer of typescript@>3.2.1 but none is installed. You must install peer dependencies yourself.
npm WARN sass-loader@8.0.2 requires a peer of node-sass@>4.0.0 but none is installed. You must install peer dependencies yourself.
npm WARN sass-loader@8.0.2 requires a peer of sass@>1.3.0 but none is installed. You must install peer dependencies yourself.
npm WARN sass-loader@8.0.2 requires a peer of fibers@>= 3.1.0 but none is installed. You must install peer dependencies yourself.
npm WARN tsutils@3.17.1 requires a peer of typescript@>2.8.0 || >= 3.2.0-dev || >= 3.3.0-dev || >= 3.4.0-dev || >= 3.5.0-dev || >= 3.6.0-dev || >= 3.6.0-beta || >= 3.7.0-dev || >= 3.7.0-beta but none is installed. You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.1.2 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.12 (node_modules\webpack-dev-server\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.12: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.12 (node_modules\watchpack\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.12: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.12 (node_modules\jest-haste-map\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.12: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ bs-custom-file-input@1.3.4
added 1 package from 2 contributors and audited 931949 packages in 42.761s
found 1 low severity vulnerability
  run `npm audit fix` to fix them, or `npm audit` for details
```

Vi bør vente på dokumentet som er en klar event, og deretter kalle på init-metoden for å gjøre den tilpassede filinndata dynamisk. Init metoden kaller vi i componentDidMount metoden. Denne metoden er tilgjengelig etter at komponenten har montert (mount). Det er etter at HTML fra render er ferdig lastet. Det kalles i komponentens livssyklus, og det signaliserer at komponenten og alle dens underkomponenter har rendered ordentlig.

```
componentDidMount() {
  bsCustomFileInput.init()
}
```



3.3.1.13 Admin leksjonside side med quiz

Det vanskeligste med denne siden var å kalle på et skjema for nye spørsmål og et skjema til variant av mulige svar. Hele problemet oppstartet ved å tilsette disse elementene slik at input lagres i et bestemt array med mulighet til å bli endret. I tillegg skal forrige spørsmålene bli vist på siden sammen med de nye oppdaterte "formene". Den skal altså dukke opp på siden for nye spørsmål.

Selve AdminModulQuizPage siden inneholder Question komponent og felleskomponenter for alle sider for admin (Breadcrumbs, ButtonGroupAdmin, PaginationRow, SaveDelete).

Question komponentet er ansvarlig for å lage selve quizen. Med andre ord, for å bygge opp spørsmål og svar, lagre disse dataene og vise dem på Admin siden. Vi prøvde å lage en veldig intuitivt og enkelt interface for quiz.

The screenshot shows a web-based quiz creation interface. At the top left is the OSLOMET logo. Top right buttons include 'Modules', 'About', and 'Login'. Below the header, a breadcrumb navigation shows 'Home / Library / Data'. A sidebar on the left lists 'Text', 'Video', and 'Quiz' (which is selected). In the main area, there's a text input field 'Type in your question' with a checkbox below it. To the right of this is a button labeled 'Add new option for the answer' with a red arrow pointing to it. Below the question input is a 'Type in answer for the question and explanation' input field with a blue plus icon in the center. Another red arrow points to this icon. At the bottom of the main area are 'Delete' and 'Save' buttons, along with links for 'About us', 'Contacts', 'Something', and 'Personal info'. The footer contains copyright information: 'Website made with ♡ Copyright © 2020 All Rights Reserved'.

For hvert ikon eller “form” uten synlig beskrivelse, lagde vi “Tooltip”. Det dukker opp hvis man peker på et element.

This screenshot shows the same quiz creation interface as above, but with additional tooltip descriptions for various elements. A tooltip 'Tick the correct answer' appears over the blue plus icon in the 'Type in answer for the question and explanation' field. Another tooltip 'Delete' appears over the 'Delete' button. At the bottom of the interface, two more tooltaps are shown: 'Add one more answer option' over the first blue plus icon and 'Add one more question' over the second blue plus icon.

Ved å trykke på “add one more answer option” ikonet får man et nytt skjema som skal fylles inn.

The screenshot shows a user interface for a question-and-answer application. On the left, there's a text input field labeled "Type in your question" with the placeholder "Our first question is.....". To the right of the input field is a green "G" icon with a gear symbol. Below the input field is a list of four answer options, each with a small checkbox and a name: "Ukraine", "Brasil", "Norway", and "Greece". The fourth option, "Pakistan", is highlighted with a blue selection bar and has a small blue "+" icon to its right.

Både spørsmål og svar kan korrigeres før skjemaet blir lagret. Den siste versjon skal lagres i et Array. Muligheten for å korrigere spørsmål etter lagring skal jobbes med i fase to.

I Consol av DevTools kan vi se hvordan data lagres og behandles:

The screenshot shows the React DevTools console with the state of the "Question" component expanded. The state object contains several properties: "numChildren" (0), "name" ("Our first question is....."), "explanation" (""), and an "answers" array of four objects. Each object has a "name" property (e.g., "Ukraine") and an "isRight" property set to false. A red arrow points from the "answers" array to the "Lagret mulige svar" annotation. Another red arrow points from the "name" property to the "Lagret mulig spørsmålet" annotation.

```
▼ Question ⓘ
  isMounted: (...)

  replaceState: (...)

  ▶ props: {}

  ▶ context: {}

  ▶ refs: {}

  ▶ updater: {isMounted: f, enqueueSetState: f, enqueueReplaceState: f}

  ▶ state:
    numChildren: 0
    name: "Our first question is....."
    explanation: ""
    ▶ answers: Array(4)
      ▶ 0: {name: "Ukraine", isRight: false}
      ▶ 1: {name: "Brasil", isRight: false}
      ▶ 2: {name: "Norway", isRight: false}
      ▶ 3: {name: "Greece", isRight: false}
      length: 4
```

Først initialiserte vi state i konstruktøren til komponenten.	<pre>export class Question extends Component { constructor(props) { super(props); this.state = [numChildren: 0, name: "", explanation: "", answers: [], newAnswer: { name: "", isRight: false },]; } }</pre>
Konstruktør () - metoden kalles før noe annet og her settes opp start tilstanden og andre startverdier.	"super (props)" vil starte opp foreldrenes konstruktur metode. Den lar komponentene arve metoder fra sine overordnede (React.Component). En betydelig del med å jobbe med React er gjentatt bruk, noe som betyr muligheten til å bruke en komponent i forskjellige tilfeller. Det er

	<p>ene grunnen til at vi bruker props for databehandlingen. De er eksterne og styres ikke av selve komponenten. Props sendes fra komponentene opp i hierarkiet, som også kontrollerer dataene.</p> <p>Klassebaserte komponenter kan lagre informasjon om nåværende situasjon. Denne informasjonen kalles state; den er lagret i et JSX-objekt. Koden ovenfor viser et objekt som representerer tilstanden til komponenten vår.</p>
Vi lagde en funksjon som kalles "onClick". Den tar inn event som parametere.	<pre> </pre> <pre> addAnswer(event) { this.setState({ answers: [...this.state.answers, this.state.newAnswer], newAnswer: { name: "", isRight: false }, }); } </pre>
"FormControl" kaller på funksjonen "onChangenewAnswer onChange" og passer på indeksen av alle nye mulige svar. Slik tar vi være på former, tilstand og overfører verdiene til en funksjon.	<pre> <FormControl aria-label="Text input with checkbox" value={this.state.answers[index].name} onChange={this.onChangenewAnswer} data-index={index} /> </pre>
En onChange-hendelsesbehandling returnerer et objekt som inneholder nyttige metadata som ID, navn og nåværende verdi. Vi får tilgang til dataens verdi inne i onChangenewAnswer ved å gå inn på event.target.value.	<pre> onChangenewAnswer(event) { if (event.target.dataset.index == undefined) { this.setState({ newAnswer: { name: event.target.value, isRight: this.state.newAnswer.isRight, }, }); } else { let data = this.state.answers; data[event.target.dataset.index * 1].name = event.target.value; this.setState({ answers: data }); } } </pre> <p>"event" er en syntetisk hendelse. React definerer disse syntetiske hendelsene i henhold til W3C-spesifikasjonen. Det gjør at vi ikke trenger å bekymre oss for kompatibilitet mellom nettlesere.</p>

For å bruke en klasse-komponent, må vi binde onChange-hendelseshåndterer en til konteksten av *this*.

```
this.addQuestion = this.addQuestion.bind(this);
this.addAnswer = this.addAnswer.bind(this);
this.changeQuestionTitle = this.changeQuestionTitle.bind(this);
this.changeQuestionExplanation = this.changeQuestionExplanation.bind(this);
this.onChangeNewAnswer = this.onChangeNewAnswer.bind(this);
```

For å velge riktig svar så må Admin sett kryss i riktig CheckBox.

Type in your question here the question

optional answer

optional second answer

right answer

optional third answer

For å lagre data fra CheckBox så kodet vi i funksjonen "handleCheckboxChange". Denne funksjonen behandler checked props for å sette dem til enten true eller false.

```
handleCheckboxChange(event) {
  console.log(event.target.dataset.index, event.target.checked);
  if (event.target.dataset.index == undefined) {
    let answer = this.state.newAnswer;
    answer.isTrue = event.target.checked;
    this.setState({
      newAnswer: answer,
    });
  } else {
    let data = this.state.answers;
    data[event.target.dataset.index * 1].isRight = Boolean(
      event.target.checked
    );
    this.setState({ answers: data });
  }
  console.log(this.state);
}
```

Spørsmål har to riktig svar

```
numChildren: 0
name: "fb"
explanation: ""
answers: Array(2)
  ▶ 0: {name: "fv", isRight: true}
  ▶ 1: {name: "fdv", isRight: true}
  length: 2
  ▶ __proto__: Array(0)
newAnswer: {name: "", isRight: false}
```

For å lage funksjonalitet som gir mulighet til å sette inn nye spørsmål, så brukte vi det samme prinsippet som ble beskrevet for mulige svar.

Vi trenger å ha "keys" for våre elementer i massiv data. Slik kan vi ta vare på elementer i array. Da kan vi slette, tilsette eller endre dem.

Vi hadde et problem med designet av blokk med spørsmål. Problemet var at det var usynlige grenser som kunne bli veldig frustrerende for brukerne.

OSLOMET

Modules About Login

En blokk for å lage spørsmål

Type in answer for the question and explanation

Type in your question

Type in answer for the question and explanation

Type in your question

Type in answer for the question and explanation

Type in your question

Derfor endret vi designet på denne siden, for å øke brukervennligheten.

OSLOMET

Modules About Login

Type in your question

Type in answer for the question and explanation

Type in your question

Type in answer for the question and explanation

Figur 27 - Administrator leksjonside side med quiz endelig design

3.3.1.14 Login side

Figur 28 – Login side

På login siden har vi to felter. Vi har felter som inkluderer e-post og passord. Knappen "Submit" fører til admin siden. Dette kan videreutvikles med at vanlig bruker også kan registrere seg og logge seg inn. Da vil brukeren komme inn på modul hovedsiden. For dette login skjema brukte vi bootstrap elementer: form, button og checkBox.

For å sikre at et element i webapplikasjonen returnerer gyldige data så bygger vi automatisk validering i koden. Får å oppnå dette målet bruker vi en "Formik". Dette er en pakke som brukes for å bygge skjemaer. Vi tilpasser Formik-komponentene med en validering funksjon. Det gjøres for "onSubmit" for feilmeldinger, og viser dem til brukeren dersom nødvendig.

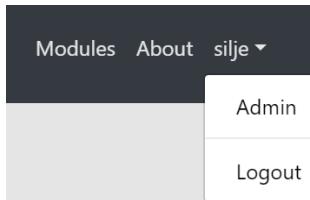
<pre>this.state = { email: '', password: '', formErrors: { email: '', password: '' }, emailValid: false, passwordValid: false, formValid: false, modalShow: false, };</pre>	<p>Her initialiserer vi state for Login komponentet. this.props er satt opp av React selv og this.state har en spesiell betydning. Her står vi fritt til å legge til flere felt i klassen manuelt. Dette gjøres dersom vi trenger å lagre noe som ikke er med i dataflyten.</p>
<pre><label htmlFor="password">Password</label></pre> <p>Password</p> <input type="password" value="....."/>	<p>For input feltene bruker vi label. Det gjør at passord kan skrives inn uten å se passordet. Med andre ord så blir passordet skjult.</p>
<pre>validationSchema={Yup.object().shape({ email: Yup.string().required("Email is required"), password: Yup.string().required("Password is required"), })}</pre> <p>Email</p> <input type="text"/> <p>Email is required</p>	<p>Siden E-post og passord er påkrevd for admin, så bruker vi "Yup" i tillegg til "Formik". Yup er en skjemavalidering som ofte brukes i forbindelse med Formik.</p>

```

onSubmit={({ email, password }, { setStatus, setSubmitting }) => {
  console.log("email", email, "password", password);
  setStatus();
  authenticationService.login(email, password).then(
    (user) => {
      const { from } = this.props.location.state || {
        from: { pathname: "/" },
      };
      this.props.history.push(from);
    },
    (error) => {
      setSubmitting(false);
      setStatus(error);
    }
  );
}

```

Vi trenger også en callback av “onSubmit”. Denne callback vil ta to parametere, verdier av et objekt (email og password) og verdiene som representerer inngangsverdiene fra skjemaet. (setStatus, setSubmitting) onSubmit kaller på setSubmitting fra props.



I tillegg tar vi vare på brukerens state. Det er nødvendig slik at man ikke må logge seg inn hver gang siden oppdateres. For å gå til admin siden og logge ut så lagde vi dropdown i nav. Denne logikken ble bygget basert på authenticationService og currentUserValue.

```

const controlPage = () => {
  console.log("uno", localStorage.getItem("sessionToken"));
  if (authenticationService.currentUserValue.token) {
    return (
      <>
        <NavDropdown
          title={authenticationService.currentUserValue.username}
          id="nav-dropdown"
          eventKey="3"
        >
          <NavDropdown.Item eventKey="3.1" href="/admin">
            Admin
          </NavDropdown.Item>
          <NavDropdown.Divider />
          <NavDropdown.Item onClick={authenticationService.logout} href="/">
            Logout
          </NavDropdown.Item>
        </NavDropdown>
      </>
    );
  } else {
    return (
      <>
        <Nav.Link href="/login">Login</Nav.Link>
      </>
    );
  }
}

```

For å lage logg inn funksjonaliteten med Facebook og Google, så jobbet Silje først med Frontend delen. Det virket for kompleks for å jobbe med den i starten av prosjektet. Det førte til at denne delen ble reservert for senere. Det som ble laget ligger i vedlegg.

Deretter var Login delen korrigert til UI som ble definert tidligere, og det ble lagt til registreringsside.

3.3.1.14 Registrering Side

The screenshot shows a "Log in" form with fields for Email and Password, a "Check me out" checkbox, and two buttons: "Submit" and "Register". A red arrow points from the text below to the "Register" button.

Ved å trykke på "Register" knappen så blir brukeren sendt til registreringsskjema.

Registreringsskjemaet fremstilles med fem skjemaer for input og to knapper (Save, Close). "Name", "email" og "Password"

```
validationSchema={Yup.object().shape({
  name: Yup.string().required('Name is required'),
  email: Yup.string().required('Email is required'),
  password: Yup.string().required('Password is required'),
})}
```

må fylles ut. Det kontrollerer vi ved hjelp av Yup.

Vi håndterer skjema validering via et bibliotek som heter Formik. Slik kan isValid og isInvalid-rekvisitter legges til for å danne controllere manuelt. Den kan også bruke valideringsstiler. Skjemavalidering skjer ved hjelp av Yup.

Her bruker vi <Modal> fra bootstrap til å vise registrerings form som en pop-up vindu.

The screenshot shows a "Register page" modal with fields for Name, Surname, Phone, Email, and Password. It includes "Close" and "Save" buttons at the bottom.

3.3.1.15 Ruter

React er et bibliotek som fungerer bra for å lage komponenter og gir et system for å administrere tilstand, men for å lage en kompleks applikasjon vil den kreve ytterligere moduler. Derfor bruker vi React Router for å lage deep linking. Her er rendering av forskjellige komponenter med en React-ruter.

```

export default class App extends Component {
  render() {
    const sessionToken = localStorage.getItem("sessionToken");
    return (
      <Layout>
        <Switch>
          <Route exact path="/" component={CardModules} />
          <Route path="/login" component={LoginPage} />
          <Route path="/modules" exact component={CardModules} />
          <Route path="/about" component={AboutPage} />
          <Route
            exact
            path={"/modules/:moduleId"}
            render={(props) => <ModulForm {...props} />}
          ></Route>
          <Route exact path="/modules/:moduleId/lessons/:lessonId">
            <ModulDetailLesson />
          </Route>
          <Route exact path={"/modules/:moduleId/finish"}>
            <Finish />
          </Route>
          <Route path="/admin" component={CardModulesAdmin} />
          <Route path="/create" component={AdminModulMainPage} />
          <Route
            exact
            path="/modules/:moduleId/edit"
            component={AdminModulMainPage}
          />
          <Route
            exact
            path="/modules/:moduleId/edit/lessons/create"
            component={AdminModulDetailLesson}
          />

          <Route
            exact
            path="/modules/:moduleId/edit/lessons/:lessonId/edit"
            component={AdminModulDetailLesson}
          />
        </Switch>
      </Layout>
    );
  }
}

```

<Route path="/login" component={LoginPage} /> er bare en rute, og det er veldig enkelt å lage direkte linken til for eksempel Login Side.

Men her bruker vi også mange ruter som baseres på gjeldende path og bestemmer hvilken komponent som skal vises. Hver <Route> med en samsvarende URL vil gi ut en tilsvarende komponent. Ruteren hjelper oss med å lage de nestede rutene.

Komponentene blir nestet sammen i henhold til hvilken retning rutene følger. Når brukeren navigerer til et bestemt LessonId, vil ruteren plassere LessonId-komponenten i Lessons og Lessons i ModulId og ModulId i Modules. Som et resultat, når man bytter LessonId, vil fire nestede komponenter i ruten vises.

```
/modules/:moduleId/lessons/:lessonId
```

React-ruteren vil overføre verdien moduleId som en Modul-egenskap. I modul vil den være tilgjengelig som let {moduleId} = this.props.match.params.

```
componentDidMount() {
  let { moduleId } = this.props.match.params;
  apiService
    .getModuleByID(moduleId)
    .then((data) => this.setState({ ModulData: data }));
  console.log(this.props);
}
```

Et "match" objekt inneholder informasjon om hvordan en rute (<Route path>) matcher URL-en. Matchings objekter inneholder følgende egenskaper:

- params - (objekt) Nøkkel- / verdipar som er analysert av URL-adressen som tilsvarer de dynamiske segmentene til path,
- isExact - (boolsk) er sant hvis hele URL-en ble matchet (ingen etterfølgende tegn)
- path - (string) Path mønsteret som brukes til å matche. Nyttig for å bygge nestede ruter
- url - (string) Den matchede delen av nettadressen. Nyttig for å bygge nestede <Link>

3.3.1.16 Samspil med BackEnd delen

Før API fra backEnd var klar til å brukes fra Frontend, brukte vi JSONPlaceholder <https://jsonplaceholder.typicode.com/>. JSONPlaceholder er et gratis online REST API som vi kan bruke når vi trenger falske data. Også har vi brukt Json (JavaScript Object Notation) som en av måtte til å lagre data. Det var en midlertidig løsning før vi fikk APlet. Med Json ble objekter beskrevet.

I tillegg laget vi en egen falske database til å utføre samarbeid mellom klient-delen og backend.

Med APlet, fikk vi testet dem igjen ved hjelp av Swagger og Postman. Da brukte vi APlets fire forskjellige HTTP Metoder: GET, POST, PUT og DELETE.

```
[{"email": "admin@admin.com", "pass": "admin123", "role": "admin"}, {"email": "user1@user.com", "pass": "user123", "role": "user"}]
```

The screenshot shows the Swagger UI for a RESTful API named "My API". The top navigation bar includes the Swagger logo, a "Select a definition" dropdown set to "My API V1", and a "My API V1" button. Below the header, the title "My API" is displayed with a "v1" badge and a "GAS" badge. A link to "/swagger/v1/swagger.json" is shown.

The API is organized into several sections:

- Admin** (Blue background):
 - GET /api/Admin
 - POST /api/Admin
 - GET /api/Admin/{adminID}
 - PUT /api/Admin/{adminID}
 - DELETE /api/Admin/{adminID}
- Persons** (Light Blue background):
 - GET /api/Persons
 - POST /api/Persons
 - GET /api/Persons/{personID}
 - PUT /api/Persons/{personID}
 - DELETE /api/Persons/{personID}
- Lessons** (Light Green background):
 - GET /api/module/{moduleID}/Lessons
 - POST /api/module/{moduleID}/Lessons
 - GET /api/module/{moduleID}/Lessons/{lessonID}
 - PUT /api/module/{moduleID}/Lessons/{lessonID}
 - DELETE /api/module/{moduleID}/Lessons/{lessonID}
- Teacher** (Light Orange background):
 - GET /api/Teacher/{teacherID}
 - PUT /api/Teacher/{teacherID}
 - DELETE /api/Teacher/{teacherID}
- Modules** (Light Red background):
 - GET /api/Modules
 - POST /api/Modules
 - GET /api/Modules/{moduleID}
 - PUT /api/Modules/{moduleID}
 - DELETE /api/Modules/{moduleID}

Figur 29 – API fra Swagger

GET-metoden brukes til å gjenopprette data fra en server med den angitte ressursen. For eksempel har vi et API med endepunkt for Moduler. Å lage en GET-forespørsel til det endepunktet, bør returnere en liste over alle tilgjengelige Moduler. GET-forespørsel bare ber om data og ikke endrer ressурсе. Og det er viktig å sjekke hvert kjent endepunkt (end-point) med en GET-forespørsel.

Derfor bør vi sjekke at en gyldig GET-forespørsel returnerer en 200 statuskode. HTTP 200 OK det er suksess status svarkode indikerer at forespørselen har lyktes. Et svar på 200 er som

General

Request URL: <https://wixapi.azurewebsites.net/api/Persons/authenticate>

Request Method: POST

Status Code: 200 OK

Remote Address: 52.176.149.197:443

Referrer Policy: no-referrer-when-downgrade

Response Headers view source

```

Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: http://localhost:3000
Content-Encoding: gzip
Content-Type: application/json; charset=utf-8
Date: Thu, 14 May 2020 10:53:43 GMT
Server: Microsoft-IIS/10.0
Set-Cookie: ARRAffinity=6fb6b68a2c3be2ece683de467ba1afb8f706e7c645a72f80
Transfer-Encoding: chunked
Vary: Origin,Accept-Encoding
X-Powered-By: ASP.NET

```

API for autorisasjon

```

import { authenticationService } from './auth/authentication.service';

export function handleResponse(response) {
  return response.text().then(text => {
    const data = text && JSON.parse(text);
    if (!response.ok) {
      if ([401, 403].indexOf(response.status) !== -1) {
        // auto logout if 401 Unauthorized or 403 Forbidden response returned from api
        authenticationService.logout();
        //window.location.reload(true);
      }
    }
    const error = (data && data.message) || response.statusText;
    return Promise.reject(error);
  })
  return data;
}

```

cardData: {...} ↴

- ▶ cardData: {moduleId: 11, teacherID: 5, adminID: 5, institution: "string", description: "string", ...}
- ▶ __proto__: Object
- ▶ {moduleId: 11, teacherID: 5, adminID: 5, institution: "string", description: "string", ...}

standard cacheable. Betydningen av en suksess for GET metoden er ressursen er hentet og overføres i message bodyen.

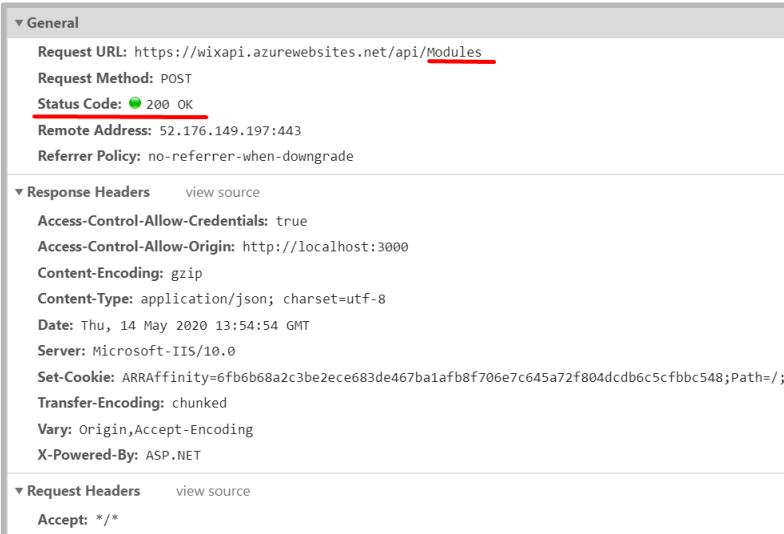
Også bør vi forsikre oss at en GET-forespørsel til en bestemt ressurs returnerer riktig data. For eksempel returnerer GET / Modules en liste over moduler.

POST-forespørsler brukes til å sende data til API-serveren for å opprette eller oppdatere en ressurs. For eksempel er et registreringsskjema. Når vi fyller ut feltene i et skjema og trykker på Save, blir dataene lagt inn i response bodyen på forespørselen og sendt til serveren. Det betyr

at POST muterer data på backend-serveren ved å opprette eller oppdatere en ressurs, i motsetning til en GET-forespørsel som ikke endrer noen data.

Siden POST-forespørsler endrer data, er det viktig å ha API-tester for alle POST-metodene. For å teste POST-forespørsler opprett vi en ressurs med en POST-forespørsel og sørge for at en 200 statuskode returneres. Deretter lager en GET-forespørsel for den ressursen, og kontroller at dataene ble lagret riktig. Også legg til tester som sikrer at POST-forespørsler mislykkes med feil eller dårlig formatert data.

PUT-forespørsler brukes til å sende data til API for å opprette eller oppdatere en ressurs.



The screenshot shows a network request details panel from a browser's developer tools. The 'General' section displays:

- Request URL: <https://wixapi.azurewebsites.net/api/Modules>
- Request Method: POST
- Status Code: 200 OK
- Remote Address: 52.176.149.197:443
- Referrer Policy: no-referrer-when-downgrade

The 'Response Headers' section shows:

- Access-Control-Allow-Credentials: true
- Access-Control-Allow-Origin: http://localhost:3000
- Content-Encoding: gzip
- Content-Type: application/json; charset=utf-8
- Date: Thu, 14 May 2020 13:54:54 GMT
- Server: Microsoft-IIS/10.0
- Set-Cookie: ARRAffinity=6fb6b68a2c3be2ece683de467ba1afb8f706e7c645a72f804dcdb6c5cfbbc548; Path=/; ;
- Transfer-Encoding: chunked
- Vary: Origin,Accept-Encoding
- X-Powered-By: ASP.NET

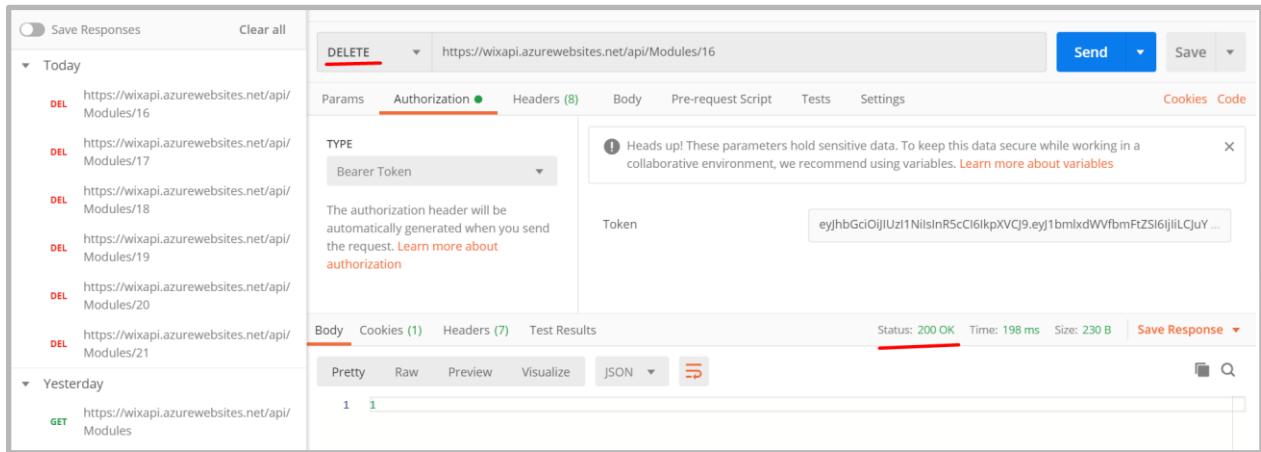
The 'Request Headers' section shows:

- Accept: */*

Forskjellen er at PUT-forespørsler er identiske. Det vil si at å kalle samme PUT-forespørsel flere ganger vil alltid gi samme resultat. I motsetning til det å kalle en POST-forespørsel flere ganger har bivirkninger av å opprette den samme ressursen flere ganger. Derfor når vi oppretter API-tester for PUT, gjør vi det til å bekrefte denne oppførselen. Det betyr at gjentatte ganger av PUT-forespørsel returnerer alltid det samme resultatet. Etter å ha oppdatert en ressurs med en

PUT-forespørsel, bør en GET-forespørsel for den ressursen returnere de nye dataene. PUT-forespørsler skal mislykkes hvis ugyldige data leveres i forespørselen - ingenting skal oppdateres.

DELETE skal slette ressursen på den angitte URL-en. For å teste DELETE-forespørsel oppretter vi en ny Module med en POST-forespørsel. Når module-ID er returnert fra POST, lager vi en DELETE-forespørsel til / modules / {{moduleID}}. En påfølgende GET-forespørsel til / modules / {{moduleID}} bør returnere en 404-statuskode som betyr at / modules / {{moduleID}} ikke ble funnet.



Figur 30 – Testing i Postman

På Frontend delen erklærte vi spesielle metoder på komponentklassen for å kjøre en kode når en komponent monteres. Når vi henter inn API-data, ønsker vi å bruke componentDidMount, fordi vi vil sørge for at komponenten har blitt gjengitt til DOM før vi tar inn dataene.

```
componentDidMount() {
  //fetch("https://wixapi.azurewebsites.net/api/Modules").then(x=>{return x.json()}).then (x=>{this.setState({cardData:x})})
  apiService.modules().then(
    (data) => {
      this.setState({ cardData: data });
    },
    (error) => [
      //setSubmitting(false);
      //setStatus(error);
    ]
  );
}
```

ComponetDidMount er “livssyklusmetoden”. Metoden componentDidMount () kjøres etter at komponentutgangen er blitt gjengitt til DOM.

```
export const apiService = {
  modules,
};

function modules() {
  const requestOptions = {
    method: "GET",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${authenticationService.currentUserValue.token}`,
    },
  };

  return fetch(`https://wixapi.azurewebsites.net/api/Modules`, requestOptions)
    .then(handleResponse)
    .then((data) => [
      // store user details and jwt token in local storage to keep user
      // logged in between page refreshes
      console.log(data),
      return data;
    ]);
}
```

Når denne metoden kjøres, ble komponenten allerede gjengitt en gang med render () -metoden, men den vil gjengis igjen når de dataene som ble hentet ble lagret i den lokale tilstanden til komponenten med setState (). Vi bruker this.setState () til å planlegge oppdateringer til komponentens lokale tilstand. Etterpå kan den lokale tilstanden brukes i render () -metoden for å vise den eller for å gi den ned som props.

Først prøvde vi å gjøre det med direkte Api lenken. Og det var

fungerte fint. Etterpå lagde vi et separat apiService-fil hvor vi deklarerer apiService const.

```

componentDidMount() {
  let { moduleId } = this.props.match.params;
  let promise = fetch(`https://wixapi.azurewebsites.net/api/Modules/${moduleId}`);

  promise
    .then((value) => {
      return value.json();
    })
    .then((value) => {
      console.log(value);
      this.setState({ modulData: value });
    });
  console.log(this.props);
}

```

Sammen med autentisering og autorisasjon, fikk vi bruke de andre delene av programmet uten problemer.

Når vi fetcher data for modular bruker vi JavaScript-promises for å løse den asynkronne responsen. Når dataene hentes veldig raskt, blir de lagret i lokal tilstand med Reacts this.setState () -metoden. Da vil render() -metoden utløses igjen, og vi kan vise data som har blitt hentet.

<pre> <FormControl as="textarea" aria-label="With textarea" value={this.state.title} onChange={this.changeState} id="title" rows="2" /> </pre>	<p>For hver av input feltene, definerer vi verdi med et bestemt ID og kaller funksjonen onChange.</p>
<pre> changeState = (event) => { let s = {}; let field = event.target.id; s[field] = event.target.value; this.setState(s); }; </pre>	<p>Den funksjonen endrer state til våre objekter.</p>
<pre> <Button type="button" onClick={this.saveModul}> Save </Button> </pre>	<p>For å ha bestemt tidspunkt/hendelse når data blir lagret, tilsetter vi Save knapp på interface. Ved å klikke på knappen, kaller vi på saveModul funksjon.</p>
<pre> saveModul = (event) => { apiservice.createModule(this.state).then((x) => { console.log(x); }); }; </pre>	<p>SaveModul er en funksjon som kaller på APlet fra apiService klassen og lagrer data. Disse dataene kan vi se i konsollen.</p>
<pre> <SummaryAdmin value={this.state} onChange={this.updateState} /> </pre>	<p>For å oppdatere data som sendes gjennom child komponenter, trenger vi å "løfte" opp state fra child komponent til parent komponent. Derfor laget vi updateState funksjon.</p>

<pre>updateState = (data) => { this.setState(data); console.log(data); };</pre>	updateState funksjon tar inn data som kommer fra child komponent som parameter og oppdaterer state.
<pre><Form.Control placeholder="Presented institution" value={this.state.institution} onChange={this.changeState} id="institution" /></pre>	I child element for hvert input felt definerer vi også value med et bestemt ID og kaller funksjonen changeState.
<pre>changeState = (event) => { let s = this.state; let field = event.target.id; if (typeof s[field] == "number") s[field] = event.target.value * 1; else s[field] = event.target.value; this.setState(s); this.props.onChange(s); };</pre>	Funksjonen changeState befinner seg i child komponent og endrer state av element. I tillegg bruker vi IF-statements for de feltene som inneholder numeriske verdier. På denne måten får vi ikke konflikter med databasen.

3.3.2 Backend

Backend inneholder alle API-er som brukes av applikasjonen til å utføre forskjellige forespørsler. Vår applikasjon er bygget opp ved bruk av ASP.NET core rammeverket. .Net er en utviklingsplattform som består av ulike verktøy, programmeringsspråk og biblioteker. De brukes for å bygge ulike typer applikasjoner. I vårt tilfelle brukte vi den for å bygge en webapplikasjon, og vi kodet med programmeringsspråket C#.

3.3.2.1 Database

Vi startet med å bruke Entity Framework⁹ ¹⁰, som gjør det mulig å jobbe mot database i .Net, ved bruk av .Net objekter uten å måtte tenke på SQL. Den har veldig mye funksjonalitet som vi ikke trenger og den kan skape mye problemer under migrering (oppdatering mot database). Derfor bestemte vi senere ut i prosjektet at det var bedre å benytte seg av Dapper.

Det er klassifisert som en “micro” ORM (Object-Relational Mapper), det vil si mapper som lager objekter basert på database forespørsler. Fordeler med det er at det ikke er så komplekst, som betyr at det er mindre som kan gå galt, og gir oss renere SQL funksjonalitet som “create, read, update og delete” uten å skjule hva som skjer i bakgrunnen, slik som Entity Framework.

I starten brukte vi Sqlite for databasen, men det er en lokal database i motsetning til Microsoft SQL Server som kan kjøres både lokal og fra en felles server. Lokal database er fint når man ønsker å jobbe alene på databasen. Siden vi skulle jobbe både lokalt og mot felles database, så ble SQL Server valgt. Den var gratis for utvikling i Azure og den blir lagt inn med Visual Studio. Vi har brukt enkel standard SQL, som gjør at det skal fungere godt mot andre relasjonsdatabaser som MySQL eller PostgreSQL.

SQL Server er en relasjonsdatabase som består av tabeller med relasjoner til hverandre. Vi bestemte oss for å bruke relasjonsdatabase fordi vi hadde mer kjennskap til det fra studiet. En

⁹ Shailendra, 2011.

¹⁰ Microsoft, 2016.

annen grunn var at vi ønsket å lage skjemaer for alle tabellene vi skulle ha. Strukturen for en relasjonsmodell kan bli sett i tabellen¹¹:

SQL	Relasjonsmodellen
Tabell	Relasjon
Kolonne	Attributt
Rad	Forekomst

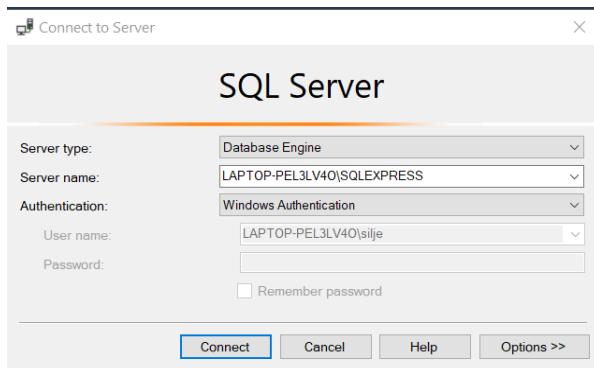
Med bruk av relasjonsmodellen, har vi: (i) en generell datastruktur, der alle data lagres som relasjoner, (ii) integritetskontroll, med bruk av primær- og fremmednøkler, (iii) spørrespråk mot relasjoner.¹²

SQL Server

Det er mange forskjellige versjoner av SQL Server: Express LocalDB, Express, Developer, Web, Standard og Enterprise.

Express LocalDB kjøres opp sammen med applikasjonen slik som for eksempel SQLite. Express er en enklere versjon som er gratis og kan brukes som server eller lokalt, slik som for eksempel MySQL og PostgreSQL. Developer er kun for utvikling, men har de aller fleste funksjonene som den som koster penger. Web, Standard og Enterprise koster penger og har mange flere funksjoner, som, for eksempel, Oracle.¹³

For lokal utvikling brukte vi Express LocalDB og Express, mens i Azure brukte vi Azure SQL. Den vil som i trial vil være ganske lik som developer, men forskjellen mellom SQL Server og Azure SQL er at den siste oppdaterer seg kontinuerlig, mens den første blir oppdatert med noen års mellomrom.

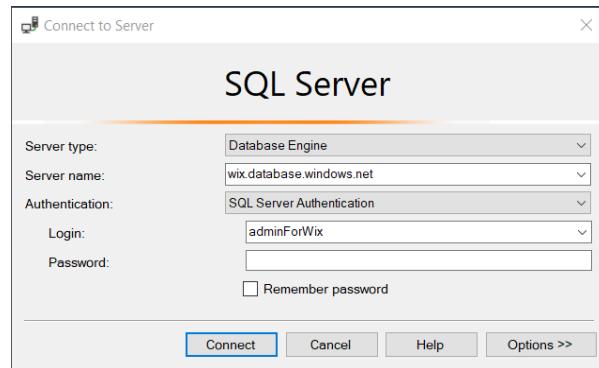


Bildet til venstre viser et eksempel på pålogging på SQL Server lokalt med Windows brukerkonto ved bruk av SQL Server Management Studio (SSMS).

¹¹ Basert på tabellen fra boken til Kristoffersen, Bjørn. Databasesystemer. Side 144. Universitetsforlaget. 4. Utgave.

¹² Kristoffersen 135.

¹³ Microsoft, 2019.



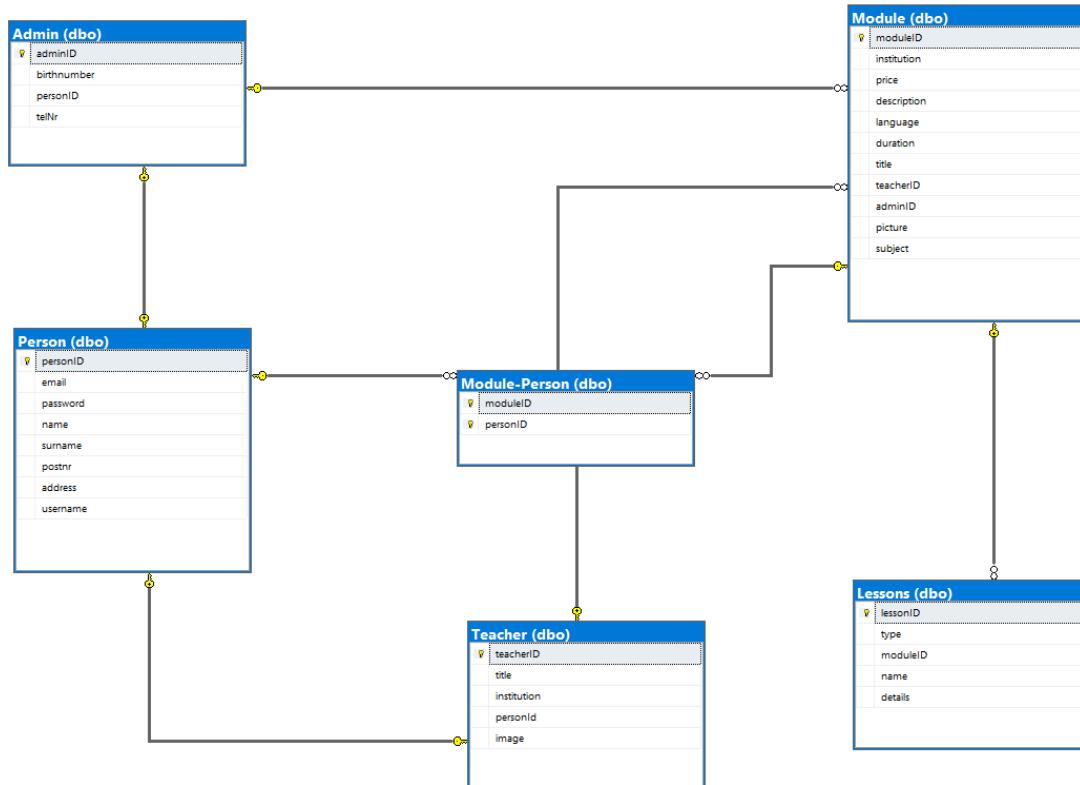
Bildet til høyre viser et eksempel på pålogging på Azure SQL med SQL bruker.

Database oppbygging

Databasen vår består av 6 tabeller og 25 stored procedures. Vi bruker en stored procedure for hvert kall fra Dapper.

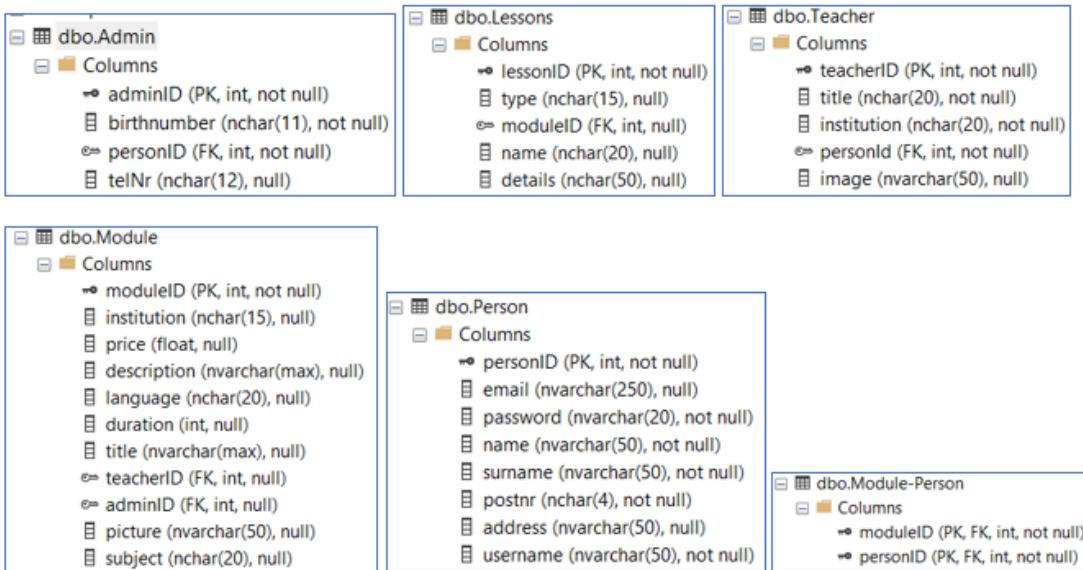
Tabeller

Her er et diagram med alle de tabellene som vi har i slutten av prosjektet.



Figur 31 – Tabelldiagram for database

PersonID i tabell Person, adminID i tabell Admin, lessonID i tabell Lesson og teacherID i tabellen Teacher er hovednøkler (primary keys). En oversikt over alle fremmednøklene kan bli sett på neste bilde:



Figur 32 – Oversikt over tabeller

I tabellen Module-Person er moduleId og personID fremmednøkler (foreign keys), men sammen representerer de hovednøkkelen. Noe som kalles sammensatt primærnøkkel¹⁴ for tabellen, som skal holde oversikt over de personene som tar modules. Hovednøkkel ser slik ut: .

Tabellene Admin og Teacher får attributtet personID fra tabellen Person som fremmednøkler, representert slik: . Når man registrerer seg så er det ikke bestemt om man er admin eller lærer enda. Det er noe som blir bestemt av de eksisterende administratorer, som tildeler en eksisterende person en teacherID eller adminID. Det gjøres basert på deres rolle i virksomheten. Med denne strukturen ser man at flere personer kan være lærere eller admin, imens en teacherID eller adminID er tildelt til kun en person.

Tabell Lessons tar moduleId fra Module tabell som fremmednøkkel, fordi en modul kan inneholde flere leksjoner, men en leksjon kan tilhøre til kun en modul.

En modul skal ha en lærer og en admin. Derfor er teacherID og adminID fremmednøkler i tabellen Module.

Stored Procedures

¹⁴ Kristoffersen 66.

En “stored procedure” i SQL Server er en samling av en eller flere “SQL statements”. Grunnen til at vi bestemte oss for å bruke akkurat dette er, blant annet, fordi:

- (i) Det er lettere å vedlikeholde databasen, fordi vi får større kontroll over hva som endres da vi kaller databasen fra koden. Vi kan også teste prosedyrene uten å måtte gjøre det fra .NET applikasjonen.
- (ii) Man får større kontroll på hva spørringen gjør og kan gi høyere rettigheter inne i prosedyren mot databasen enn det brukeren har ellers lov til.¹⁵
- (iii) Stored procedures er parametriserte, noe som betyr at de er mer robuste mot SQL-injeksjon.¹⁶
- (iv) Er raskere enn håndskrevet SQL spørringer og mindre krevende for databasen.
- (v) Kan legge med regler (schemabound) slik at du ikke kan gjøre endringer i tabellene som gjør at prosedyren feiler.¹⁷

Vi har laget prosedyrer for å opprette, oppdatere, slette og hente fra de forskjellige tabellene i databasen.

Til update prosedyrene, prøver vi å oppdatere tabellen med nye opplysninger som brukeren eller admin gir. Create prosedyrene vil lage nye rader fra de tabellene. Delete vil da slette de ønskede radene. Til slutt, fetch skal hente ut data fra de tabellene som vil bli brukt til å beregne eller vise på nettsiden.

Opprettning av en procedure

For å opprette procedure må man gå inn på Sql Server til webapplikasjonen vår. Vi må skrive en script for å få dette til. Vi skriver CREATE PROCEDURE og det kjøres en gang. Hvis vi skal forandre noe i en eksisterende prosedyre, så bruker vi ALTER PROCEDURE istedenfor CREATE PROCEDURE.

Slik ser en procedure ut i SQL Server:

```
CREATE PROCEDURE [dbo].[CreateModule]
    @title nchar(20), @teacherID int, @adminID int, @institution nchar (15), @price float, @description nchar(30), @language nchar(20), @picture nvarchar, @subject nchar(20)
AS
BEGIN
    INSERT INTO [dbo].[Module] (title, teacherID, adminID, institution, price, [description], [language], picture,subject)
    VALUES (@title, @teacherID, @adminID, @institution, @price, @description, @language, @picture, @subject)
END
GO
```

Det finnes skjermbilder av alle prosedyrene i vedlegg.

Før vi fant ut av at vi burde lage procedures brukte vi Entity Framework. Det viste seg å abstraherte databasen for mye og gjøre ting på sin egen måte. Dette betyr at man kan få mye problemer med valgene dette rammeverket tar, som vi så må fortelle med .NET kode hvordan vi vil ha det. Da er det enklere og ryddigere å bare gjøre det rett i databasen.

¹⁵ Microsoft, 2017.

¹⁶ Microsoft, 2017.

¹⁷ Microsoft, 2017.

Oppretting av tabeller

Her bruker vi tabellen Person som et eksempel. De andre tabellene ligger i vedlegg.

Column Name	Data Type	Allow Nulls
personID	int	<input type="checkbox"/>
email	nvarchar(250)	<input checked="" type="checkbox"/>
password	nvarchar(20)	<input type="checkbox"/>
name	nvarchar(50)	<input type="checkbox"/>
surname	nvarchar(50)	<input type="checkbox"/>
postnr	nchar(4)	<input type="checkbox"/>
address	nvarchar(50)	<input checked="" type="checkbox"/>
username	nvarchar(50)	<input type="checkbox"/>
		<input type="checkbox"/>

Vi bruker inkrementell primary key på tabellene, noe som betyr at primary key alltid er autogenerert og unik. Det samme gjelder for de andre tabellene.

I SQL Server lagde vi tabellene ved å bruke de tekniske ressursene som SSMS tilbød oss. Om vi hadde kjørt et script for å lage denne tabellen, så hadde den sett slik ut:

```
CREATE TABLE [dbo].[Person](
    [personID] [int] IDENTITY(1,1) NOT NULL,
    [email] [nvarchar](250) NULL,
    [password] [nvarchar](20) NOT NULL,
    [name] [nvarchar](50) NOT NULL,
    [surname] [nvarchar](50) NOT NULL,
    [postnr] [nchar](4) NOT NULL,
    [address] [nvarchar](50) NULL,
    [username] [nvarchar](50) NOT NULL,
    CONSTRAINT [PK_Person] PRIMARY KEY CLUSTERED
```

Struktur i Visual Studio

Connection Strings

I appsettings.json, legger vi til “ConnectionStrings”. Det vil si en string som spesifiserer hvordan SQL klienten i .NET skal koble seg til databasen¹⁸. Appsettings.json er der man som oftest legger programmets konfigurasjon¹⁹.

Slik ser vår “ConnectionStrings” ut:

¹⁸ Affakes, no date.

¹⁹ Microsoft 2017.

```

3 "ConnectionStrings": {
    "DefaultConnection": "Server=tcp:wix.database.windows.net,1433;Initial Catalog=WixDatabase;Persist Security Info=False;User
    ID=*****;Password=*****;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;"}
},

```

Den er koblet til serveren i Azure via TCP med SQL login, som betyr brukernavn og passord. I Azure SQL må man også åpne opp firewall til godkjente IP addresser.

3.3.2.2 Arkitektur

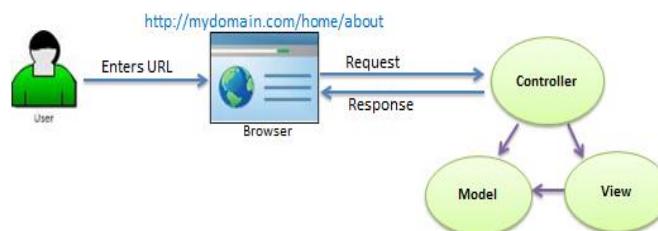
For å bygge vårt prosjekt så brukte vi MVC Arkitektur som base, med noen justeringer. MVC står for Model-View-Controller, og er en arkitektur og Microsoft har sitt eget MVC-rammeverk kalt MVC. MVC - arkitekturen deler man i tre ansvarsområder:

- Model er domeneobjekter og datastrukturer. Den brukes for å fortelle om tilstanden til webapplikasjonen.
- View er en mal for brukergrensesnitt. Den brukes sammen med modellen for å generere, for eksempel, HTML som vises til brukeren av applikasjonen.
- Controller-ene brukes til å håndtere web forespørsler, bygge modellen og returnere riktig View med Model

Fordelen med en slik arkitektur er at du har oversikt over de ulike funksjonalitetene på en ryddig måte. Det separerer logikken rundt henting og bygging av data på backend og bygging av frontend. Det er lett å vedlikeholde denne arkitekturen, for man kan lett finne frem og forstå de ulike kodebasene.

På backend delen av prosjektet brukte vi ikke rammeverket sin View motor for generering av HTML, fordi vi jobbet i to forskjellige lag, der frontend tok seg av all visning til brukeren via React. Kommunikasjonen mellom frontend og backend gikk derfor via web API fra backend der vi returnerte modellene som JSON til frontend.

20



Figur 33 - MVC arkitektur

Models and repositories

Modellene er strukturering av data. Repositoryene tar seg av databasoperasjoner. Diagrammene nede viser strukturen til både modeller og repositoryene i vårt prosjekt.

²⁰ Figur 33 - Tutorials Teacher, no date.



Figur 34 – Oversikt over modell og repositories

Vi skal bruke modell for Person og PersonRepository for å gi en enkel oversikt i hvordan de fungerer. Skjermbilder av alle de andre modellene og repositoriene ligger i vedlegg.

Modellen Person blir brukt når man skal hente en person eller flere personer. Grunnen for at vi har attributtet [JsonIgnore] over den er vi vil ikke at passordet til personene skal lekke ut og bli synlige for alle.

```

namespace BachelorOppgave.Data.Models
{
    5 references
    public class Person
    {
        //integers
        1 reference
        public int personID { get; set; }

        //strings
        0 references
        public string email { get; set; }

        [JsonIgnore]
        1 reference
        public string password { get; set; }
        0 references
        public string name { get; set; }
        0 references
        public string surname { get; set; }
        0 references
        public string postnr { get; set; }
        0 references
        public string address { get; set; }
        1 reference
        public string username { get; set; }
        1 reference
        public string Token { get; set; }

        1 reference
        public int? adminID { get; set; }
        1 reference
        public int? teacherID { get; set; }
    }
}

```

Repositoriet tar imot eller returnerer en modell. I metodene for å lage og oppdatere person, modules, lessons, admin og teacher så bruker vi også hjelpemodeller. Vi gjør det slik at swagger og andre verktøy hverken tar med de feltene vi ikke trenger på hver operasjon - som personID, som blir autogenerert - eller felter som vi ikke får fra databasen - som for eksempel Token på Person modell. Hjelpemodellen til Person er CreatePerson.cs.

```

namespace BachelorOppgave.Controllers.PersonsModels
{
    2 references
    public class CreatePerson
    {
        //strings
        0 references
        public string email { get; set; }
        0 references
        public string password { get; set; }
        0 references
        public string name { get; set; }
        0 references
        public string surname { get; set; }
        0 references
        public string postnr { get; set; }
        0 references
        public string address { get; set; }
        0 references
        public string username { get; set; }
    }
}

1 reference
public int CreatePerson(CreatePerson person)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();
        var result = connection.Execute(@"EXEC [dbo].[CreatePerson]
        @name = @name,
        @surname = @surname,
        @username = @username,
        @email = @email,
        @password = @password,
        @postnr = @postnr,
        @address = @address",
        person);
        return result;
    }
}

```

I repositories gjør vi en mapping fra modellen til parametrene på stored procedure i databasen for å få mer kontroll på hva som går inn i den.

Metoden CreatePerson i PersonRepository lager en kobling mot databasen i SQL serveren med “new SqlConnection” og på grunn av “using” lukkes koblingen automatisk når spørringen er ferdig.

Controllers

Kontrolleren håndterer brukerforespørslene - en HTTP request - som kommer fra React. Den bygger modellene og returnerer dem som Json.

Testing med “hard-code”

I starten av prosjektet benyttet vi oss av “hard-coding” for å teste om dataen vi lagde faktisk ble lagt til i databasen. For eksempel så kunne vi teste om det går an å slette et kurs, eller legge til et kurs.

Fordeler med å teste det i begynnelsen av prosjektet var at man kunne finne feil i en tidlig fase dersom noe ikke fungerte.

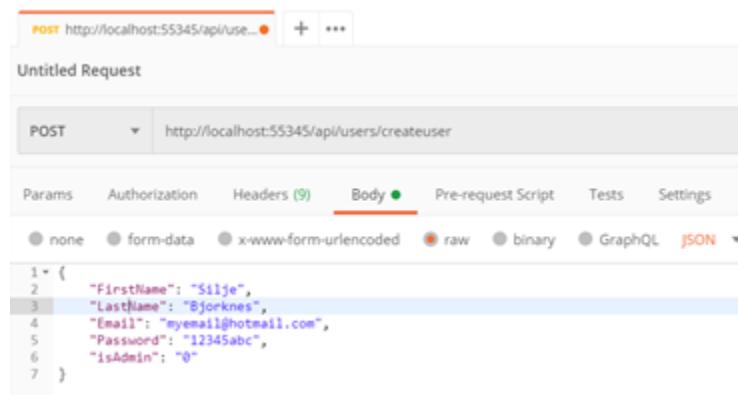
Et eksempel på hvordan “hard-coding” så ut i starten av prosjektet, da vi hadde model for bruker:

```
[HttpPost("createuser")]
0 references
public void CreateUser()
{
    dataRepository.CreateUser("Ola", "Nordmann", "olanordmann@email.com", "8282828282", true);
}
```

Hver gang vi kjørte koden, så ble en ny Ola Nordmann lagt til i tabellen. Vi åpnet URLen med riktig path, så fikk vi en liste med alle de Ola Nordmennene som ble lagt til i tabellen og de kunne slettes også.

Med dette på plass, så begynte med å bygge funksjonaliteten for å få APlet til å fungere uten bruk av statisk data, men med bruk av JSON objekter. Vi brukte verktøyet Postman for å teste ut funksjonene i APlet.

Her bruker vi av JSON for å teste funksjonaliteten for å lage en bruker:

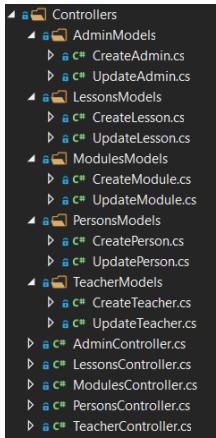


RESTful API

Etter å ha kjørt testen i Postman, begynte vi å fokusere på å bygge et RESTful API (Application Programming Interface - eller applikasjonsprogram grensesnitt) som bruker HTTP standarden for å gjøre forespørsler med bruk av HTTP-metodene GET, PUT, POST og DELETE.

Strukturen for controllers

Bildet under viser de 5 controllerne vi har for Admin, Lessons, Modules, Persons og Teacher, sammen med de hjelphemodellene.



For alle de controllere, så setter vi en rute (Route), som er endepunktet som React skal bruke for å kommunisere med controlleren. Så, for eksempel, hvis Route er [Route("api/[controller]")]
og navnet på controller er "AdminController", så er endepunktet api/Admin.

I alle controllers har vi attributtet [authorize], fordi det er nødvendig at brukerne av applikasjonen skal være logget inn, slik at appen vet hvem som er hvem. Det er unntak i de metodene der det står [AllowAnonymous]. Og i noen tilfeller er det forskjellige roller som brukeren som er logget inn må ha for å få lov til å bruke noen av de metodene. Vi går dypere i det når vi snakker om [autentisering og autorisasjon](#).

Vi bruker her Admin controller som et eksempel for å vise strukturen for controllerne.

```
[Route("api/[controller]")] //api/admin
[ApiController]
[Authorize(Roles = "Admin")]
[EnableCors("CORS")]
1 reference
public class AdminController : Controller
{
    private readonly AdminRepository _adminRepository;

    References
    public AdminController(AdminRepository adminRepository)
    {
        _adminRepository = adminRepository;
    }

    [HttpGet] //api/admin
    //to get a list of admins
    References
    public IEnumerable<Admin> GetAdmins()
    {
        return _adminRepository.FetchAdmin();
    }

    //to get one specific admin
    [HttpGet("{adminID}")]
    References
    public IActionResult FetchAdminByID(int adminID)
    {
        var admin = _adminRepository.FetchAdminByID(adminID);

        if (admin != null)
        {
            return Ok(admin);
        }

        return NotFound(new { Message = $"Module with id {adminID} is not available." });
    }
}
```

I bildet har vi konstruktøren "AdminController" som tar imot et repository via dependency injection.

Metodene GetAdmins() og FetchAdminByID(int adminID) bruker HTTP-metode GET. Grunnen til det er at de henter data fra databasen. GetAdmins() HTTP-metoden spesifiserer ikke en ID for henting av data, fordi den skal hente en liste av alle admin. I tilfelle den ikke eksisterer, så returnerer metoden en HTTP status 404 NotFound.

```

//to create admin
[HttpPost]
0 references
public IActionResult CreateAdmin(CreateAdmin createAdmin)
{
    var result = _adminRepository.CreateAdmin(createAdmin);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

//to update admin
[HttpPut("{adminID}")]
0 references
public IActionResult UpdateAdmin(int adminID, UpdateAdmin updateAdmin)
{
    updateAdmin.adminID = adminID;
    var result = _adminRepository.UpdateAdmin(updateAdmin);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

//to delete admin
[HttpDelete("{adminID}")]
0 references
public IActionResult DeleteAdmin(int adminID)
{
    var result = _adminRepository.DeleteAdmin(adminID);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

```

Metode CreateAdmin(CreateAdmin createAdmin) bruker HTTP-metode POST, fordi den skal legge til data i databasen.

UpdateAdmin(int adminID, CreateAdmin createAdmin) bruker HTTP-metode PUT, fordi den skal oppdatere data i databasen. Metoden UpdateAdmin tar to parametrere. Admin skal oppdateres med adminID fra stien til UpdateAdmin på controlleren, mens "createAdmin" tar ikke imot adminID i det hele tatt, siden den skal genereres i databasen automatisk.

DeleteAdmin(int adminID) bruker HTTP-metode DELETE, for å slette data fra databasen.

Betingelsen result>0 betyr at vi får HTTP status 200 Ok som svar når spørringen er vellykket.

Skjermbilder av de andre controllerne ligger i vedlegg. Men det er unntak som skiller noen av de controllerne fra hverandre.

API stien for lessons er api/module/{moduleID}/Lesson, noe som betyr at man må spesifisere hvilke module lesson tilhører til for så å få gjennomføre handlingene. Ellers fungerer de metodene på samme måte som på controller for admin.

I controller for Modules og for Lessons legger vi attributtet [AllowAnonymous] over de metodene som bruker HTTP-metode GET, fordi alle skal få tilgang til en liste av modules eller lessons eller en bestemt module eller lesson, uavhengig om man er logget inn eller ikke.

Attributtet [Authorize(Roles ="Admin")] over metodene for å lage, oppdatere og slette Module, Lesson, Admin og Teacher etablerer en regel slik at kun personer som er logget inn og som har rolle som admin har tillatelse til å gjøre det.

Når det gjelder lærere, så kan alle personer som bruker applikasjonen få sett informasjon om alle lærere eller en bestemt lærer når de søker om det. Derfor satt vi attributtet

[AllowAnonymous] over metodene GetTeacher og FetchTeacherByID, som ikke behøver å være logget inn for hente informasjonen.

Angående metoden UpdateTeacher (int teacherID, UpdateTeacher updateTeacher) både admin og selv læreren kan forandre dataen til læreren, derfor må personen være logget inn og enten ha rolle som admin eller som lærer, med riktig ID.

Enabling Cross-Origin Requests (CORS)

Da vi skulle prøve å koble backend og frontend for testing av controllerne, fikk vi problemer. Grunnen til det var at browseren har en sikkerhetsmekanisme som forhindrer en webside fra å sende forespørsler til et annet domene enn det som serverte websiden. Denne begrensningen kalles samme same-origin policy. Denne policyen forhindrer et skadelig nettsted fra å lese sensitive data fra et annet nettsted.

For å få applikasjonen til å fungere som vi ville, måtte vi aktivere CORS - Cross-Origin Requests (CORS). Det er en W3C-standard som lar en server reduserer sikkerheten ved å tillate forespørsler fra forskjellige domener.²¹

I metoden ConfigureServices i startup klassen la vi til følgende kode, slik at CORS-policy skulle blitt satt til alle applikasjonens endepunktene med den angitte opprinnelsen.

```
services.AddCors(options =>
{
    options.AddPolicy("CORS",
        corsPolicyBuilder => corsPolicyBuilder
            // SetIsOriginAllowed(x => _ = true)
            .AllowAnyOrigin()
            .WithOrigins("http://localhost:4200", "http://localhost:3000")
            // Apply CORS policy for any type of origin
            .AllowAnyMethod()
            // Apply CORS policy for any type of http methods
            .AllowAnyHeader()
            // Apply CORS policy for any headers
            .AllowCredentials()
    );
    // Apply CORS policy for all users
});
```

I metoden Configure, legger vi til UseCors. Den legger til en CORS middleware komponent og må være plassert mellom UseRouting og UseEndpoints, fordi vi bruker endpoint routing.²² Det viste seg at rekkefølgen er viktig, ellers fungerer ikke applikasjonen når vi kjører den.

```
app.UseRouting();

app.UseCors();

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
});
```

²¹ Microsoft, 2020.

²² Microsoft, 2020.

Attributtet [EnableCors(«Cors»)] etablerer CORS for de ønskende endepunktene, der “Cors” er navnet på policyen som ble brukt.

```
[Route("api/[controller]")]
[ApiController]
[Authorize]
[EnableCors("CORS")]

```

3.3.2.3 Publisering av APIet

For å få til koblingen mellom frontend og backend så trengte vi et sted der vi kunne publisere APIet. Vi fant løsningen i Microsoft Azure, som tilbyr ett år gratis tilbud for forskjellige skytjenester, blant annet publisering av API og database lagring. Med å bruke Azure, så får vi en Platform as a Service (PaaS) cloud computing-modell der en tredjepartsleverandør leverer maskinvare- og programvareverktøy. Dermed slipper vi blant annet å tenke på oppsett av virtuelle maskiner og oppdatering av operativsystem.

En annen grunn for å ha valgt Azure er at de tilbyr en gratis prøveversjon for ett år, noe som passet oss veldig bra. Den er også godt dokumentert og har enkle verktøy. Og selv om det finnes mange andre alternativer som, for eksempel, AWS og Google Cloud, er Azure godt integrert med Visual Studio og har veldig god støtte for .net og dotnet core.

Stegene for å publisere APIet i Azure ligger i vedlegg.

3.3.2.4 Autentisering og autorisasjon

Bakgrunn

I begynnelsen av prosjektet bestemte vi at vi skulle få til autentisering for admin, slik at admin måtte logge seg inn for å, blant annet, lage, oppdatere og slette moduler, leksjon og lærer. I løpet av utviklingsfasen så ble det klart at vi måtte tillate også for lærere å oppdatere sin egen informasjon og vi ville at alle som skulle ønske å åpne en modul eller sett profilen for en lærer, skulle få lov til det uten å måtte logge seg inn. Samt se en liste av alle moduler og alle lærere. Men vi kunne ikke la vanlige brukere i applikasjonen ha tilgang til det samme som admin, av sikkerhetsgrunner. Det er der autentisering- og autorisasjon kommer inn i bildet.

Autentisering

Autentisering og autorisasjon er to konsepter som er veldig nært hverandre og som brukes til å bygge sikkerhetsmekanisme i systemer og applikasjoner. Informasjonssikkerhet er praksisen med å beskytte informasjon mot uautorisert tilgang, bruk eller, til og med, modifisering.

Autentisering har med personens identitet å gjøre. Det gjelder å verifisere om den personen er hvem de sier de er. Autorisasjon har med hva personen er autorisert til å gjøre, basert på hvilke tillatelser han har i organisasjonen.

I første omgang, da vi hadde bestemt å lage et web API med bruk av ASP.Net Core, så vurderte vi om å bruke rammeverket Microsoft Identity²³ ²⁴ for å bygge funksjonaliteten for autentisering

²³ Microsoft, 2020.

²⁴ Microsoft, 2016.

og autorisasjon. I løpet av prosjektet, etter å ha bygget opp databasen med bruk av SQL server og Dapper, viste det seg at det skulle bli for komplisert å bruke Identity uten å implementere databasen med Entity Framework. Da skulle vi fått veldig mange flere tabeller som vi har i dag, og mange av dem skulle være helt unødvendige for prosjektet. Derfor bestemte vi å bruke enkel claims-based identity.²⁵ Det gjør at løsningen er mindre låst til Microsoft produkter og vil være enklere for noen med, for eksempel, en Java bakgrunn å forstå. Løsningen vi fant bruker JWT (JSON Web Token) autentisering med enkelte deler av Microsoft Identity uten å legge til hele rammeverket.

Med bruk av claims-based identity, så fikk vi jobbe med konsepter som claims, tokens og identitetsleverandører. Claims kan inneholde informasjon om brukeren, roller eller tillatelser, nyttig for å bygge en fleksibel autorisasjonsmodell. Tokenet inneholder ett eller flere krav, og hvert krav inneholder spesifikk informasjon. Tokenet er signert digitalt av token-utsteder når det opprettes, slik at det kan verifiseres ved mottakerenden.²⁶ Siden den er signert, trenger man ikke å spørre databasen for hver gang man leser ut informasjonen. ASP.NET Core sjekker signaturen, som ikke vil være gyldig hvis noen har endret på tokenet.



Figur 35 - Token²⁷

JSON Web Token (JWT) er et dataformat for brukerinformasjon i OpenID Connect-standarden²⁸ – en åpen standard (RFC 7519) som definerer en måte for sikker overføring av informasjon mellom parter som et JSON-objekt²⁹. Den håndterer autentisering og er bygget over OAuth 2.0-protokollen som er en industristandard protokoll for autorisasjon³⁰.

Med JWT autentisering, gir en klient et JSON Web Token, og tokenet vil bli validert mot en nøkkel, sertifikat eller en ekstern tjeneste.

Brukeren logger seg først på autentiseringsserveren (Authentication Server) ved å bruke serverens påloggingssystem (for eksempel, med å bruke brukernavn og passord eller en tredjepart identifisering leverandører som Facebook, Google, Twitter, og så videre). Autentiseringsserveren oppretter deretter et JWT tokenet og sender det til brukeren. Når

²⁵ Watmore, 2020.

²⁶ Krawczyk, 2015.

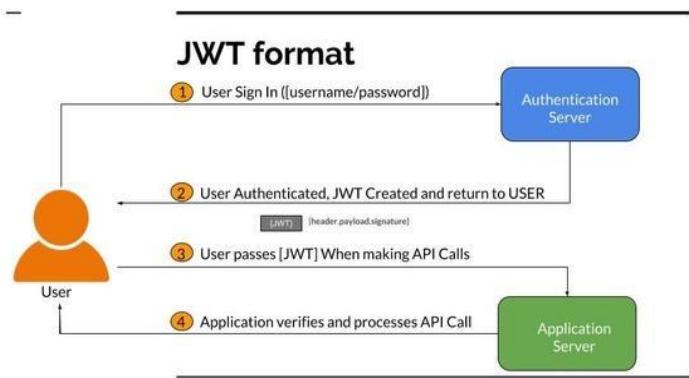
²⁷ Figur 35 Peipman, 2013

²⁸ NGINX, no date.

²⁹ JWT, no date.

³⁰ Internet Engineering Task Force (IETF), 2012.

brukeren kaller APIet til applikasjonen, sender brukeren JWT tokenet sammen med API-kallet. I dette oppsettet vil applikasjonsserveren være konfigurert til å bekrefte at det innkommende JWT er opprettet av autentiseringsserveren.³¹



Figur 36 – JWT format³²

På grunn av hvordan databasen ble strukturert, med brukernavn og passord på Person model, begynte vi å legge autentisering funksjonaliteten koblet til model Person, istedenfor Admin model.

```

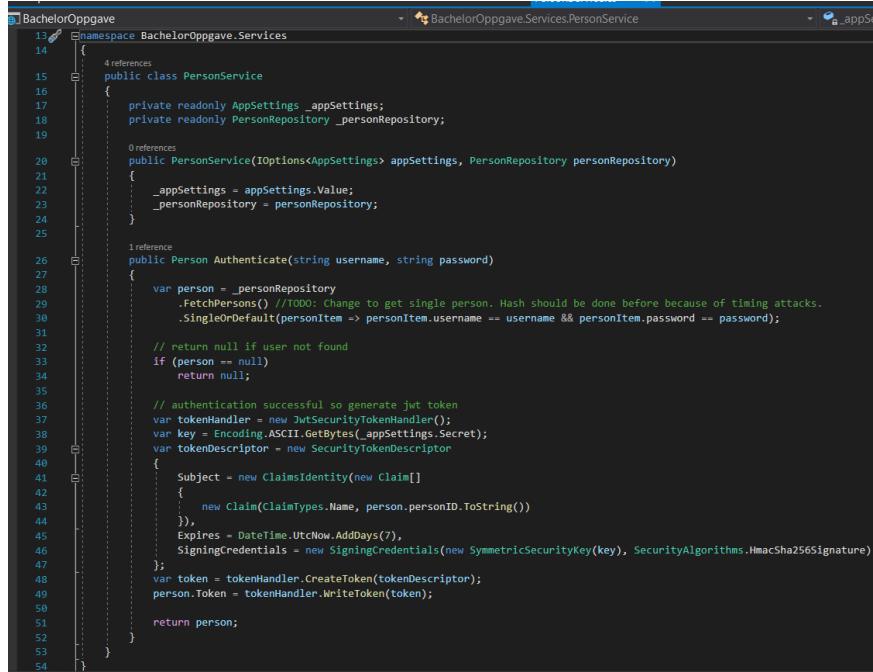
BachelorOppgave
1  using System.Text.Json.Serialization;
2
3  namespace BachelorOppgave.Data.Models
4  {
5      public class Person
6      {
7          //integers
8          public int personID { get; set; }
9
10         //strings
11         public string email { get; set; }
12
13         [JsonIgnore]
14         public string password { get; set; }
15         public string name { get; set; }
16         public string surname { get; set; }
17         public string postnr { get; set; }
18         public string address { get; set; }
19         public string username { get; set; }
20         public string Token { get; set; }
21
22         public int? adminID { get; set; }
23         public int? teacherID { get; set; }
24
25     }
26
27 }
```

³¹ Walke, 2019.

³² Figur 36 – Prashant, 2019

I eksisterende modell Person, la vi til string Token. Attributtet [JsonIgnore], over Password, forhindrer at passordet blir serialisert og returnert når vi får svar fra APIet.

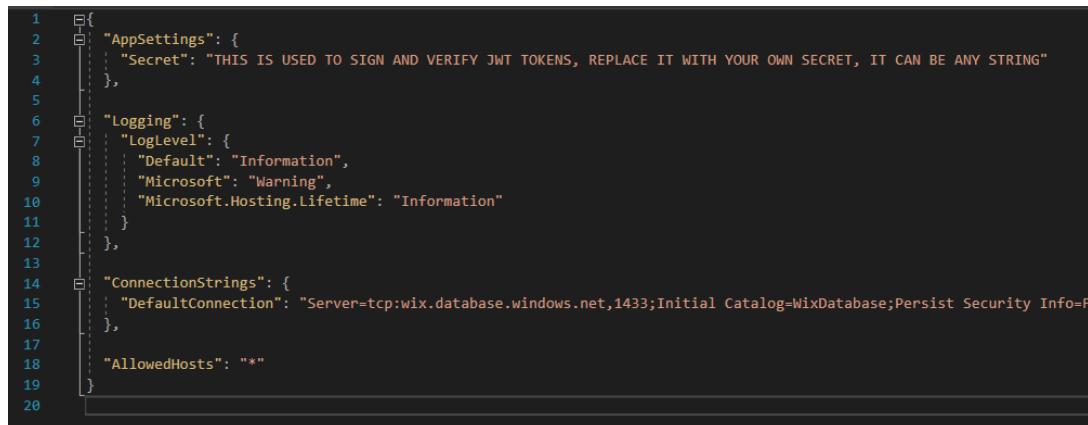
Vi lagde en ny klasse som heter PersonService, som inneholder en metode for å autentisere brukeropplysninger (user credentials) og returnere et JWT-token, og en metode for å få alle brukere i applikasjonen.



```
13  namespace BachelorOppgave.Services
14  {
15      public class PersonService
16      {
17          private readonly AppSettings _appSettings;
18          private readonly PersonRepository _personRepository;
19
20          public PersonService(IOptions<AppSettings> appSettings, PersonRepository personRepository)
21          {
22              _appSettings = appSettings.Value;
23              _personRepository = personRepository;
24          }
25
26          public Person Authenticate(string username, string password)
27          {
28              var person = _personRepository
29                  .FetchPersons() //TODO: Change to get single person. Hash should be done before because of timing attacks.
30                  .SingleOrDefault(personItem => personItem.username == username && personItem.password == password);
31
32              if (person == null)
33                  return null;
34
35              // authentication successful so generate jwt token
36              var tokenHandler = new JwtSecurityTokenHandler();
37              var key = Encoding.ASCII.GetBytes(_appSettings.Secret);
38              var tokenDescriptor = new SecurityTokenDescriptor
39              {
40                  Subject = new ClaimsIdentity(new Claim[]
41                  {
42                      new Claim(ClaimTypes.Name, person.personID.ToString())
43                  }),
44                  Expires = DateTime.UtcNow.AddDays(7),
45                  SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
46              };
47              var token = tokenHandler.CreateToken(tokenDescriptor);
48              person.Token = tokenHandler.WriteToken(token);
49
50              return person;
51          }
52      }
53  }
```

Ved vellykket autentisering genererer Authenticate metoden et JWT (JSON Web Token) ved bruk av JwtSecurityTokenHandler -klassen, som er signert digitalt ved hjelp av en hemmelig symmetrisk nøkkel (secret) lagret i appsettings.json. JWT-tokenet returneres til klientapplikasjonen som må inkludere det i HTTP Authorization bearer header for påfølgende sikre forespørsler.

Filen appsettings.json skal se ut som vist i bildet under. Der det står Secret, skal vi lage vår egen nøkkel.



```
1  {
2      "AppSettings": {
3          "Secret": "THIS IS USED TO SIGN AND VERIFY JWT TOKENS, REPLACE IT WITH YOUR OWN SECRET, IT CAN BE ANY STRING"
4      },
5
6      "Logging": {
7          "LogLevel": {
8              "Default": "Information",
9              "Microsoft": "Warning",
10             "Microsoft.Hosting.Lifetime": "Information"
11         }
12     },
13
14     "ConnectionStrings": {
15         "DefaultConnection": "Server=tcp:wix.database.windows.net,1433;Initial Catalog=WixDatabase;Persist Security Info=False;User ID=wix;Password={REDACTED};MultipleActiveResultSets=true"
16     },
17
18     "AllowedHosts": "*"
19 }
```

Vi lager en ny model, som heter AuthenticateModel.cs, som ser slik ut:

```
BachelorOppgave
1   using System.ComponentModel.DataAnnotations;
2
3   namespace BachelorOppgave.Data.Models
4   {
5       public class AuthenticateModel
6       {
7           [Required]
8           public string Username { get; set; }
9
10          [Required]
11          public string Password { get; set; }
12      }
13  }
14
```

Den definerer parameterne for innkommende forespørsler til route /api/authenticate.

Attributtet [Required] angir både brukernavn og passord som obligatoriske felt, så hvis en av disse mangler returneres en feilmelding fra APlet.

I startup.cs, inkluderte vi flere namespace, som Microsoft.IdentityModel.Tokens og Microsoft.AspNetCore.Authentication.JwtBearer :

```
BachelorOppgave
1   using BachelorOppgave.Data;
2   using BachelorOppgave.Helpers;
3   using Microsoft.AspNetCore.Builder;
4   using Microsoft.AspNetCore.Hosting;
5   using Microsoft.Extensions.Configuration;
6   using Microsoft.Extensions.DependencyInjection;
7   using Microsoft.Extensions.Hosting;
8   using Microsoft.OpenApi.Models;
9   using Microsoft.IdentityModel.Tokens;
10  using System.Text;
11  using Microsoft.AspNetCore.Authentication.JwtBearer;
12  using BachelorOppgave.Services;
```

Så legger vi til konfigurasjon så at ASP.Net core forstår hvordan autentiseringen og JWT er satt opp:

```
var appSettings = appSettingsSection.Get<AppSettings>();
var key = Encoding.ASCII.GetBytes(appSettings.Secret);
services.AddAuthentication(authenticationOptions =>
{
    authenticationOptions.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    authenticationOptions.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(jwtBearerOptions =>
{
    jwtBearerOptions.RequireHttpsMetadata = false;
    jwtBearerOptions.SaveToken = true;
    jwtBearerOptions.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(key),
        ValidateIssuer = false,
        ValidateAudience = false
    };
});
```

I metoden Configure, legger vi til UseAuthentication og UseAuthorization. Den første legger til autentisering middleware komponent slik at brukerne må være logget inn for å få tilgang til

ressursene i endpunktene. Den andre gir tillatelse til en bruker for å ha tilgang til ressursene. Siden vi bruker endpoint routing, må UseAuthentication bli plassert (i) etter UseRouting, slik at informasjon om route er tilgjengelig for autentiseringsavgjørelser og (ii) før UseEndpoints, slik at brukere skal være autentiserte før de har tilgang til de endpoints-ene.³³ UseAuthorization() skal bli plassert rett etter UseAuthentication().³⁴ UseCors, UseAuthentication, and UseAuthorization må alltid være plasserte i den rekkefølgen.

```
app.UseRouting();
app.UseCors();
app.UseAuthentication();
app.UseAuthorization();

app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
});
```

```
[Route("api/[controller]")]
[ApiController]
[Authorize]
[EnableCors("CORS")]
[AllowAnonymous]
```

I alle kontrollerne, legger vi til attributter [Authorize]. Det vil si at brukeren må ha riktig rolle for å få lov til å bruke de metodene i controlleren, med unntak av de metodene under attributtet [AllowAnonymous].

I PersonController, i tillegg til de andre metodene, legger vi til metoden for å logge seg inn, som skal se slik ut:

```
[AllowAnonymous]
[HttpPost("authenticate")]
0 references
public IActionResult Authenticate([FromBody]AuthenticateModel model)
{
    var person = _personService.Authenticate(model.Username, model.Password);

    if (person == null)
        return Unauthorized(new { message = "Username or password is incorrect" });

    return Ok(person);
}
```

Den har attributtet [AllowAnonymous], som betyr at alle kan ha tilgang til den og prøve å logge seg inn med sine brukeropplysninger. Men når man prøver å bruke de andre metodene før man er autentisert, så skal man få «Unauthorized» som svar.

Vi testet funksjonaliteten både i Swagger og i Postman - skjermbildene ligger i vedlegg.

Potensiale for videre utvikling

Med autentisering funksjonaliteten på plass slik som den er, så er passordene lagret i databasen i klart tekst. Hash og salt er ikke implementert enda, da vi ønsket å kunne endre passordet enkelt i databasen under utvikling. For sikkerhetsgrunner så må hash og salt utvikles i fremtiden, med mindre det blir bestemt å for eksempel gjenbruke eksisterende brukere i Active Directory. Vi har ikke hatt den diskusjonen med produkteieren heller.

Med bruk av hashing, så blir data av hvilken som helst størrelse kartlagt til en fast størrelse ved hjelp av en algoritme, som for eksempel, SHA-256. Det er brukt for å garantere at informasjonen som blir sendt beholder sin autentisitet. Når passordet blir hashet kryptert, så

³³ Microsoft, 2020.

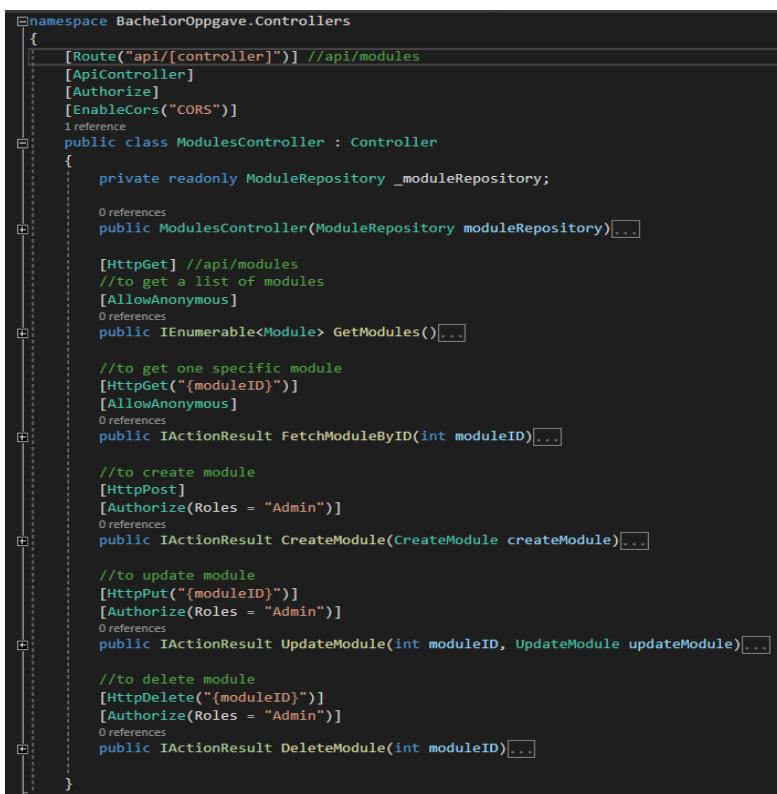
³⁴ Microsoft, 2020.

kan en person annen enn eieren av passordet se hash-verdien etter å ha dekryptert den med å bruke samme algoritmen, men de kan ikke se hva passordet er. De ser kun hashen.³⁵

Hvis vi kun bruker hashing, så blir det mulig å se når to personer har samme passord, selv om man ikke kan se hva passordet er. Derfor er lurt også å bruke salt, som et en unik verdi som er lagt til passordet før den blir hashet, slik at den blir mer kompleks og mindre utsatt for “brute force” angrep - når en datamaskin prøver alle mulige kombinasjoner av bokstav og nummer for å prøve å finne ut ett passord.³⁶

Autorisasjon

For å tilordne tillatelser til brukere basert på deres rolle i en organisasjon bruker vi Role-Based Access Control (RBAC) Model. I prosjektet vårt, skal kun de som har rolle “admin” få lov til å lage, oppdatere eller slette moduler, leksjoner og lærere. Roller skal bli satt i controllers:



```
namespace BachelorOppgave.Controllers
{
    [Route("api/[controller]")] //api/modules
    [ApiController]
    [Authorize]
    [EnableCors("CORS")]
    1 reference
    public class ModulesController : Controller
    {
        private readonly ModuleRepository _moduleRepository;

        0 references
        public ModulesController(ModuleRepository moduleRepository)...

        [HttpGet] //api/modules
        //to get a list of modules
        [AllowAnonymous]
        0 references
        public IEnumerable<Module> GetModules()...

        //to get one specific module
        [HttpGet("{moduleId}")]
        [AllowAnonymous]
        0 references
        public IActionResult FetchModuleByID(int moduleId)...

        //to create module
        [HttpPost]
        [Authorize(Roles = "Admin")]
        0 references
        public IActionResult CreateModule(CreateModule createModule)...

        //to update module
        [HttpPut("{moduleId}")]
        [Authorize(Roles = "Admin")]
        0 references
        public IActionResult UpdateModule(int moduleId, UpdateModule updateModule)...

        //to delete module
        [HttpDelete("{moduleId}")]
        [Authorize(Roles = "Admin")]
        0 references
        public IActionResult DeleteModule(int moduleId)...
    }
}
```

Som bildet viser, ved å bruke [AllowAnonymous] på metodene, så kan hvem som helst få tilgang til alle moduler og hente et bestemt modul, selv om man ikke er logget inn. Hvis man er logget inn og har rolle som admin, så kan man lage, oppdatere eller slette moduler, på grunn av [Authorize(Roles = “Admin”)] attributtet.

Samme logikken ble brukt for leksjoner.

For lærere så brukte vi en lignende løsning. Kun admin kan lage og slette lærer, men alle kan få tilgang til en liste med alle lærere og hente en bestemt lærer.

Når det gjelder oppdatering av lærer, så bestemte vi at selv lærere skal få lov til å oppdatere egne informasjon. Derfor er koding i den bestemte metoden litt annerledes:

³⁵ Nohe, 2018.

³⁶ Nohe, 2018.

```

//to update teacher
[HttpPut("{teacherID}")]
0 references
public IActionResult UpdateTeacher(int teacherID, UpdateTeacher updateTeacher)
{
    //Admin can change every teacher, but teacher can only modify their own data.
    //Run check, in case you are not admin, to see if we own the teacher item we try to modify.
    if(!base.User.IsInRole("Admin"))
    {
        //Get personId from claims:
        var personId = base.User.Claims
            .Single(claim => claim.Type == ClaimTypes.Name)
            .Value;

        //Check the teacher in database that we want to change
        var teacherItem = _teacherRepository.FetchTeacherByID(teacherID);

        //If teacher is not the logged in person, then we are not allowed to modify the teacher item we don't own.
        if (teacherItem?.personID.ToString() != personId)
        {
            throw new UnauthorizedAccessException();
        }
    }
}

```

Her må du være logget inn på grunn av attributt på controller-nivå. Dersom du er admin, har du lov til å oppdatere alle lærere, men dersom du er ikke er admin, gjør koden en ekstra sjekk, som vist over. Denne sjekken passer på at du bare kan oppdatere din egen bruker.

3.4 Testing

3.4.1 Brukertesting av prototype

Vi har valgt å skrive om brukertesting i prosessdokumentet fremfor i en egen testrapport. Brukertesting er en viktig del av den første fasen og den er avgjørende for hvordan det videre produktet utfolder seg, derfor valgte vi å sette fokus på den i produktdokumentasjonen. Delkapittelet beskriver brukertesting fra et teoretisk perspektiv og våre valg som viker fra teorien. Til slutt er det link til brukertest dokumentet som beskriver spørsmålene og svarene fra testen.

Brukertesting

Når vi brukertester, så er vi ute etter å analysere og måle opplevelsen av produktet som helhet, eller en mindre del. Opplevelsen av et produkt er avgjørende for å se om produktet er mulig å bruke, om det er noen som har lyst til å bruke produktet eller om det finnes feil ved produktet.

Når man brukertester, så får man mye nyttig data som er med på å påvirke utviklingsprosessen videre. Derfor er det viktig å brukerteste så tidlig som mulig. Dette gjør vi ved å brukerteste en prototype eller en MVP. Du kan lese om prototypen i [kapittel 3.2.4](#).

I vårt tilfelle var den funksjonelle beskrivelsen vi fikk av oppdragsgiver så ledende og konkret at den utformet brukertesting annerledes. Det vi ønsket å lære fra brukertestene var om flyten og opplevelsen ga mening fra et brukergrensesnitt- og et universelt utforming-perspektiv. Det var med andre ord ikke nødvendig å teste “proof of concept”. Dette gjorde vi fordi vi var trygge på applikasjonen var av lav kompleksitet fra et brukerperspektiv.

Brukertestdokumentet (om dokumentet)

I vårt brukertestdokument utarbeidet vi bare de mest nødvendige tilleggsopplysningene. Dette omhandlet en kort prosjektbeskrivelse, bakgrunn for oppgaven, oppdragsgiver og mål for applikasjonen. Vi konstaterte også i brukertest dokumentet at noen bestemte personopplysninger - som kjønn, alder eller yrkeserfaring - ikke ville bli samlet eller lagret. Dette ble det tatt en avgjørelse på av gruppen basert hvordan vi skulle anvende dataene.

Gjennomførelsen

Brukertesting ble gjennomført i Figma (se verktøybeskrivelse) der brukeren gikk gjennom prototypen av brukersiden og administrasjonsiden. Vi var nøye på å ikke rettlede testeren gjennom sidene, men få konstant feedback gjennom å få brukeren til å tenke høyt. Vi konstaterte at testeren ikke ble evaluert og at det ikke var noen riktig eller galt svar. Ved siden av spørsmålene vi stilte til hver side hadde vi også fokus på dialog for å forstå nøyere hva brukeren tenkte. I tillegg til en som noterte ned spørsmål og svar hadde vi også en som holdt øye med musebevegelsene til brukeren og noterte ned interessante observasjoner.

Brukertestene

Brukertest 1

Testleder

Testleder var Elias Pederstad. Han hadde ansvar for å forberede og utføre testene. Det gjaldt å sette opp en PC og miljø som var klart for en bruker. Testlederen forklarte at intensjonen med testen var ikke at brukeren ikke ble testet, men selve applikasjonen. Testlederen forklarte at brukeren skulle fortelle hva en tenkte høyt. Testlederen gir også instrukser der det er nødvendig og stiller spørsmål for mer utdypende svar.

Observatør

Observatøren var Alina Zielinska. Hun hadde ansvar for å fylle inn tabellen, tar notater og stiller oppfølgingsspørsmål.

Tilbakemelding

Side	Tilbakemelding	Forbedringspotensial
Logg inn side	Estetisk fin, og forståelig hva knappene uttrykket.	Trenger en registreringknapp
Modulside	Oversiktlig	Ha bilder i modulene som fanger lyet
About side	God layout!	Innholdet må endres

Modul info side	Veldig viktig med en slik side, gir god kunnskap	Brukeren blir oppmerksom på font, skriftstørrelse og ber om endringer.
Modul tekst side	Poengterer tekststørrelse og font. Liker oversikten som brødsmulene gir	Endre farge fra blå til grønn for å vise hvilken side brukeren er på.
Modul video side	Forståelig	
Modul quiz side	Forståelig	
Admin Modulside	For mange redigering muligheter.	Flytte redigeringen av modulkortet til modul infosiden siden.
Admin About side	Enkelt og greit.	
Admin Modul info side	Spørsmål om ”instructur” skal redigeres eller legges til.	Forslag om å velge fra instruktørene som eksisterer i systemet.
Admin Modul tekst side	Forståelig.	
Admin Modul video side	Forståelig	
Admin Modul quiz side	Litt lite funksjonalitet. Men fungerer slik det skal.	

3.4.2 Integrasjonstesting

Inngangskriteriet for begynnelse på integrasjonstest er fullføring av enhetstesting.

I integrasjonstesting skal vi teste alle funksjonaliteter. Alle endepunktene og utfall fra disse (mulige scenarioer der det blir returnert data). Alt skal testes mot databasen. Her konsentrerer vi oss hovedsakelig på businesslogikken i programvarearkitekturen. Før testing måtte vi forstå API-kravene grundig. Det er veldig viktig å kjenne formålet med APIet. Det er et solid grunnlag for å forberede testdataene for input og output. Det hjelper også med å definere bekreftelsesmetoden.

API-testing krever både positive og negative tester for å sikre at API fungerer som den skal. Siden API-testing er betraktet som en type svartboks-testing, er begge typer testing drevet av input og output data. Til å planlegge bruk av data trengte vi også tenke på grenseverdier. Vi måtte passe på at alle gjensidige avhengigheter fungerer som de skal. I integrasjonstesting går

vi gjennom alle endepunktene i APlet. For å bekrefte testing sammenligner vi innholdet i testresultatet med den informasjonen som er forventet.

Vi måtte undersøke arkitekturdesignet til applikasjonen og identifisere de kritiske modulene, prioritere de som var mest kritiske for oss og kjøre de sammen. Til å utføre integrasjonstesting, brukte vi Postman. Se vedlegg 6.10.

3.4.3 Systemtesting

Systemtesting betyr å teste systemet som en helhet. Alle modulene / komponentene er integrert for å bekrefte om systemet fungerer som forventet eller ikke.

Systemtesting utføres etter integrasjonstesting. Dette spiller en viktig rolle i å levere vår minste brukbare produkt (Minimum Valuable Product- MVP) med høy kvalitet.

Systemtesting innebærer å teste programvarekoden på følgende:

- Å sjekke hvordan komponenter interagerer med hverandre og med systemet i sin helhet.
- Kontrollere grundig testing av hver input i applikasjonen for å se etter ønskede output.
- Testing av brukerens opplevelse med applikasjonen.

Systemtesting utførte vi mot databasen. Før testing måtte vi definere test data og spesifisere forventede resultater. Vi lagde test cases i Excel Document. Etterpå ble brukertester kjørt manuelt. Som resultat fikk vi et dokument med bugs og tester som ble bestått. Se vedlegg 6.11.

I Systemtesting representerer hele systemet et testobjekt. Systemtester validerer ikke bare designet (UI, UX), men også aspekter som atferds-, avhengigheter og brukervennlighet.

Programvareproduktet må oppfylle kriterier for å bli akseptert av en bruker, en kunde eller et annet system. Derfor er suksess for hvert prosjekt avhengig av evnet til en utviklingsgruppe til å møte kundens behov. Se kravspesifikasjon i vedlegg 6.3.

Kriterier er unike for hver brukerhistorie og definerer funksjonens atferd fra perspektivet til sluttbrukeren. Velskrevne kriterier hjelper til å unngå uventede resultater i slutten av en utviklingsfase og sikre at alle interessenter og brukere er fornøyde med det de får.

For å akseptere systemet bør det ikke være noen feil med kategorier A og B. Det kan være ikke mer enn 3% feil med kategori C. Og ikke mer enn 7% feil med kategori D. Generelt bør det være avtalt mellom kunde og utviklingsfirmaet om akseptansekriterier og muligens feilprosent. I vårt tilfelle er det oss som har bestemt om slike tall selv, basert på vår erfaring.

Generelt har vi 95 tester i systemtesting. Vi prøvde å beskrive flest mulig test cases. Men, selvfølgelig, det finnes mye mer som kan dukke opp i utviklingsprosessen og i diskusjonen med kunde.

82 tester var bestått, 6 tester ble feilet og 7 teste cases er ikke tilgjengelig (funksjonaliteten ble ikke gjennomført). Det er bare 6,3% av tester som feilet. Alle de feilene har høy prioritet. Det

betyr at de er ganske viktig for systemet. Liste med alle test cases, feil og kategori kan man se i vedlegg 6.11

3.5 Utfordringer og refleksjoner

En av de viktigste lærdommene vi fikk var at det ikke alltid er det man velger på først som er best. Vi lærte også at det er viktig å gjøre undersøkelser før man tar valg. I begynnelsen av prosjektet begynte vi å lære oss Angular og .Net, men fant fort ut av at det ikke var det som var beste for oss, så byttet vi til React og .Net Core. Vi var heldige siden vi var i startfasen og det ikke førte til store konsekvenser. Det samme gjaldt valg av rammeverket for databasen, som ble byttet fra Entity Framework til Dapper.

Det er krevende å jobbe med backend og frontend uten erfaring fra før. På frontend delen var det krevende med JavaScript og React. Det gikk ganske greit i startfasen men det mer krevende etterhvert. Det var enkelt å bruke Bootstrap komponenter for å fylle opp med statisk innhold på websiden, men det var krevende å skrive “Conditional Rendering” siden vi hadde ikke erfaring med dette.

Vi hadde mangel på erfaring og det var vanskelig å be om hjelp fra noen på universitetet. Vi måtte derfor holde motivasjonen oppe og å jobbe selvstendige. På denne måten klarte vi å løse de problemene vi møtte. Vi var også bekymret for samspill mellom frontend og backend, samt arbeid med GitHub og publisering. Vi gikk gjennom forskjellige online kurs for å oppnå kompetanse på områdene. Vi gikk igjennom JavaScript kurs på Codeacademy, React kurs på ReactWarrios (<https://reactwarriors.com/>) og .Net Core kurser på Udemy.

Vi lærte også om hvor viktig kommunikasjonen er i en gruppeoppgave som denne. Da prosjektet startet møttes vi på universitetet minst 3-4 ganger i uka og hadde derfor god kommunikasjon. Vi jobbet effektivt sammen og hjalp hverandre dersom noen satt fast. Etter en liten periode kom Korona utbruddet. Omstendighetene gjorde at vi fikk en ekstra utfordring da vi prøve å gjennomføre dette prosjektet, siden kommunikasjon mellom frontend og backend delene ble dårlig i begynnelsen av krisen. Det var vanskelig å koordinere online møter og planlegge hvem som skulle jobbe med hva. Vi møtte på vanskeligheter med databasen som ikke passet med frontend inputdata. Også APIet fungerte ikke på grunn av forskjell i utviklingsmiljøet.

Andre lærdommer vi fikk var at ting tok lengre tid enn forventet. Vi fikk problemer med SQL server, og feilmeldinger som gjorde at ting tok lengre tid. Vi lærte også viktigheten av sekvensdiagrammer, at det var viktig å kunne ha oversikt over hvilke funksjoner webapplikasjonen skulle ha.

3.6 Oppsummering av produktdokumentasjon

Hovedmålet for produktdokumentasjonen var å tilrettelegge for enkel videreutvikling ved eventuell overtakelse av prosjektet senere.

I produktdokumentasjonen går vi gjennom utviklingen av UI- utforming, frontend, backend og testing ved bruk av all lærdommen vi fikk under bachelorperioden. Vi brukte SQL Server for utviklingen av databasen, ASP.Net Core som rammeverk på backend og React som rammeverk på frontend. Det viste seg å være en god kombinasjon av disse for å oppnå målet vårt. UI-utforming ble utført i designet av brukergrensesnitt for brukertesting, design og prototyping, og evaluering.

Det er viktig med testing, for det viser om produktet ditt passer med kravspesifikasjonene. Gruppen gjennomførte brukertesting der vi fikk god tilbakemelding og bekreftelse på at vi hadde gjort godt forhåndsarbeid. Gjennom prosjektutviklings fasen utførte vi unit testing og integrasjonstesting, som ga oss god kodekvalitet. Vi gikk gjennom systemtesting for å teste at produktet vårt oppnådde målet med å være et «minst brukbar produkt» (Minimum Valuable Product- MVP).

4 Brukerveiledning

Brukerveiledningen er et dokument rettet mot kunden som mottar produktet. Dokumentet er en manual for hvordan produktet fungerer og kan skal hjelpe administratoren å komme i gang med å bruke det. Brukerveiledningen tar for seg én side av applikasjonen om gangen og går videre til den siden som vil være naturlig i en vanlig flyt. Den mest aktuelle flyten er; 1) opprettelse av modul, 2) Pålogging 3) gjennomførelse av modul.

4.1 Pålogging

For å logge inn har du tre valg. brukeren eller administrator kan skrive inn Email og passord og trykke "Submit" for å logge på. Ved førstegangsregistrering trykkes det på "Registrering" som vil åpne en ny side

The screenshot shows the OSLOMET login interface. At the top left is the OSLOMET logo. At the top right are links for 'Modules', 'About', and 'Login'. The main title 'Log in' is centered above two input fields: 'Email' and 'Password', each with its own input box. Below these is a checkbox labeled 'Check me out'. At the bottom are two yellow buttons: 'Submit' and 'Register'. A horizontal line separates this from a blue button labeled 'SIGN IN WITH FACEBOOK'. Above the 'SIGN IN WITH FACEBOOK' button, the text 'or signup with' is visible.

Figur 37 - Pålogging

Register page

Name

Surname

Phone

Email

Password

[Close](#)

[Save](#)

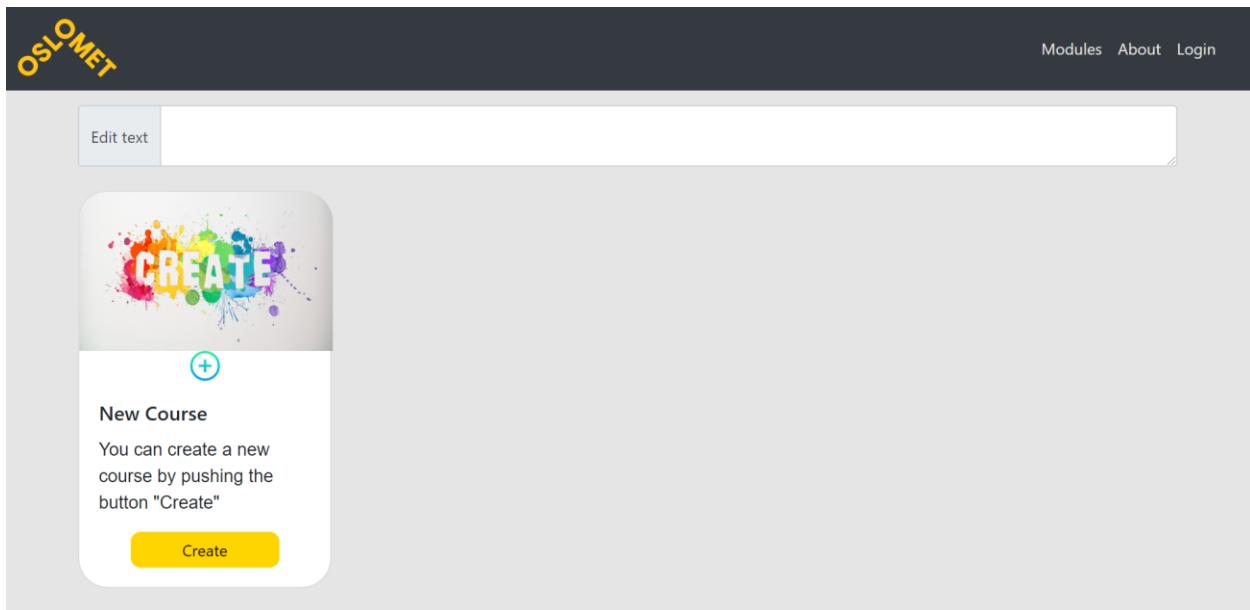
This is a screenshot of a registration form titled "Register page". The form contains five input fields: "Name", "Surname", "Phone", "Email", and "Password". Below the fields are two buttons: "Close" (grey) and "Save" (yellow). The entire form is enclosed in a black-bordered box.

Figur 38 - Registrering

4.2 Opprettelse av en ny modul av administrator

For å opprette en ny modul må du ha administratorrettigheter. Ved pålogging som administrator vil du bli møtt med landingssiden som inneholder alle modulene og muligheten til å opprette nye moduler. I figur 39 ser du landingssiden med kortet som tillater deg å opprette en ny modul. Øverst på siden under «Edit text» kan du legge til en overskrift eller en informativ tekst.

For å opprette en ny modul trykker du på den gule knappen på kortet hvor det står "Create". Da vil du komme til en ny side hvor du også får muligheten til å redigere hvordan kortet på landingssiden skal se ut.



Figur 39 – Opprette ny modul

Når du har trykket på "Create" vil du få opp en ny side (se figur 40). Denne siden kalles "Admin rediger modul side". Her kan du skrive navnet på kurset ditt i første tekstfelt, og i tekstfelt under kan du beskrive selve kurset. I tabellen nederst på siden kan du fylle inn pensumet studenten skal gjennomgå i modulen. Til høyre på siden kan du fylle ut informasjon om kurset, herunder hvor lenge det varer, hvilken institusjon som står bak, hvilket emne modulen tilhører, prisen på modulen og hvilket språk kurset er på. Under tekstfeltene på høyre side kan du velge hvem som er foreleseren for kurset. Her kan du trykke på bildesymbolet for å legge til et bilde av foreleseren og deretter skrive navn på foreleser, institusjon foreleseren tilhører og tittelen til foreleseren.

På denne siden (figur 40) er det totalt 5 gule knapper:

1. "Download schedule" tillater brukeren å laste ned hele pensumet.
2. "Save" lagrer innholdet du har fylt inn på siden, slik at du kan komme tilbake seinere å fullføre innholdet.
3. "Delete" sletter hele modulen. Denne knappes skal kun brukes dersom du ikke ønsker at kurset skal eksistere lenger. Du kan ikke få tilbake innholdet dersom du velger å bruke delete-knappen.
4. "Add content" er neste steg i å lage en modul. Knappen fører deg videre til sidene der du kan legge til inn innhold, enten det er tekst, video, spørsmål eller et annet format.
5. "Publish" bruker du når du er ferdig med utformingen av kurset. Når alt av informasjon og innhold er klart til å tas i bruk, er det denne knappen som tilgjengeliggjør modulen for studentene.

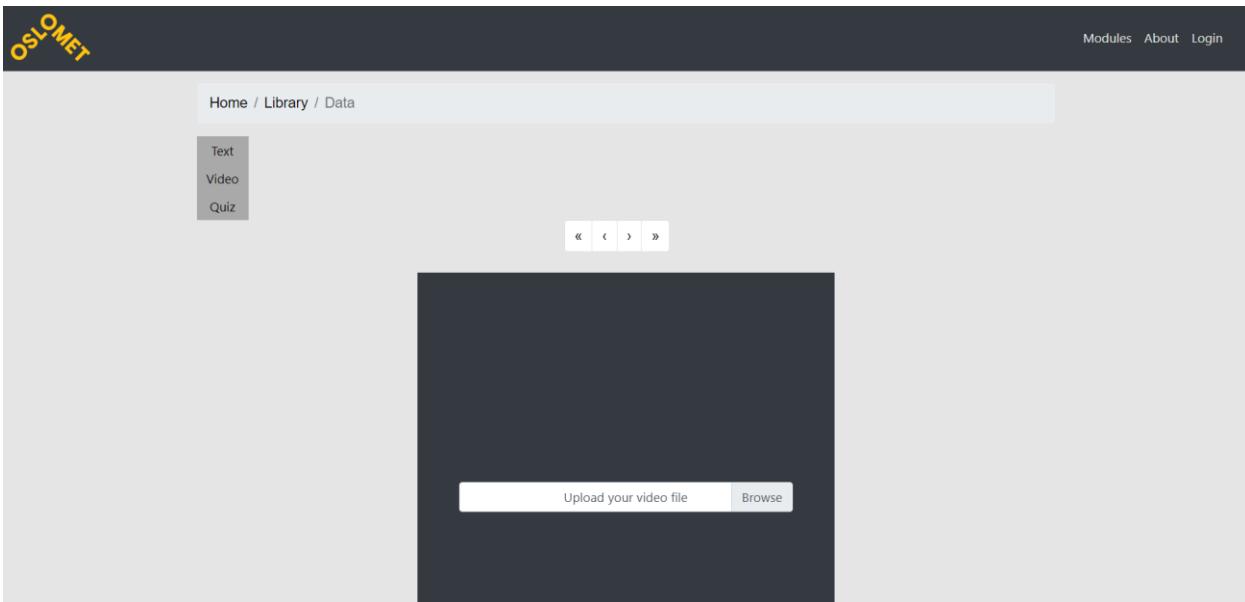
The screenshot shows a web-based course creation interface. At the top left is the OSLOMET logo. Top right links include Modules, About, and Login. The main area has several input fields:

- Name of your course:** A text input field with placeholder "Edit text".
- Describe briefly your course:** A large text area with placeholder "Edit text". To its right is a yellow button labeled "Add a content".
- Duration:** A dropdown menu set to "Whole course duration".
- Institution:** A dropdown menu set to "Presented institution".
- Subject:** A dropdown menu set to "Define subject".
- Price:** A dropdown menu set to "Price for the course".
- Language:** A dropdown menu set to "Course's main language".
- Describe lecturer of the course:** A text area with a small icon of a person speaking.
- Download schedule:** A yellow button.
- Describe schedule of the course:** A table with columns: Number, Name, and Duration. It has 5 rows for entries.
- Action buttons:** "Publish", "Delete", and "Save".

Figur 40 – Lage informasjonsside

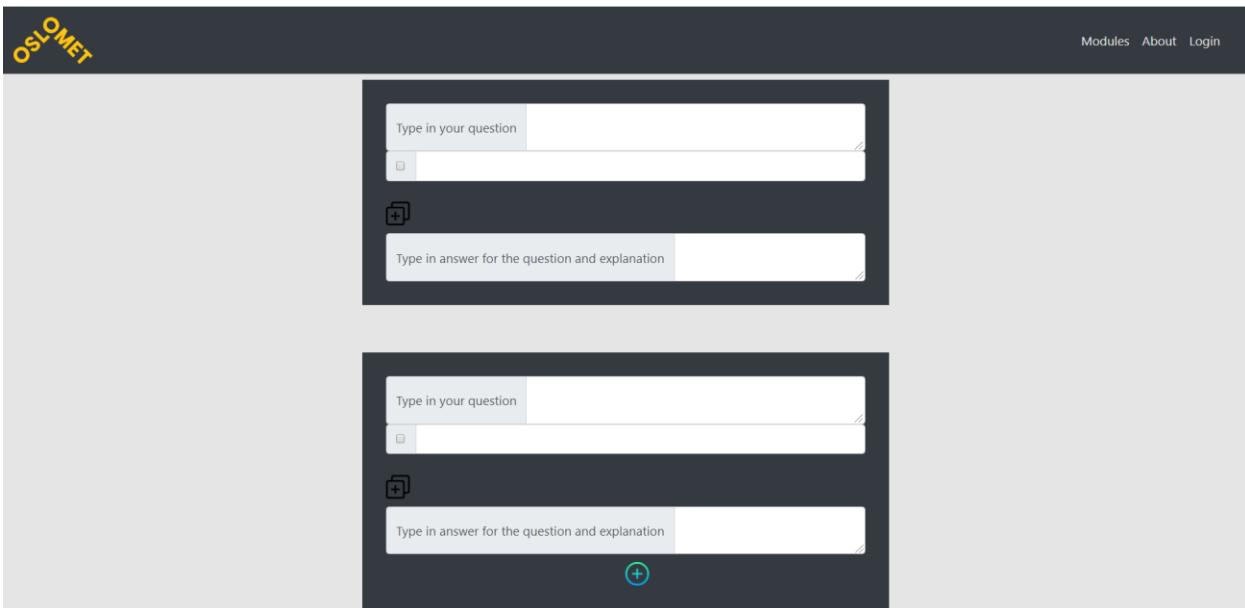
4.3 Lage innhold

Når du er i gang med å lage innhold i et kurs kan du velge mellom flere formater: Ett av formatene er video. For å laste opp en video velger du knappen merket "Video" i menylinjen på venstre side, slik du ser i figur 41 under. Da dukker det opp en trykkbar knapp midt på skjermen "Upload your video file" og "Browse". Her kan du enten dra og slippe en videofil inn i feltet for å laste opp videoen eller trykke browse for å åpne datamaskinens mappesystem. Det er også mulig å kopiere inn en youtube link som vil bli vist inne på kurssiden.



Figur 41 – Sett in video

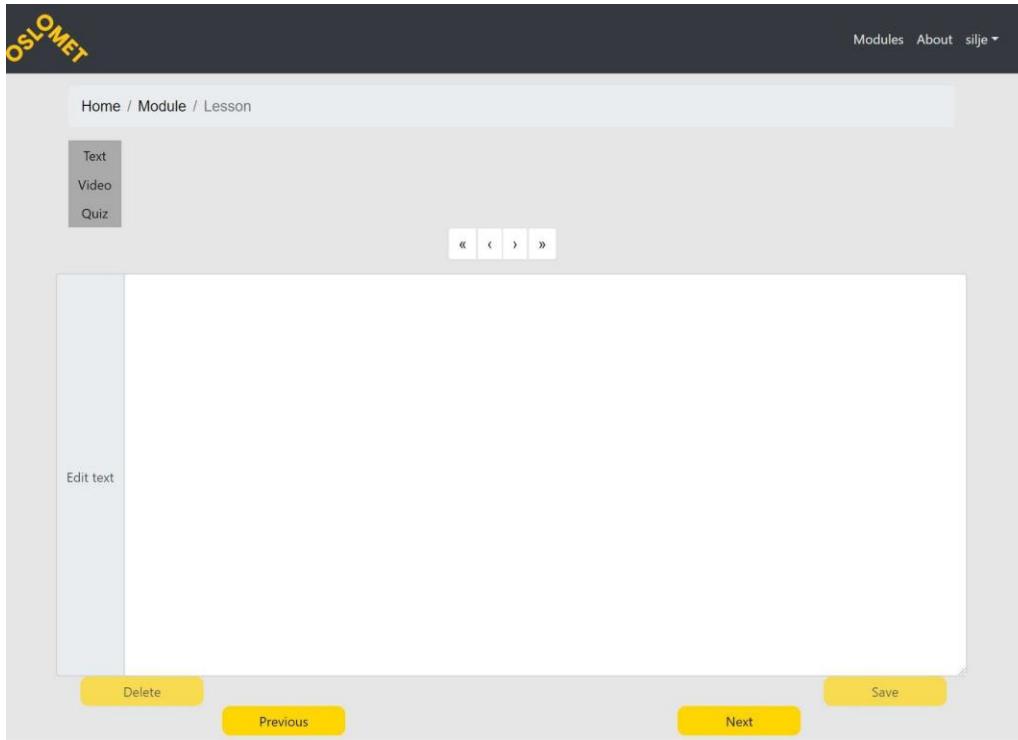
Et annet innholdsformat du kan velge er ”Quiz”. Da trykker du på knappen markert ”Quiz” øverst på venstre side av skjermen på figur 42.



Figur 42 – lage quiz

Når du har valgt Quiz-formatet vil siden se ut som figur 42 under. Her ser du hvordan hvert spørsmål tilhører en svart boks. I det første tekstfeltet kan du skrive inn ditt spørsmål. I tekstfeltet under kan du skrive et svaralternativ. For å legge til flere svaralternativer trykker du på det firkantede pluss-symbolet. Det nederste tekstfeltet i den svarte boksen skal være det riktige svaret. Dette svaret vil derimot se ut som de andre svaralternativene slik at du ikke avslører hvilket svaralternativ som er det riktige.

Formatet text har en enkel fremgangsmåte for å legge til innhold. Administrator er innholdseier skriver eller limer inn teksten som er ønsket. Vedkommende kan da enten slette eller lagre teksten ved å trykke på den gule knappen “Delete” eller “Save”.



Figur 43 – Lage tekst

4.4 Gjennomførelse av modul

Når en sluttbruker åpner nettsiden vil modul siden være det første vedkommende møter. Her kan brukeren velge tre veier å gå. 1. brukeren kan trykke på “About” øverst til venstre som vil ta brukeren til informasjonssiden om WIX prosjektet. 2. brukeren kan velge og logge inn (se 4.3). Da får brukeren mulighet til å beholde prosesjon i kursarbeidet og kommentere. 3. brukeren velger en av modulene som tar vedkommende videre til modul informasjon siden.

The screenshot shows the 'Modules' section of the OSLOMET website. At the top left is the OSLOMET logo. At the top right are links for 'Modules', 'About', and 'Login'. Below the header, the title 'Modules' is displayed, followed by the sub-instruction 'Select a module you would like to learn more about.' Four modules are listed in a grid:

Module Image	Creator Name	Duration	Description	Action
	Erika Gubrium	60 min	It will be name of the module	Learn more
	Aadne Gorstad	160 min	It will be next name of the module	Learn more
	Alina Zielinska	120 min	It will be next next name of the module	Learn more
	Alina Zielinska	120 min	It will be next next name of the module	Learn more

At the bottom of the page, there are links for 'About us', 'Contacts', 'Something', and 'Personal info'. A small note at the bottom left says 'Website made with '. The bottom right contains copyright information: 'Copyright © 2020 All Rights Reserved'.

Figur 44 – Modulsiden bruker

Når brukeren har valgt en modul kommer brukeren videre til informasjon siden til modulen. Her kan brukere lese tittelen til modulen, kort informasjon om modulen og innholdsfortegnelsen. Til høyre kan brukeren lese tekniske spesifikasjoner om modulen som hvor lang tid den tar, hvilken institusjon som har laget den, temaet for modulen, pris og hvilket språk modulen er på. Under blir det informert om hvilke mennesker som har vært deltagende i utviklingen av modulen. Når brukeren er klar for å starte kurset trykker vedkommende på den store gule knappen “Start the course”.

Header for the module

SContrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32. The standard chunk of Lorem Ipsum used since the 1500s is reproduced below for those interested. Sections 1.10.32 and 1.10.33 from "de Finibus Bonorum et Malorum" by Cicero are also reproduced in their exact original form, accompanied by English versions from the 1914 translation by H. Rackham.

Course topics

Number	Name	Duration
1	The Norwegian welfare state	4 min
1	The Norwegian welfare state	4 min
1	The Norwegian welfare state	4 min

Start the course

Duration	30 min
Institution	OsloMet
Subject	Social welfare
Price	Free
Language	English

Elias
lector
OsloMet

Alina Zielinska
lector
OsloMet

Bob Marley

Figur 45 – Informasjonsside modul

Når brukeren starter et kurs kan vedkommende bli møtt av fem forskjellige sider med innhold. På alle sidene kan brukeren trykke på de gule knappene “Previous” eller “Next” for å navigere videre i kurset. På toppen av siden er det også en navigasjonsbar som kan bli brukt til å navigere og få informasjon om hvor en er i kurset. Øverst til venstre er brødsmuler som kan også brukes til å navigere tilbake til informasjonssiden osv. De for forskjellige sidene brukeren kan bli møtt med er en tekst, video, PDF, PowerPoint eller quiz side. Det alle sidene har til felles er knappene beskrevet over.

The name of this course

The Norwegian Welfare State Ideals in tension Norway is a thin, stretched country in Scandinavia. It has a population of only 5 million people. Oslo, the capital, has about 650,000 inhabitants. The metropolitan area, which includes the densely populated areas around Oslo proper, has 1.7 million people. In Norway all political parties basically agree that the quality of health, education and social services should be as similar as possible throughout the country. The Norwegian welfare state has the following general characteristics: Benefits and services are financed through a re-distributive, general tax system. Those with lower incomes pay relatively less and those with higher incomes (up to a limit) pay relatively more. The welfare system is founded on the ideals of equality, social justice, social security, solidarity and social integration. These ideals are subject to positive and negative tensions. The goals of economic redistribution and social integration require high employment and economically active people. The focus on employment and employability may clash with the ideals of equality, social justice and social security. Resources Wikipedia. Oslo. World Population Review. Oslo population 2018. The World Population Review is a private, US based web site which presents demographic data in a convenient way. Image Lysefjord, Norway.

Previous Next

Write your comment

Comment

About us Contacts Something Personal info

Website made with Copyright © 2020 All Rights Reserved

Figur 46 – Tekstside

5 Referanseliste

Achard, Chris. Beginning Javascript for React Native - Switch / Case Statement. Hentet fra: <<https://caster.io/lessons/beginning-javascript-for-react-native-switch-case-statement>>

Affakes, Sebastian. Connection Strings Explained. Hentet fra: <<https://www.connectionstrings.com/connection-strings-explained/>>

Auth0. Introduction to JSON Web Tokens. Hentet fra: <<https://jwt.io/introduction/>>

Auth0. Role-Based Access Control. Hentet fra: <https://auth0.com/docs/authorization/concepts/rbac>

Bemis, Cole. Building a Checkbox Component with React and styled-components. 4. november 2018. Hentet fra: <<https://medium.com/@colebemis/building-a-checkbox-component-with-react-and-styled-components-8d3aa1d826dd>>

Bernheim, Laura. 7 Best WYSIWYG Web Builder Reviews. 29. oktober 2019. Hentet fra: <https://www.hostingadvice.com/how-to/wysiwyg-web-builder/>

Chauhan, Shailendra. Introduction to Entity Framework. 25. Juni 2011. Hentet fra: <<https://www.dotnettricks.com/learn/entityframework/introduction-to-entity-framework>>

Chinda, Glad. How To Build Custom Pagination with React. 12. desember 2019. <<https://www.digitalocean.com/community/tutorials/how-to-build-custom-pagination-with-react>>

Cummings, Neil. Complete Guide to Building an App with ASP.Net Core and React. Hentet fra: <<https://www.udemy.com/course/complete-guide-to-building-an-app-with-net-core-and-react/lecture/15865054#overview>>

Datatilsynet. Skytjenester. Hentet fra:<<https://www.datatilsynet.no/personvern-pa-ulike-områder/internett-og-apper/skytjeneste>>

Diku - Direktoratet for Internasjonalisering og kvalitetsutvikling i høyere utdanning. Hentet fra: <<https://diku.no/>>

Draft.js. Rich Text Editor Framework for React. Hentet fra: <<https://draftjs.org/>>

Forskningsradet. Hentet fra: <<https://www.forskningsradet.no/>>

Forsiden [Bilde]. Hentet fra: <https://storage.googleapis.com/snl-no-media/media/144223/standard_sosialdemokrati.jpg>

Ighodaro, Neo. A beginner's guide to the React component lifecycle. 27. mars 2018. Hentet fra: <<https://blog.pusher.com/beginners-guide-react-component-lifecycle/>>

Internet Engineering Task Force (IETF). The OAuth 2.0 Authorization Framework. Hentet fra: <<https://tools.ietf.org/html/rfc6749>>

Ken Schwaber, Ken og Sutherland, Jeff. The Scrum Guide. November 2017. Hentet fra: <<https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf>>

Konsekvenser [Bilde 5]. Hentet fra: <info.ecoonline.no>

Krawczyk, Tomasz. Introduction to claims-based authentication and authorization in .NET. 4. mars 2015. Hentet fra: <<https://kariera.future-processing.pl/blog/introduction-to-claims-based-authentication-and-authorization-in-net/>>

Kristoffersen, Bjørn. Databasesystemer. Universistetsforlaget. 4. Utgave.

Liew, Zell. Understanding And Using REST APIs. 17. januar 2018. Hentet fra: <<https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/>>

Material-UI. Checkbox. Hentet fra: <<https://material-ui.com/components/checkboxes/>>

McGinnis, Tyler. URL Parameters with React Router. 16. januar, 2018. Hentet fra: <<https://tylermcginnis.com/react-router-url-parameters/>>

Microsoft. Application Settings (appsettings.json). 3. Juli 2017. Hentet fra: <<https://docs.microsoft.com/en-us/iis-administration/configuration/appsettings.json>>

Microsoft. Creating Natively Compiled Stored Procedures. 16. mars 2017. Hentet fra: <<https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/creating-natively-compiled-stored-procedures?redirectedfrom=MSDN&view=sql-server-ver15>>

Microsoft. Editions and supported features of SQL Server 2019. 4. november 2019. Hentet fra: <<https://docs.microsoft.com/en-us/sql/sql-server/editions-and-components-of-sql-server-version-15?view=sql-server-ver15#-editions>>

Microsoft. Enable Cross-Origin Requests (CORS) in ASP.Net Core. 17. April 2020. Hentet fra: <<https://docs.microsoft.com/en-us/aspnet/core/security/cors?view=aspnetcore-3.1>>

Microsoft. EXECUTE AS Clause (Transact-SQL). 14. mars 2017. Hentet fra: <<https://docs.microsoft.com/en-us/sql/t-sql/statements/execute-as-clause-transact-sql?view=sql-server-ver15>>

Microsoft. Overview of ASP.Net Core Authentication. 3. Mars 2020. Hentet fra: <<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-3.1>>

Microsoft. SQL injection. 16. Mars 2017. Hentet fra: <<https://docs.microsoft.com/en-us/sql/relational-databases/security/sql-injection?view=sql-server-ver15>>

Murray, Nate. Fullstack React: 30 days of React. Hentet fra: <<https://www.newline.co/fullstack-react/30-days-of-react/>>

NGINX. Setting up JWT Authentication. Hentet fra: <<https://docs.nginx.com/nginx/admin-guide/security-controls/configuring-jwt-authentication/>>

Nohe, Patrick. The difference between Encryption, Hashing and Salting. 19. desember 2018. Hentet fra: <<https://www.thesslstore.com/blog/difference-encryption-hashing-salting/>>

Npm - Github. bs-custom-file-input. Hentet fra: <<https://www.npmjs.com/package/bs-custom-file-input>>

Peipman, Gunnar [Figur 35]. What is claims-based-authentication? 8. Juli 2013. Hentet fra: <<https://gunnarpeipman.com/what-is-claims-based-authentication/>>

Quick, James. How To Validate a Login Form With React and Formik. 9. april 2020. Hentet fra: <<https://www.digitalocean.com/community/tutorials/how-to-validate-a-login-form-with-react-and-formik>>

React Training. Become a React Expert. Hentet fra: <<https://reacttraining.com/>>

React Training. Quick Start. Hentet fra: <<https://reacttraining.com/react-router/web>>

React Training. withRouter. Hentet fra: <<https://reacttraining.com/react-router/web/api withRouter>>

React. A JavaScript Library for Building User Interfaces. Hentet fra: <<https://reactjs.org/>>

React. Handing Events. Hentet fra: <<https://reactjs.org/docs/handling-events.html>>

React. React Component. React. Hentet fra: <<https://reactjs.org/docs/react-component.html>>

React. State and Lifecycle. Hentet fra: <<https://reactjs.org/docs/state-and-lifecycle.html>>

ReactJS Examples. ReactJS component for easy pagination. 4. mars 2019. Hentet fra: <<https://reactjsexample.com/reactjs-component-for-easy-pagination/>>

Reichert, Cody. 7 HTTP methods every web developer should know and how to test them. 5. juni 2017. Hentet fra: <<https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them>>

Rippon, Carl. ASP.NET Core 3 and React. Packt Publishing. Hentet fra: <https://books.google.no/books/about/ASP_NET_Core_3_and_React.html>

Thachankary, Geo J. Using Dapper For Data Access In ASP.NET Core Applications. 11. november 2019. Hentet fra: <<https://www.c-sharpcorner.com/article/using-dapper-for-data-access-in-asp-net-core-applications/>>

Travery Media. Simple Frontend Pagination | React. 23. juni 2019. Hentet fra: <<https://www.youtube.com/watch?v=IYCa1F-OWmk>>

TutorialsTeacher [Figur 33]. ASP.NET MVC Architecture. Hentet fra: <<https://www.tutorialsteacher.com/mvc/mvc-architecture>>

Upmostly. React onChange Events (With Examples). 8. september 2019. Hentet fra: <<https://upmostly.com/tutorials/react-onchange-events-with-examples>>

Visual Paradigm. What is Planning Poker in Agile? Hentet fra: <<https://www.visual-paradigm.com/scrum/what-is-agile-planning-poker/>>

W3Schools. React Lifecycle. Hentet fra:
https://www.w3schools.com/react/react_lifecycle.asp

Walke, Prashant[Figur 36]. How to set up authentication in Wordpress. 21. mars 2019.
<https://walkeprashant.wordpress.com/2019/03/21/how-to-setup-authentication-in-wordpress-rest-api-2/>

Whatmore, Jason. Tutorial for JWT authentication with Asp.Net Core - with entity framework. 15. mai 2020. Hentet fra: <https://jasonwatmore.com/post/2019/10/11/aspnet-core-3-jwt-authentication-tutorial-with-example-api#user-service-cs>

Whatmore, Jason. Tutorial for JWT authentication with Asp.Net Core - simple version. 2. januar 2020. Hentet fra: <https://jasonwatmore.com/post/2019/10/14/aspnet-core-3-simple-api-for-authentication-registration-and-user-management#user-service-cs>

Wieruch, Robin. How to fetch data in React. 6. juli 2018. Hentet fra: <https://www.robinwieruch.de/react-fetching-data>

Wieruch, Robin. React Conditional Rendering. 16. januar, 2018. Hentet fra:
<https://www.robinwieruch.de/conditional-rendering-react>

Wålberg, Jon Arve. Kanban eller Scrum? 18. april 2018. Hentet fra:
<https://www.prosjektbloggen.no/kanban-eller-scrum>

6 Vedlegg

6.1 Terminologi

MVP	MVP er en versjon av et nytt produkt som tillater å samle mest mulig data med minst mulig innsats.
Modul	En modul kan bli sett på som et kurs, men inneholder en informasjonsside.
Kurs	Et kurs er en rekke med informasjon som er ment til å utdanne vedkommene som mottar informasjonen.
IDE	Et integrert utviklingsmiljø er et program som gir omfattende fasiliteter til dataprogrammerere for programvareutvikling. En IDE består vanligvis av minst en kildekode editor, bygg automatiseringsverktøy og en feilsøking.
API	Et application program interface-applikasjonsprogram grensesnitt(API) er et sett med rutiner, protokoller og verktøy for å bygge programvare. I utgangspunktet spesifiserer en API hvordan programvarekomponenter skal samhandle. I tillegg brukes APIer når du programmerer grafiske brukergrensesnitt (GUI) -komponenter.
Backlog	Er en samling av oppgaver som må gjøres men som ikke er igangsatt. Backloggen finnes i Jira.
MOOCs	Massive Online Open Course og er kurs som er rettet mot å ha ubegrenset antall deltagere.
CSS	Cascading Style Sheets (CSS, gjennomgående stilark) er et språk som brukes til å definere utseende på filer skrevet i HTML eller XML
HTML	HTML is the standard markup language for creating Web pages.
npm	er en pakkebehandler for JavaScript-programmeringsspråket.
JSX	er en syntaksutvidelse til JavaScript
DOM	Document Object Model er et plattform og språknøytralt grensesnitt som lar programmer og skript dynamisk få tilgang til og oppdatere innholdet, strukturen og stilens til et dokument.
URL	Uniform Resource Locator, og er en enkelt forklart en lenke / adresse til en nettside

Immutable	I objektorientert og funksjonell programmering er et uforanderlig objekt et objekt hvis tilstand ikke kan endres etter at det er opprettet. Dette i motsetning til et mutabelt objekt, som kan endres etter at det er opprettet
SSMS	SQL Server Management Studio (SSMS) er et integrert miljø for å administrere enhver SQL-infrastruktur. Bruk SSMS for å få tilgang til, konfigurere, administrere, administrere og utvikle alle komponentene i SQL Server, Azure SQL Database og SQL Data Warehouse.
MVC	MVC. Står for "Model-View-Controller." MVC er en applikasjonsdesign modell som består av tre sammenkoblede deler. ... MVC-modellen eller "mønsteret" brukes ofte for å utvikle moderne brukergrensesnitt. Det er de grunnleggende brikkene for å designe programmer for desktop eller mobil, så vel som webapplikasjoner.
Hard-coding	(data eller parametere) i et program på en slik måte at de ikke kan endres uten å endre programmet.
HTTP-metode	HTTP definerer et sett med forespørrelse metoder for å indikere ønsket handling som skal utføres for en gitt ressurs. POST, GET, PUT, PATCH og DELETE. Disse tilsvarer henholdsvis opprette, lese, oppdatere og slette operasjoner.
SSL	SSL (Secure Sockets Layer) er en sikkerhetsteknologi som ofte brukes til å sikre server til nettleser transaksjoner. Dette inkluderer generelt sikring av all informasjon som sendes av en nettleser (for eksempel en kundes kredittkortnummer eller passord) til en webserver (for eksempel en nettbutikk eller nettbank applikasjon).????
CORS	Cross-Origin Resource Sharing (CORS) er en mekanisme som bruker flere HTTP-overskrifter for å fortelle nettlesere om å gi et webapplikasjon som kjører med ett opphav, tilgang til utvalgte ressurser fra en annen opprinnelse.
Wireframe	Også kjent som en sideskjema eller skermtegning, er en visuell guide som representerer skjelettrammen til et nettsted.
ORM	Object - relational mapping i informatikk er en programmeringsteknikk for å konvertere data mellom systemer som er inkompatible med bruk av objektorienterte programmeringsspråk. Dette skaper faktisk en "virtual object database" som kan brukes innen programmeringsspråket.
Wideband Delphi	Estimeringsmetode er en konsensusbasert teknikk for å estimere innsats.

6.2 Nåværende WIX webside

<https://blogg.hioa.no/wixproject/>

Work inclusion of migrants [Home](#) [Background ▾](#) [Events ▾](#) [Tools ▾](#) [Images ▾](#) [Video ▾](#) [Contacts](#) [Sitemap](#) [Internal ▾](#)

Work inclusion of migrants

WIX Project

[Home](#)

Home

The main output of the WIX project will be three online courses (MOOCs) on **work inclusion of migrants in comparative urban contexts**. The first course, **Understanding and measuring poverty in comparative urban settings**, will be launched on a trial basis in 2019.

Garment workers Bangladesh. Source: Bangladesh. Gazipur BIGUF 2015. Solidarity Center

The courses will compare the cities of Oslo, Bangalore and Moscow. They will be jointly offered to Master level students at Oslo Metropolitan University, at National Law School of India University, Bangalore, and at Higher School of Economics, Moscow.

The full set will be available in a blended mode (integrated into a regular course) from 2020. The courses, which are based on the edX platform, will be published as part of the IN:Code program at OsloMet.

During the implementation of the project, joint research activities between OsloMet, NLSIU and HSE will be initiated, the main target being planned proposals targeted to EU and Norwegian research calls, focused on the integration of migrants within urban settings.

Background information about the project.

Abbreviations

- WIX: Work Inclusion edX course
- HSE: Higher School of Economics
- NLSIU: National Law School University of India
- OsloMet: Oslo Metropolitan University

Web site

This web site was set up in May 2018 to share information and resources related to the project.

Comments are welcome. Standard netiquette applies. Comments with more than one link will be stopped by the spam filter.

Recent Comments

- Liam on Review of elearning

©2020 - Work inclusion of migrants

- Weaver Xtreme Theme

Work Inclusion of Migrants

Developed and led by experts in the fields of social policy, social work, and urban studies, this course provides an introduction to work and social inclusion of migrants in the comparative urban contexts of Bangalore, Moscow and Oslo.

In the course, we will use a visual and interactive approach to explore theories and methods for evaluating work inclusion measures, as well as the crucial strategies and factors that are often not considered. You will gain knowledge of how poverty may be understood and compared across the three settings and will learn alternative conceptual approaches to the more traditional income-focused one.

You will learn new ways to evaluate the work inclusion strategies beyond those currently used. Our focus is on moving beyond an income-only focus in order to improve the social inclusion of target groups through work. Using a “situated” work inclusion approach, the course will provide an appreciation of the historical, social-material and psychological factors shaping the success of inclusion methods

We will focus on:

- methods and theories for understanding and measuring poverty in comparative settings
- policy strategies and necessary considerations developing inclusion measures
- evaluation of current work inclusion measures using alternative

What you'll learn

On completion of this course, you will be able to:

1. Understand and describe trends affecting labor markets, welfare states and systems within Norway, India and Russia, as these impact on anti-poverty policy and social and work exclusion of migrants.
2. Describe the current strategies in place for work and social inclusion within and outside the EU, applying various comparative methodologies for evaluation across and within nation-states.

Work Inclusion of Migrants

Courses Checkout Courses Profile Become a Teacher

Courses

aadneasland 18. November 2019

[Home](#) / Courses

 Search

[Qualitative
methods to
study integration](#)

Free

[enkipub
Understanding
nd Measuring
r poverty](#)

[Enroll](#)

Free

[Enroll](#)

<https://blogg.hioa.no/wixproject/>

6.3 Kravspesifikasjon

Kravspesifikasjon

Kravspesifikasjonene ble utarbeidet under samarbeid og samtaler mellom bachelorgruppen og oppdragsgiver. Hensikten med kravspesifikasjon dokumentet er å ha et felles grunnlag for kunden og bachelorgruppen om hva applikasjonen skal inneholde. Kravspesifikasjonene ble utformet under forberedelsesfasen og var et forsøk på å prioritere applikasjonens viktigste funksjoner.

Målet med applikasjonen er å...

Ikke-funksjonelle krav

- Applikasjonen skal være brukervennlig og intuitiv å bruke
- Applikasjonen skal kunne fungere i flere land
- Administrasjons delen skal ligne brukerdelen i utseende.
- En "About" side som inneholder informasjon om.
- Logisk navigasjon.

Funksjonelle krav

Administrasjon

- Pålogging for administrasjon brukere ved mail og sosiale medier.
- funksjonalitet til å legge til flere administratorer
- hver administrator kan utnevne nye administratorer
- Opprettelse og redigering av kurs.
- Mulighet for å velge mellom tekst, video, PDF, PowerPoint, quiz og kommentarfelt.
- Mulighet for å lagre kurs uten og publisere, for videre arbeid senere.
- Mulighet for å endre kurs etter opprettelse.
- Mulighet for å endre kurs etter publisering
- Det kan åpnes i alle nettlesere

Brukere

- Kan gjennomføre kurs uten å logge inn.
- Bruker kan logge inn og følge prosesjon av kurs.
- Brukeren kan se prosesjon bakover og fremover i tid.
- Brukeren har mulighet til å diskutere i kommentarfeltet med andre innloggede brukere.

Tekniske krav

- Webapplikasjon, hovedbruk gjennom nettleser.
- Hosting ubegrenset av tid, ikke bare for en fastsatt tid.

- Dokumentasjonskrav for videreføring av prosjektet.

6.4 Dagbok

Detailed report

2020-01-01 - 2020-12-31

Total 265 h 20 min

Date	Description	Duration	User
02-15	Learning Bootstrap	0:32:00	Alina Sergan
	WIX	18:28-19:00	
02-15	The complete guide to building an app	2:11:51	Alina Sergan
	WIX	19:38-21:50	
02-15	The complete guide to building an app	0:18:15	Alina Sergan
	WIX	21:50-22:08	
02-17	Learning React	6:52:00	Alina Sergan
	WIX	08:48-15:40	
03-03	(FrontEnd) About Page	2:51:00	Alina Sergan
	WIX	13:29-16:20	
03-04	(FrontEnd) Login page	6:41:32	Alina Sergan
	WIX	09:34-16:16	
03-05	(FrontEnd) Modul Main Page	4:25:00	Alina Sergan
	WIX	16:35-21:00	
03-06	(FrontEnd) Modul Detail Page Text	3:10:00	Alina Sergan
	WIX	16:50-20:00	
03-07	(FrontEnd) Modul Detail Page Video	3:00:00	Alina Sergan
	WIX	11:00-14:00	
03-08	(FrontEnd) Modul Main Page	5:25:00	Alina Sergan
	WIX	16:35-22:00	
03-09	(FrontEnd) Modul Detail Page Complited	4:45:00	Alina Sergan
	WIX	14:35-19:20	
03-10	(FrontEnd) Admin Home Page	4:50:00	Alina Sergan
	WIX	15:10-20:00	
03-11	(FrontEnd) Modul Detail Page Quiz	4:35:00	Alina Sergan
	WIX	13:25-18:00	
03-12	Beskrevet MVP (faser)	1:10:00	Alina Sergan
	WIX	16:50-18:00	
03-13	KlasseDiagram; high hierarchy for program architecture	3:30:00	Alina Sergan
	WIX	12:40-16:10	
03-16	(FrontEnd) Admin Modul Main Page	4:00:00	Alina Sergan
	WIX	14:31-18:31	
03-17	(FrontEnd) Admin Modul Detail Page Text	3:12:00	Alina Sergan
	WIX	16:48-20:00	
03-18	(FrontEnd) Admin Modul Main Page	4:30:00	Alina Sergan
	WIX	13:10-17:40	

03-19	(FrontEnd) Admin Modul Detail Page Video	5:09:00	Alina Sergan
	WIX	12:31-17:40	
03-21	(FrontEnd) Pop-up windows for AdminMainModulPage	4:39:00	Alina Sergan
	WIX	09:31-14:10	
03-22	(Report) Utviklingsverktøy	3:29:00	Alina Sergan
	WIX	10:31-14:00	
03-23	(Report) Planleggingsfaser	2:50:00	Alina Sergan
	WIX	16:10-19:00	
03-24	(Planleggin) Work with Jira, Fremdriftsplan	3:00:00	Alina Sergan
	WIX	18:01-21:01	
03-25	(FrontEnd) Create Trello board for bette work with FrontEnd Tasks	2:16:00	Alina Sergan
	WIX	10:24-12:40	
03-26	(FrontEnd) work with Github	6:05:00	Alina Sergan
	WIX	10:23-16:28	
03-27	(Report) Go gjennom alle deller.	3:00:00	Alina Sergan
	WIX	18:27-21:27	
03-28	(FrontEnd) Breadcrumbs	1:32:00	Alina Sergan
	WIX	10:28-12:00	
03-29	(FrontEnd) Fixing problem with dependencies	5:27:00	Alina Sergan
	WIX	10:33-16:00	
04-01	(FrontEnd) Work with Route	5:02:00	Alina Sergan
	WIX	10:30-15:32	
04-03	(FrontEnd) Logge inn på facebook	4:31:00	Alina Sergan
	WIX	15:29-20:00	
04-05	(FrontEnd) Fixing problem with commit to GitHub	3:34:00	Alina Sergan
	WIX	16:36-20:10	
04-06	(Opplæring) React course	5:00:00	Alina Sergan
	WIX	10:00-15:00	
04-07	(Opplæring) React course	3:00:00	Alina Sergan
	WIX	13:00-16:00	
04-08	(Opplæring) React course	2:00:00	Alina Sergan
	WIX	16:30-18:30	
04-09	(FrontEnd) Added Adminpanel Modul Page to the workflow	3:43:00	Alina Sergan
	WIX	17:17-21:00	
04-10	(Opplæring) React course	2:01:00	Alina Sergan
	WIX	11:00-13:01	
04-11	(Opplæring) React course	4:10:00	Alina Sergan
	WIX	15:50-20:00	
04-12	(Opplæring) React course	2:10:00	Alina Sergan
	WIX	12:50-15:00	
04-13	(Opplæring) React course	2:50:00	Alina Sergan
	WIX	18:10-21:00	

04-15	(Opplæring) React course	4:30:00	Alina Sergan
	WIX	17:30-22:00	
04-16	(FrontEnd) Brukte API fra Visual Studio til å knytte med FrontEnd delen	5:43:00	Alina Sergan
	WIX	16:17-22:00	
04-17	(FrontEnd) Endret Login Side	6:00:00	Alina Sergan
	WIX	14:00-20:00	
04-18	(Opplæring) React course	3:20:00	Alina Sergan
	WIX	12:40-16:00	
04-19	(FrontEnd) Added Info Icon on the Quiz Page. Fixed Login Page	5:00:00	Alina Sergan
	WIX	10:00-15:00	
04-21	(FrontEnd) Admin AdminModulVideo, router, setting data about course in json	3:45:00	Alina Sergan
	WIX	19:15-23:00	
04-22	(Opplæring, Frontend) learn how to add API, rewrite cards from static to getting data from json	3:49:00	Alina Sergan
	WIX	17:41-21:30	
04-23	Work with data for coursCard, TextPage and QuizPage	3:39:13	Alina Sergan
	WIX	17:48-21:28	
04-24	(Report) Beskrevet komponenter, package.json for FrontEnd delen	1:40:00	Alina Sergan
	WIX	19:50-21:30	
04-25	(FrontEnd) Routers	1:21:41	Alina Sergan
	WIX	15:58-17:19	
04-25	(no description)	1:34:17	Alina Sergan
	WIX	17:47-19:21	
04-25	(FrontEnd) Data for lessons in order to make right Router	1:39:54	Alina Sergan
	WIX	20:24-22:04	
04-26	(FrontEnd) Multiple Conditional Renderings	2:42:49	Alina Sergan
	WIX	13:28-16:11	
04-26	(FrontEnd) Switch statement inside a react component; Quiz Page Admin	2:50:23	Alina Sergan
	WIX	17:55-20:46	
04-27	(FrontEnd) Switch statement	2:01:54	Alina Sergan
	WIX	20:14-22:16	
04-28	Meeting	1:32:00	Alina Sergan
	WIX	11:00-12:32	
05-01	(FrontEnd) Pagination, AdminQuiz	1:56:00	Alina Sergan
	WIX	19:10-21:06	
05-01	(FrontEnd) Pagination, AdminQuiz	0:49:52	Alina Sergan
	WIX	21:21-22:11	
05-02	(FrontEnd) Add additional element on the screen with plus button (answer)	3:19:32	Alina Sergan
	WIX	16:46-20:05	
05-02	(Report) FrontEnd delen segregere i deler	1:34:30	Alina Sergan
	WIX	21:21-22:56	
05-03	(FrontEnd) Add one more question on the page	3:03:27	Alina Sergan
	WIX	14:41-17:44	

05-03	(Report) Description of the Admin Quiz Page	0:40:38	Alina Sergan
	WIX	19:48-20:29	
05-03	(Report) Description of the Admin Quiz Page	0:57:07	Alina Sergan
	WIX	21:03-22:01	
05-04	(FrontEnd) Add one more question on the page	2:24:00	Alina Sergan
	WIX	19:06-21:30	
05-05	(FrontEnd) endre design av Admin quiz side	1:00:00	Alina Sergan
	WIX	20:00-21:00	
05-06	(FrontEnd) Forbedre Admin Video Side. input file	1:56:46	Alina Sergan
	WIX	13:32-15:29	
05-06	(Report) Description of the Admin Video Page	1:11:23	Alina Sergan
	WIX	19:05-20:16	
05-06	(Report) Description of the Home Page	1:39:32	Alina Sergan
	WIX	20:20-22:00	
05-07	(Jira) oppdatering jira board	0:09:47	Alina Sergan
	WIX	12:47-12:57	
05-07	(Report) Description of the Modul Main Page, Test Page	2:19:58	Alina Sergan
	WIX	13:27-15:47	
05-07	(Report) Description of the Video Page	1:01:26	Alina Sergan
	WIX	15:55-16:56	
05-07	(Report) Description of the Video Page, utviklingsverktøy, fixse bugs for video og text side	1:29:06	Alina Sergan
	WIX	18:12-19:42	
05-10	(Report) Description of the user QuizPage	1:12:39	Alina Sergan
	WIX	14:39-15:51	
05-10	(FrontEnd) API	2:11:51	Alina Sergan
	WIX	16:15-18:26	
05-10	(Report) Login Page	0:12:10	Alina Sergan
	WIX	20:12-20:24	
05-10	(Report) Login Page	0:24:24	Alina Sergan
	WIX	20:37-21:01	
05-10	(Report) Login Page	0:39:58	Alina Sergan
	WIX	21:15-21:55	
05-11	Frontend og Backend samspill	3:00:00	Alina Sergan
	WIX	20:05-23:05	
05-13	(FrontEnd) API auth	2:17:42	Alina Sergan
	WIX	13:21-15:39	
05-14	(Report) Samspill mellom FrontEnd og BackEnd, Test Book	7:43:07	Alina Sergan
	WIX	10:15-17:58	
05-14	(Report) Samspill mellom FrontEnd og BackEnd, Test Book, AdminPanel	1:44:08	Alina Sergan
	WIX	18:43-20:27	
05-15	(FrontEnd) Delete, Update API	3:15:15	Alina Sergan
	WIX	12:05-15:21	

05-15	(FrontEnd) Delete, Update API	1:19:30	Alina Sergan
	WIX	16:35-17:54	
05-15	(FrontEnd) Update API	2:00:00	Alina Sergan
	WIX	20:24-22:24	
05-16	(Report) Oppstart med React/ struktur av filer	2:22:21	Alina Sergan
	WIX	12:58-15:21	
05-16	(Report) Modul lekse side med tekst	0:55:43	Alina Sergan
	WIX	17:18-18:13	
05-16	(Report) Modul lekse side med tekst	0:36:06	Alina Sergan
	WIX	18:58-19:34	
05-16	Testbok	0:57:00	Alina Sergan
	(no project)	21:03-22:00	
05-17	Jira	0:15:46	Alina Sergan
	WIX	21:07-21:23	
05-17	(Report) Risikovurdering	0:22:23	Alina Sergan
	(no project)	22:05-22:28	
05-18	(FrontEnd) Routers	3:00:00	Alina Sergan
	WIX	12:47-15:47	
05-18	(Report) avslutning av frontend, lucidchart	2:54:01	Alina Sergan
	WIX	20:24-23:18	
05-19	(Report) Risikovurdering, fremtdriftspan, dagbok	3:31:30	Alina Sergan
	(no project)	19:26-22:57	
05-20	(Report) Arbeid sammen med gruppe	2:40:00	Alina Sergan
	WIX	11:20-14:00	
05-20	(Report) Arbeid sammen med gruppe	3:36:00	Alina Sergan
	WIX	15:12-18:48	

Detailed report



2020-01-01 - 2020-12-31

Total 133 h 54 min

Date	Description	Duration	User
03-04	Logg inn WIX	1:09:46 14:59-16:09	Alexredfield13
03-05	Lesing til API fra Udemy kurset WIX	5:15:00 11:20-16:35	Alexredfield13
03-08	Mer om Logg Inn WIX	1:21:42 18:03-19:25	Alexredfield13
03-09	Jobbe med å få section 9 til å funke- Logg inn WIX	2:01:06 11:45-13:46	Alexredfield13
03-10	Reading about API from Udemy course WIX	4:00:00 11:46-15:46	Alexredfield13
03-11	Working on creating API WIX	1:49:15 11:31-13:20	Alexredfield13
03-12	Reading udemy course and learning about database WIX	3:00:00 10:55-13:55	Alexredfield13
03-13	Reading udemy course and learning about database(ii) WIX	3:09:00 12:23-15:32	Alexredfield13
03-14	Reading about database and figuring out the platform where the diagram will be written on WIX	4:30:00 11:17-15:47	Alexredfield13
03-15	Reading about how to create database and implement in API WIX	4:51:00 11:49-16:40	Alexredfield13
03-17	Creating API with entity framework WIX	2:35:00 13:53-16:28	Alexredfield13
03-18	Reading about database WIX	5:00:00 11:31-16:31	Alexredfield13
03-19	Reading on internet about database and attempting on creating it WIX	4:48:00 11:35-16:23	Alexredfield13
03-21	Attempting on creating database and reading how to do it WIX	4:20:00 23:16-03:36	Alexredfield13
03-24	Creating database WIX	1:20:54 13:57-15:18	Alexredfield13
04-09	Writing to rapport about database WIX	0:44:39 15:58-16:42	Alexredfield13
04-10	Checking the rapport for any mistakes WIX	2:30:00 13:22-15:52	Alexredfield13
04-13	Creating database diagram, reading about database diagrams (i) WIX	3:00:00 10:43-13:43	Alexredfield13

04-14	Reading about tables, subtables and the connections between them online and on older notes	4:00:00	Alexredfield13
	WIX	12:20-16:20	
04-15	Adding more tables and reading more about database	5:00:00	Alexredfield13
	WIX	12:10-17:10	
04-17	Adding more tables and reading more about database (ii)	3:18:00	Alexredfield13
	WIX	14:59-18:17	
04-18	Creating database diagram (ii)	0:31:20	Alexredfield13
	WIX	22:32-23:03	
04-19	Checking old notes from previous database lectures to help me understand more about database	4:47:00	Alexredfield13
	WIX	11:33-16:20	
04-20	Database diagram (iii) Checking	1:36:41	Alexredfield13
	WIX	16:34-18:11	
04-21	Reading about database diagrams	3:06:17	Alexredfield13
	WIX	17:01-20:07	
04-22	Database Diagram implementation to the Backend	2:33:16	Alexredfield13
	WIX	14:49-17:22	
04-23	Reading online and checking our notes about backend from our slutrapport to try to implement the database diagram into the backend	3:30:00	Alexredfield13
	WIX	10:30-14:00	
04-25	Database Diagram (iv) Adding additional tables	2:37:00	Alexredfield13
	WIX	17:43-20:20	
04-26	Summary of database diagram and final touches for its completion	3:50:00	Alexredfield13
	WIX	11:11-15:01	
04-27	Checking database diagram and correcting	2:53:00	Alexredfield13
	WIX	17:33-20:26	
04-28	Meeting with Group	0:53:00	Alexredfield13
	WIX	11:37-12:30	
04-28	Add attribute navn på Database diagram og sjekk om alt er riktig	4:44:00	Alexredfield13
	WIX	14:01-18:45	
04-30	Prøver å implementere diagrammet på SQL SMS- Turning the diagram to English	5:47:54	Alexredfield13
	WIX	13:03-18:51	
05-01	Implementing database from SQL SMS to Visual Studio	5:23:50	Alexredfield13
	WIX	14:18-19:41	
05-02	Creating methods that will be used to store data in database- Creating SQL Queries too	4:25:51	Alexredfield13
	WIX	14:50-19:16	
05-04	Meeting and finalizing database procedures and testing if they are working	4:48:20	Alexredfield13
	WIX	15:38-20:26	
05-13	Skrive på rapport	3:40:00	Alexredfield13
	(no project)	13:20-17:00	
05-19	Rapportering	3:44:00	Alexredfield13
	(no project)	14:27-18:11	
05-20	Rapportering (Checking, Finalizing rapport)	7:19:42	Alexredfield13
	(no project)	11:29-18:48	

Detailed report



2020-01-01 - 2020-12-31

Total 121 h 25 min

Date	Description	Duration	User
03-03	s	0:00:54	Amiinas2210
	WIX	13:36-13:37	
03-03	WIX	0:00:26	Amiinas2210
	WIX	13:37-13:38	
03-03	Jobber med Databasen for admin og manager for logg inn siden	2:07:21	Amiinas2210
	WIX	13:38-15:45	
03-09	CRUD	3:19:12	Amiinas2210
	WIX	08:52-12:11	
03-09	Jobber med hvordan legge til kurs	2:01:43	Amiinas2210
	WIX	18:44-20:46	
03-12	Kurs for CRUD	6:53:00	Amiinas2210
	WIX	15:23-22:16	
03-13	Fase 1: Legg til (CRUD) med react, .net core	4:23:00	Amiinas2210
	WIX	14:49-19:12	
03-16	Database for crud	4:01:59	Amiinas2210
	WIX	10:49-14:51	
03-16	Database	3:38:55	Amiinas2210
	WIX	16:24-20:03	
03-17	Controller for CRUD	6:57:09	Amiinas2210
	WIX	12:25-19:22	
03-18	Handler og Query for Kurs	4:04:00	Amiinas2210
	WIX	13:47-17:51	
03-19	CREATE, Delete med .netCore uten React	3:23:00	Amiinas2210
	WIX	15:54-19:17	
03-22	React for CRUD	3:23:00	Amiinas2210
	WIX	11:45-15:08	
03-25	Funksjonalitet til knappene	4:52:08	Amiinas2210
	WIX	15:21-20:13	
03-26	hooks	4:06:00	Amiinas2210
	WIX	19:13-23:19	
03-27	view redigering funksjonalitet	6:14:58	Amiinas2210
	WIX	10:06-16:21	
03-28	create cancel og edit	4:38:00	Amiinas2210
	WIX	16:05-20:43	
03-30	Form	1:21:13	Amiinas2210
	WIX	18:54-20:15	
03-31	Submisjon og components	1:30:44	Amiinas2210
	WIX	18:55-20:26	

04-06	Fortsettelse på submission	1:29:23	Amiinas2210
	WIX	13:55-15:24	
04-14	Image from database	3:03:42	Amiinas2210
	WIX	13:58-17:02	
04-15	Jobber med å få Image fra database	1:13:58	Amiinas2210
	WIX	19:14-20:28	
04-19	Lest om dapper	2:00:00	Amiinas2210
	WIX	17:04-19:04	
04-21	Satt om sql server lokalt på min pc	1:00:00	Amiinas2210
	WIX	12:00-13:00	
04-23	Skrevt notater fra CRUD for rapporten	2:00:00	Amiinas2210
	WIX	14:00-16:00	
04-28	Møte	1:40:00	Amiinas2210
	WIX	11:00-12:40	
04-28	Prøver å fikse error med sql	2:58:00	Amiinas2210
	WIX	13:38-16:36	
04-29	Fikset problemer med sql	2:00:00	Amiinas2210
	WIX	19:09-21:09	
05-01	Sql database oppdatering	2:38:16	Amiinas2210
	WIX	16:05-18:43	
05-04	Møte	0:52:37	Amiinas2210
	WIX	15:01-15:53	
05-04	Sql spøringer	1:44:05	Amiinas2210
	(no project)	18:27-20:11	
05-09	Jobber med rapporten på Backend del	3:34:22	Amiinas2210
	WIX	12:05-15:39	
05-10	Rapport	3:02:00	Amiinas2210
	WIX	13:06-16:08	
05-11	Rapport	3:00:00	Amiinas2210
	WIX	13:00-16:00	
05-12	Rapport Database	3:00:00	Amiinas2210
	WIX	15:00-18:00	
05-13	Møte backend	2:35:58	Amiinas2210
	WIX	17:59-20:35	
05-17	Rapport skriving	3:00:14	Amiinas2210
	WIX	14:04-17:04	
05-18	Jobber med rapporten på Backend del	4:52:00	Amiinas2210
	WIX	16:04-20:56	
05-19	Rapport Backend	3:00:16	Amiinas2210
	WIX	16:33-19:33	
05-20	Rapport skriving	1:59:25	Amiinas2210
	WIX	12:00-13:59	
05-20	Rapport skriving	3:44:07	Amiinas2210
	WIX	15:04-18:48	

Detailed report



2020-01-01 - 2020-12-31

Total 82 h 00 min

Date	Description	Duration	User
04-01	rapportskriving WIX	5:00:00 08:00-13:00	Pederstadelias
04-02	rapportskriving WIX	5:00:00 08:00-13:00	Pederstadelias
04-03	rapportskriving WIX	5:00:00 08:00-13:00	Pederstadelias
04-06	rapportskriving WIX	5:00:00 08:00-13:00	Pederstadelias
04-08	rapportskriving WIX	5:00:00 08:00-13:00	Pederstadelias
04-13	rapportskriving WIX	5:00:00 08:00-13:00	Pederstadelias
04-14	rapportskriving WIX	5:00:00 08:00-13:00	Pederstadelias
04-15	rapportskriving WIX	5:00:00 08:00-13:00	Pederstadelias
04-20	rapportskriving WIX	5:00:00 08:00-13:00	Pederstadelias
04-21	rapportskriving WIX	5:00:00 08:00-13:00	Pederstadelias
04-22	rapportskriving WIX	5:00:00 08:00-13:00	Pederstadelias
04-27	rapportskriving WIX	5:00:00 08:00-13:00	Pederstadelias
04-28	rapportskriving WIX	5:00:00 08:00-13:00	Pederstadelias
04-29	rapportskriving WIX	5:00:00 08:00-13:00	Pederstadelias
05-04	rapportskriving WIX	4:00:00 08:00-12:00	Pederstadelias
05-05	rapportskriving WIX	4:00:00 08:00-12:00	Pederstadelias
05-06	rapportskriving WIX	4:00:00 08:00-12:00	Pederstadelias

Created with toggl.com

Detailed report



2020-01-01 - 2020-12-31

Total 183 h 45 min

Date	Description	Duration	User
03-03	log in functionality with facebook (Udemy)	2:35:00	Siljebjork
	WIX	13:40-16:15	
03-03	login functionality with gmail, facebook, etc (documentation ASP.NET)	0:26:07	Siljebjork
	WIX	19:40-20:06	
03-04	login functionality with gmail, facebook, etc (documentation ASP.NET)	0:18:31	Siljebjork
	WIX	18:37-18:56	
03-04	log in functionality with facebook	0:43:00	Siljebjork
	WIX	20:47-21:30	
03-08	log in functionality with facebook - following udemy video and making notes of the steps	1:30:42	Siljebjork
	WIX	10:30-12:01	
03-08	log in functionality with facebook - following udemy video and making notes of the steps	0:28:39	Siljebjork
	WIX	15:10-15:39	
03-09	log in functionality with facebook - following udemy video and taking notes of the steps + coding	3:19:18	Siljebjork
	WIX	10:42-14:01	
03-10	Fixing issues with frontend, to be able to open the website.	2:00:00	Siljebjork
	WIX	11:30-13:30	
03-12	Adding new form for registering with e-mail and password + some alteration in design (LoginPage)	3:28:41	Siljebjork
	WIX	10:57-14:25	
03-15	Added Facebook login button with some basic functionality	2:10:15	Siljebjork
	WIX	12:55-15:05	
03-15	Trying to figure out a way to log out using react	0:38:26	Siljebjork
	WIX	15:05-15:44	
03-15	Trying with React Social Login, then trying to insert authentication with Gmail	1:14:04	Siljebjork
	WIX	16:58-18:12	
03-15	Inserted gmail authentication.	1:23:33	Siljebjork
	WIX	18:50-20:14	
03-16	Fixed some issues with fb and google login; managed nice button for google	2:28:43	Siljebjork
	WIX	10:50-13:18	
03-17	Preparing some documentation (theory)	2:40:00	Siljebjork
	WIX	10:40-13:20	
03-18	Writing documentation for authentication with fb and google using react (frontend)/ following steps on microsoft documentation to build authentication on backend	4:12:53	Siljebjork
	WIX	09:19-13:32	

03-19	Analyzing how to Fetch API Data with React JS + documentation	1:48:24	Siljebjork
	WIX	11:21-13:09	
03-31	Documentation for oAuth	2:01:00	Siljebjork
	WIX	09:44-11:45	
04-02	Added ClientApp folder to backend and transferred content of frontend to it	2:16:30	Siljebjork
	WIX	16:34-18:50	
04-06	Trying to fix problem	4:00:00	Siljebjork
	WIX	10:30-14:30	
04-10	Implementing database for Admin and User; publishing front end log in in master branch; reading instructions from book	6:33:21	Siljebjork
	WIX	11:11-17:44	
04-11	Stored procedure in SQL server for admin and user databases	2:13:25	Siljebjork
	WIX	10:00-12:14	
04-11	Documentation for User databases + connection with ASP.NET core in Visual Studio	1:38:47	Siljebjork
	WIX	13:24-15:03	
04-13	Working with databases for users, courses, fetching users and fetching courses. Documentation. Meeting.	8:02:06	Siljebjork
	WIX	12:32-20:34	
04-14	(no description)	3:05:00	Siljebjork
	WIX	14:25-17:30	
04-16	further work on api	1:57:26	Siljebjork
	WIX	00:10-02:07	
04-16	adding new databases + implementing restful api + documentation	7:02:01	Siljebjork
	WIX	13:09-20:12	
04-18	Fetching user by id and fetching course by id (sql server and visual studio)	5:36:40	Siljebjork
	WIX	10:03-15:39	
04-19	Create User and Create Course - coding i Visual Studios, testing i Postman (OK!), documentation	3:36:00	Siljebjork
	WIX	13:51-17:27	
04-19	Delete course/user - Visual Studios, testing in Postman (OK!), documentation. Trying to fix issue with database FetchingUserID.	1:30:11	Siljebjork
	WIX	18:04-19:34	
04-20	Update User/course - coding i Visual Studios, testing i Postman (OK!), documentation	2:02:43	Siljebjork
	WIX	12:03-14:06	
04-20	Meeting	1:00:00	Siljebjork
	WIX	17:10-18:10	
04-20	Fixing database issue so it can be available for everyone	1:16:00	Siljebjork
	WIX	18:27-19:43	
04-21	Dokumentasjon. Omskriving til norsk av hele rapporten om databaser med dapper. Restrukturering.	2:04:17	Siljebjork
	WIX	13:40-15:44	
04-21	Dokumentasjon. Omskriving til norsk av hele rapporten om databaser med dapper. Restrukturering.	0:30:32	Siljebjork
	WIX	17:30-18:00	

04-22	Dokumentasjon. Omskriving til norsk av hele rapporten om databaser med dapper. Restrukturering.	2:43:05	Siljebjork
	WIX	13:45-16:28	
04-23	Working with authentication with Azure AD (portal + Visual studios) + documentation	2:08:59	Siljebjork
	WIX	13:59-16:08	
04-23	Working with authentication with Azure AD (portal + Visual studios) + documentation	1:37:23	Siljebjork
	WIX	17:40-19:18	
04-23	Testing and trying to fix URL redirection issue - authentication with Azure AD	0:30:12	Siljebjork
	WIX	19:40-20:10	
04-24	Trying and failing completely to fix URL redirection issue. Now getting HTTPS issue too.	1:56:00	Siljebjork
	WIX	13:29-15:25	
04-28	Møte	1:32:00	Siljebjork
	WIX	11:00-12:32	
04-29	Following "ASP.NET Core 3.1 - JWT Authentication" Tutorial for login for Admin	0:59:00	Siljebjork
	WIX	09:01-10:00	
04-30	I Visual Studio: forandret alt som står "Course" til "Modules". La til "Lesson". Jobbing med controllers, models, data repository og interface.	2:12:04	Siljebjork
	WIX	06:30-08:43	
04-30	Updating database i visual studio. Trying to figure out how to establish an URL to connect backend to frontend.	2:34:42	Siljebjork
	WIX	16:22-18:57	
05-01	Reading about creating an API gateway/publishing API azure	1:01:52	Siljebjork
	WIX	12:16-13:18	
05-01	Jobbing med database + møte med backend team	3:02:12	Siljebjork
	WIX	15:42-18:44	
05-03	Fixing code for databases in Visual studio + publishing of API in azure	2:16:34	Siljebjork
	WIX	16:38-18:55	
05-03	Fixing code for databases in Visual studio, updating stored procedures in sql server.	0:48:53	Siljebjork
	WIX	19:44-20:33	
05-03	Updating stored procedures in sql server for Admin, User, Teacher and Lesson. (Create, Update, Delete, Fetch and Fetch by ID)	1:49:32	Siljebjork
	WIX	22:37-00:26	
05-04	In Visual studio: updating controllers and repositories for Modules and Person. Adding controllers and repositories for Lessons, teacher and admin.	4:49:41	Siljebjork
	WIX	08:18-13:08	
05-04	Møte frontend + backend	0:53:55	Siljebjork
	WIX	14:59-15:53	
05-04	Møte med backend + oppdatering av database i SQL server og jobbing med API	2:32:33	Siljebjork
	WIX	18:02-20:35	
05-06	Looking at login with Auth0 + some implementations in Visual Studio	1:42:15	Siljebjork
	(no project)	15:45-17:27	
05-07	Work with login	2:08:29	Siljebjork
	WIX	10:19-12:28	

05-07	Work with login	2:50:45	Siljebjork
	WIX	13:15-16:06	
05-07	Work with login	3:08:26	Siljebjork
	WIX	16:51-19:59	
05-09	Added frontend files to project for testing purposes, small modifications on backend side	1:03:45	Siljebjork
	WIX	21:56-23:00	
05-10	Login functionality for admin - visual studio	6:48:36	Siljebjork
	WIX	15:56-22:45	
05-11	Møte frontend + backend	1:49:43	Siljebjork
	WIX	20:30-22:20	
05-12	Authentication and authorization - Visual studio and testing in postman.	10:31:08	Siljebjork
	WIX	09:10-19:41	
05-13	Working on documentation	2:14:24	Siljebjork
	WIX	10:43-12:57	
05-13	Working on documentation and meetings	3:58:00	Siljebjork
	WIX	16:33-20:31	
05-14	Documentation - authorization/authentication	0:31:21	Siljebjork
	WIX	09:27-09:59	
05-14	Documentation - authorization/authentication	2:42:23	Siljebjork
	WIX	14:52-17:34	
05-15	New fix in databases + documentation	4:15:00	Siljebjork
	WIX	10:15-14:30	
05-16	Created Roles - created models for all controllers	2:15:23	Siljebjork
	WIX	18:10-20:25	
05-18	Last fixing in Visual Studio - work with report	1:28:23	Siljebjork
	WIX	09:49-11:18	
05-18	Omstrukturering av rapporten for database + index	2:08:59	Siljebjork
	WIX	11:21-13:30	
05-19	Fixing modules in sql server and visual studio. Rapport skriving.	0:34:40	Siljebjork
	WIX	09:20-09:54	
05-19	Jobbing med tables og stored procedures (ssms) + rapport skriving + møte	2:14:12	Siljebjork
	WIX	16:15-18:29	
05-19	Rapport skriving	1:38:00	Siljebjork
	WIX	19:35-21:13	
05-20	Rapport skriving på backend - controllers, autentisering og autorisasjon + møte + hjelpe med andre punkter i rapporten	8:20:50	Siljebjork
	WIX	10:28-18:49	

Created with toggl.com

6.5 Sprinter Jira board

WIX Sprint 1 ▾ ...

Status Report * Issue added to sprint after start time

Completed Issues						View in Issue Navigator
Key	Summary	Issue Type	Priority	Status	Story Points (-)	
WIX-4	Vurdering Angular Vs React	Task	Highest	DONE	-	
WIX-11 *	[Learning] Introduksjon	Task	Medium	DONE	-	
WIX-12 *	[Learning] Walking skeleton part 1	Task	Medium	DONE	-	
WIX-13 *	[Learning] Walking skeleton part 2	Task	Medium	DONE	-	

Issues Not Completed						View in Issue Navigator
Key	Summary	Issue Type	Priority	Status	Story Points (-)	
WIX-1 *	[Design] Develop UI from the Admin perspective	Task	Medium	IN PROGRESS	-	
WIX-2 *	[Git Hub] Debug common work in the GitHub	Story	Highest	IN PROGRESS	-	
WIX-3	Gå gjennom JavaScript kurs	Task	Highest	TO DO	-	
WIX-5 *	[Design] Develop UI from the users (students) perspective	Task	Highest	IN PROGRESS	-	
WIX-7 *	[Workflow]	Task	Medium	READY FOR QA	-	

Projects / WIX / WIX board

WIX Sprint 2

Starte med login og Frontend.

☆ 0 days remaining Complete sprint ...

Only My Issues Recently Updated

BLOCKED	TO DO	IN PROGRESS	READY FOR QA	DONE
	<ul style="list-style-type: none"> [Rettigheter til admin] Hoved admin ✓ ↗ AZ WIX-16 [Rettigheter til admin] Kurs admin ✓ ↗ WIX-17 	<ul style="list-style-type: none"> Gå gjennom JavaScript kurs ✓ ↗ AZ WIX-3 [Design] Develop UI from the users (students) perspective ✓ ↗ WIX-5 [Design] Develop UI from the Admin perspective ✓ ↗ WIX-1 Registrer Bruker ✓ ↗ - WIX-22 		<ul style="list-style-type: none"> [Workflow] ✓ ↗ AZ WIX-7 [Git Hub] Debug common work in the GitHub ✓ ↗ - WIX-2 [Home Page] ✓ ↗ AZ WIX-8 [Modul Main Page] ✓ ↗ AZ WIX-10 [About Page] -
		<ul style="list-style-type: none"> Registrer Bruker ✓ ↗ - WIX-22 Logg inn funksjonalitet etter registrert bruker ✓ ↗ AZ WIX-15 [front-backend] Login - 		<ul style="list-style-type: none"> ✓ ↗ AZ WIX-10 [About Page] ✓ ↗ AZ WIX-19 [Login Page] FrontEnd ✓ ↗ AZ WIX-25

IN PROGRESS	READY FOR QA	DONE
[front-backend] Login functionality with facebook WIX-23		[Finish Page] Frontend WIX-25
Check normal log in from Udemy Course WIX-24		[Finish Page] Frontend WIX-26
[Video Page] User Page Frontend WIX-27		

Projects / WIX / WIX board

WIX Sprint 3

Continue with FrontEnd. Begin with BackEnd.

0 days remaining | Complete sprint | [Share](#) | [...](#)

[Only My Issues](#) | [Recently Updated](#)

BLOCKED	TO DO	IN PROGRESS	READY FOR QA	DONE
	<ul style="list-style-type: none"> [Rettigheter til admin] Hoved admin WIX-16 [Rettigheter til admin] Kurs admin WIX-17 	<ul style="list-style-type: none"> [front-backend] Login functionality with facebook WIX-23 Registrer Bruker WIX-22 Logg inn funksjonalitet etter registrert bruker WIX-15 Check normal log in from Udemy Course WIX-24 		<ul style="list-style-type: none"> [Report] Create list of content WIX-28
				<ul style="list-style-type: none"> [Design] Develop UI from the users (students) perspective WIX-5 [Design] Develop UI from the Admin perspective WIX-4 Database Initialization WIX-34

Projects / WIX / WIX board

WIX Sprint 4 Admin

Be ready with Admin FrontEnd and start to transfer API

1 day remaining | Complete sprint | [Share](#) | [...](#)

[Only My Issues](#) | [Recently Updated](#)

BLOCKED	TO DO	IN PROGRESS	READY FOR QA	DONE
	<ul style="list-style-type: none"> [Front End] Router WIX-46 AZ Skrive inn notatene for kurs siden i google docs WIX-55 AZ 	<ul style="list-style-type: none"> [Learning] React course WIX-44 AZ [Planlegging] Programarkitektur WIX-45 AZ [Backend] Løse diagram for 	<ul style="list-style-type: none"> [Rapport] Presentasjon av kunde WIX-62 AZ [Rapport] Bakgrunn for oppgaven WIX-63 AZ 	<ul style="list-style-type: none"> [Front End] Home Page (Admin) WIX-29 AZ [Frontend] AdminModulMainPage WIX-36 AZ

BLOCKED	TO DO	IN PROGRESS	READY FOR QA	DONE
	<p>API for oppdatering av kurs WIX-57 </p> <p>[Rapport] Skrive "faglige forutsetning" WIX-61 </p> <p>[Rapport] Beskrivelse av løsningen WIX-67 </p>	<p>[Backend] Lage diagram for databasene WIX-50 </p> <p>Create Course WIX-54 </p> <p>[Rapport] Presentasjon av Prosjektgruppe WIX-59 </p> <p>[Rapport] Verktøybeskrivelse WIX-66 </p> <p>[Rapport] Fyll inn Scrum WIX-68 </p>	<p>[Rapport] Problemstilling WIX-64 </p> <p>[Rapport] Konsept WIX-65 </p>	<p>[Breadcrumbs] Add breadcrumbs to the particular Pages WIX-35 </p> <p>[PM] Describe Phases of the project. WIX-37</p> <p>frontend - login functionality with facebook WIX-40 </p> <p>frontend - login functionality with google WIX-41 </p>
		<p>Fix database issue so everyone can have access to it WIX-74 </p> <p>[backend] Login for admin WIX-72 </p> <p>SQL Sporing WIX-73 </p>		<p>Backend - creating database for admin and users WIX-43 </p> <p>API for sletting av kurs WIX-56 </p> <p>API for å se kurs, altså "View Course" WIX-58 </p> <p>[FrontEnd] Endre Login Side WIX-69 </p> <p>Bytte databaser for Kurs fra Entity Framework til Dapper WIX-70 </p>

WIX Sprint 5 API (Admin)

 0 days remaining Complete sprint



BLOCKED		TO DO	IN PROGRESS	READY FOR QA	DONE
		<p>Skrive inn notatene for kurs siden i google docs WIX-55 AS</p> <p>API for oppdatering av kurs WIX-57 AS</p>	<p>[Front End] Router WIX-46 AZ</p> <p>Create Course WIX-54 AS</p> <p>[Rapport]Beskrivelse av løsningen WIX-67 AS</p> <p>[Rapport] Publishing in Azure WIX-89 SB</p> <p>[Rapport] Updating doc for databases in SQL server WIX-90 AS</p> <p>[Rapport] - functionality for login WIX-91 SB</p> <p>[Report] Describe frontend part WIX-98 AZ</p> <p>Fikse backend, skrive rapport WIX-103</p>	<p>[Rapport]Presentasjon av Prosjektgruppe WIX-59</p> <p>[Rapport] Presentasjon av kunde WIX-62</p> <p>[Rapport]Problemstilling WIX-64</p> <p>[Rapport]Fyll inn Scrum (2.2.1) WIX-68</p> <p>[API on frontend] Get data on card of the home page WIX-95 AZ</p>	<p>[backend] Creating roles WIX-88 SB</p> <p>[Learning] React course WIX-44 AZ</p> <p>[Backend]: Lage diagram for databasene WIX-50 N</p> <p>[Rapport]Skrive "faglige forutsetning" WIX-61</p> <p>[Rapport]Bakgrunn for oppgaven WIX-63</p> <p>[Rapport]Konsept WIX-65</p> <p>[Rapport]Verktøybeskrivelse WIX-66</p> <p>Fix database issue so everyone can have access to it WIX-74 SB</p> <p>[backend] Login for alle brukere WIX-72 SB</p> <p>[Rapport] Database for bruker og kurs WIX-75 SB</p> <p>[Front End] Adding data instead of static WIX-78 AZ</p> <p>Implementation of Database Diagram to SQL Server Management Studio and Visual WIX-79 N</p> <p>Publisering av API i Azure WIX-80 SB</p> <p>Oppdatering av stored procedures i SQL server WIX-81 SB</p>

			[Frontend] Admin Video Side WIX-85 AZ
			[Rapport] Updating doc for databases in SQL server WIX-90 AZ
			[API on frontend] Create course WIX-92 AZ
			[API on frontend] Edit course WIX-93 AZ
			[API on frontend] Delete course WIX-94 AZ
			[API on frontend] Delete course WIX-94 AZ
			[Report] Coworking between Frontend and Backend WIX-96 AZ
			[API on frontend] Authorisation WIX-97 AZ

Projects / WIX / WIX board

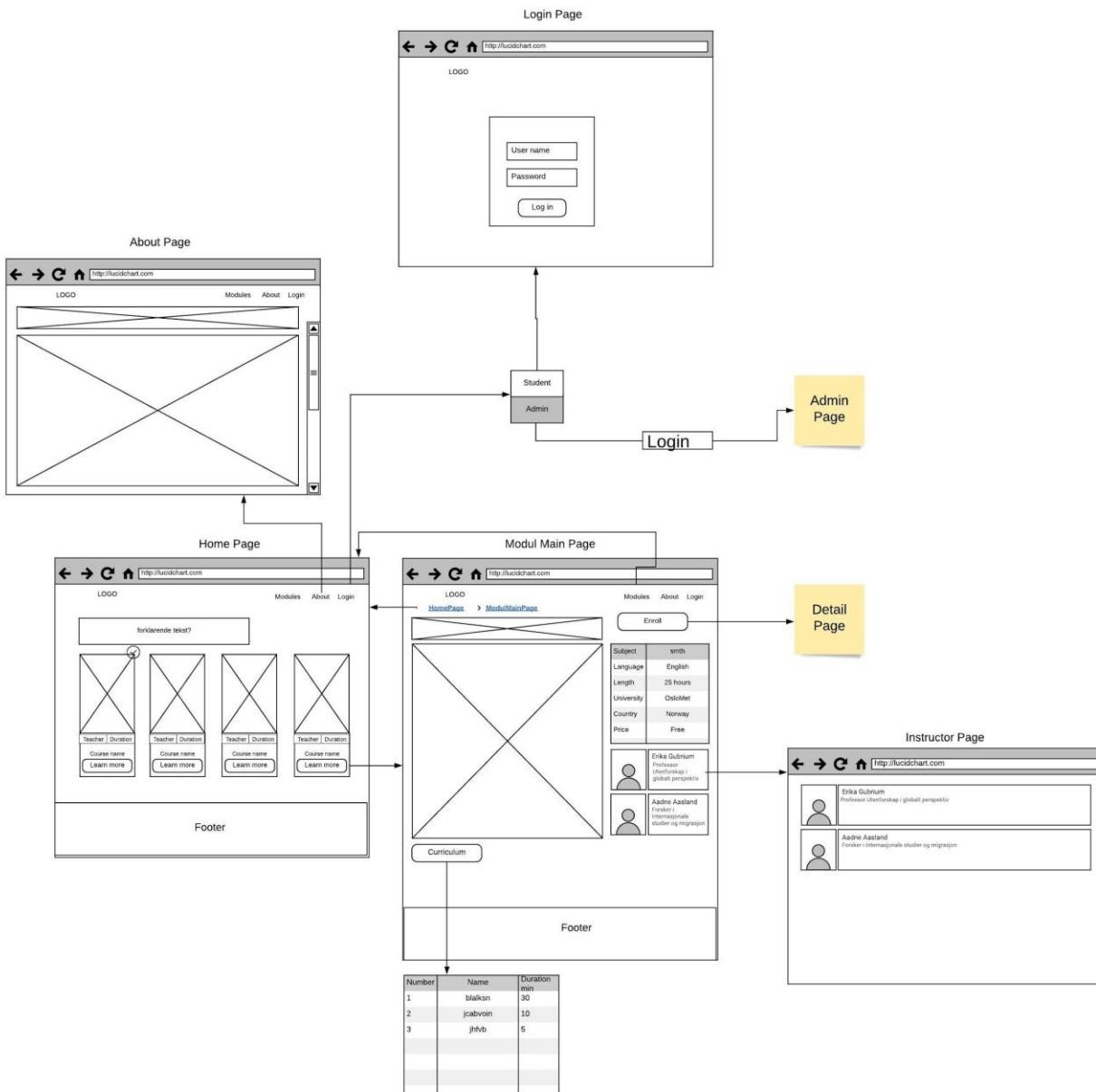
WIX Sprint 6

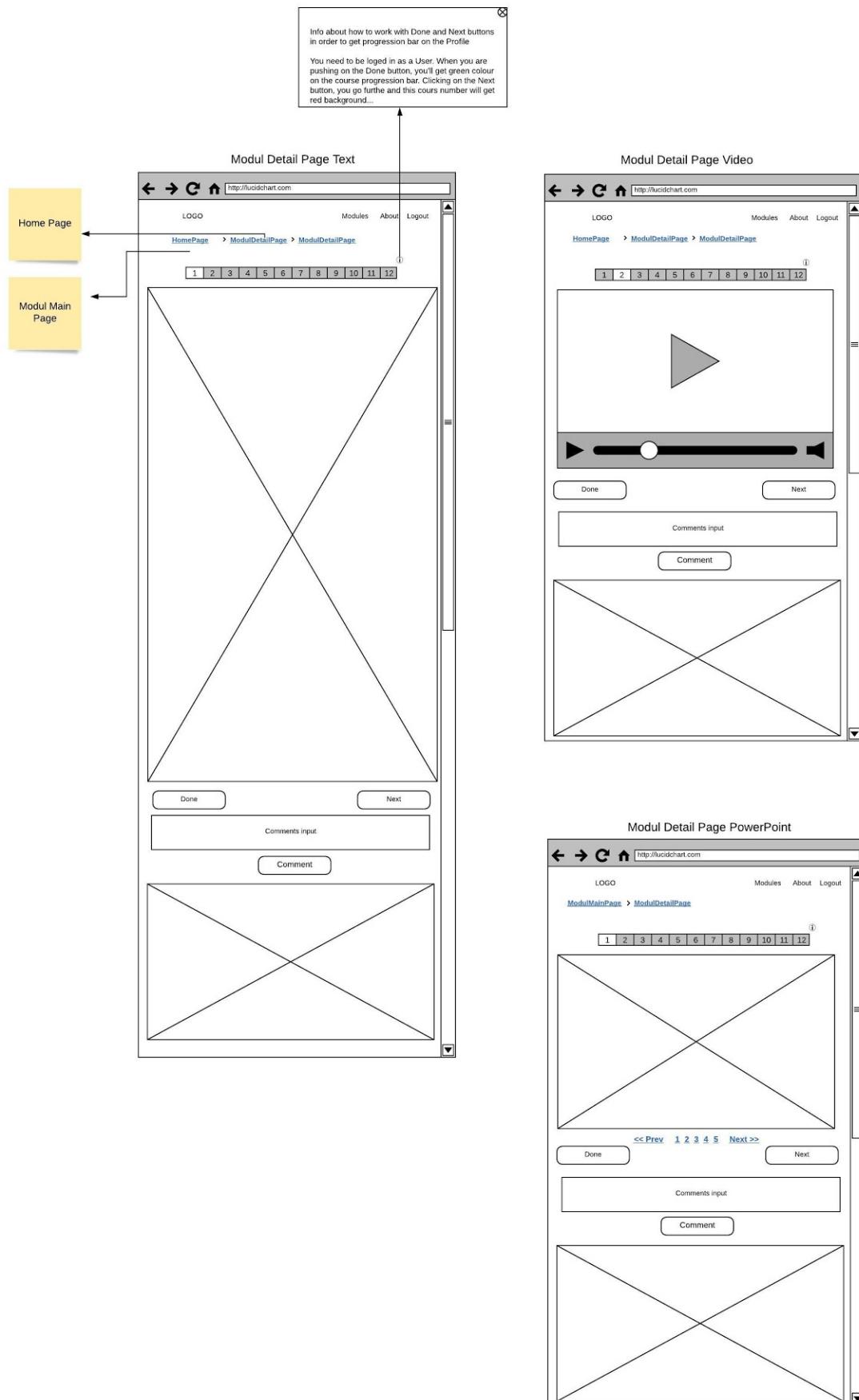
Testing, Rapport, Stabilisering

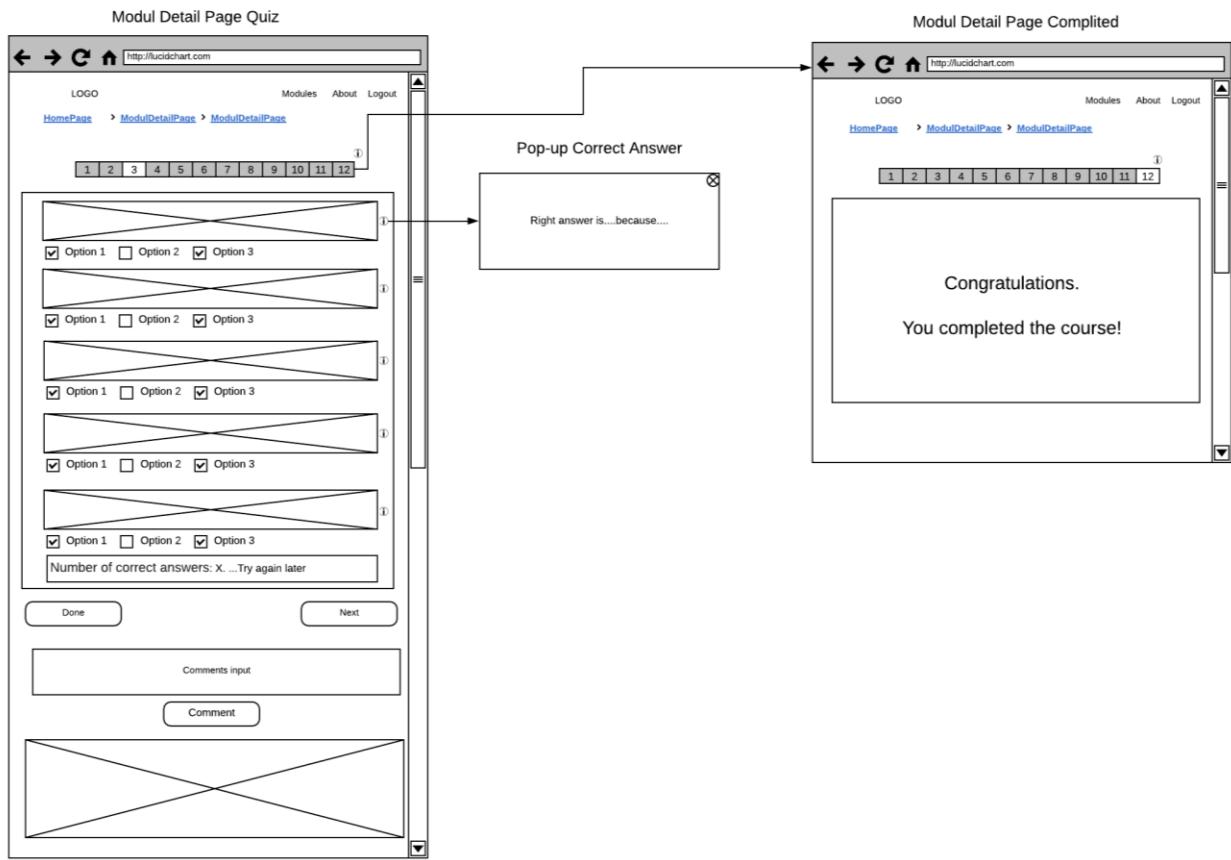
☆ 2 days remaining Complete sprint ⚙ ...

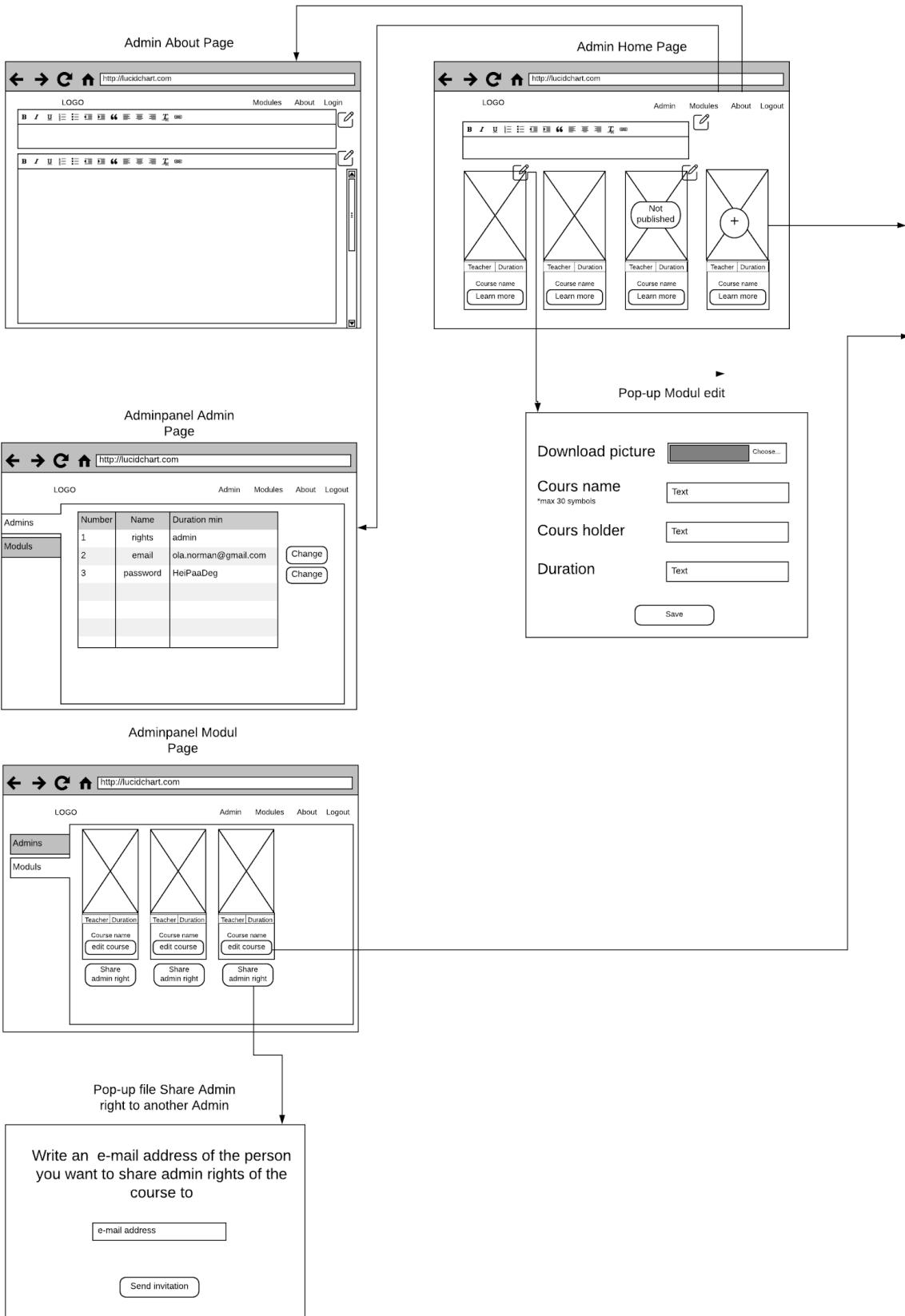
BLOCKED	TO DO	IN PROGRESS	READY FOR QA	DONE
	(Rapport) om GitHub WIX-104	[Rapport] Publishing in Azure WIX-89 SB	[Rapport]Fyll inn Scrum (2.2.1) WIX-68	[Front End] Router WIX-46 AZ
	(Rapport) Kravspesifikasjon WIX-105	[Rapport] - functionality for login WIX-91 SB	[API on frontend] Get data on card of the home page WIX-95 AZ	[Rapport]Presentasjon av Prosjektgruppe WIX-59
	(Rapport) Testrapport WIX-106	Fikse backend, skrive rapport WIX-103		[Rapport] Presentasjon av kunde WIX-62
	(QA) Systemtesting WIX-107			[Rapport]Problemstilling WIX-64
				[Rapport]Beskrivelse av løsningen WIX-67
				[Report] Describe frontend part WIX-98 AZ

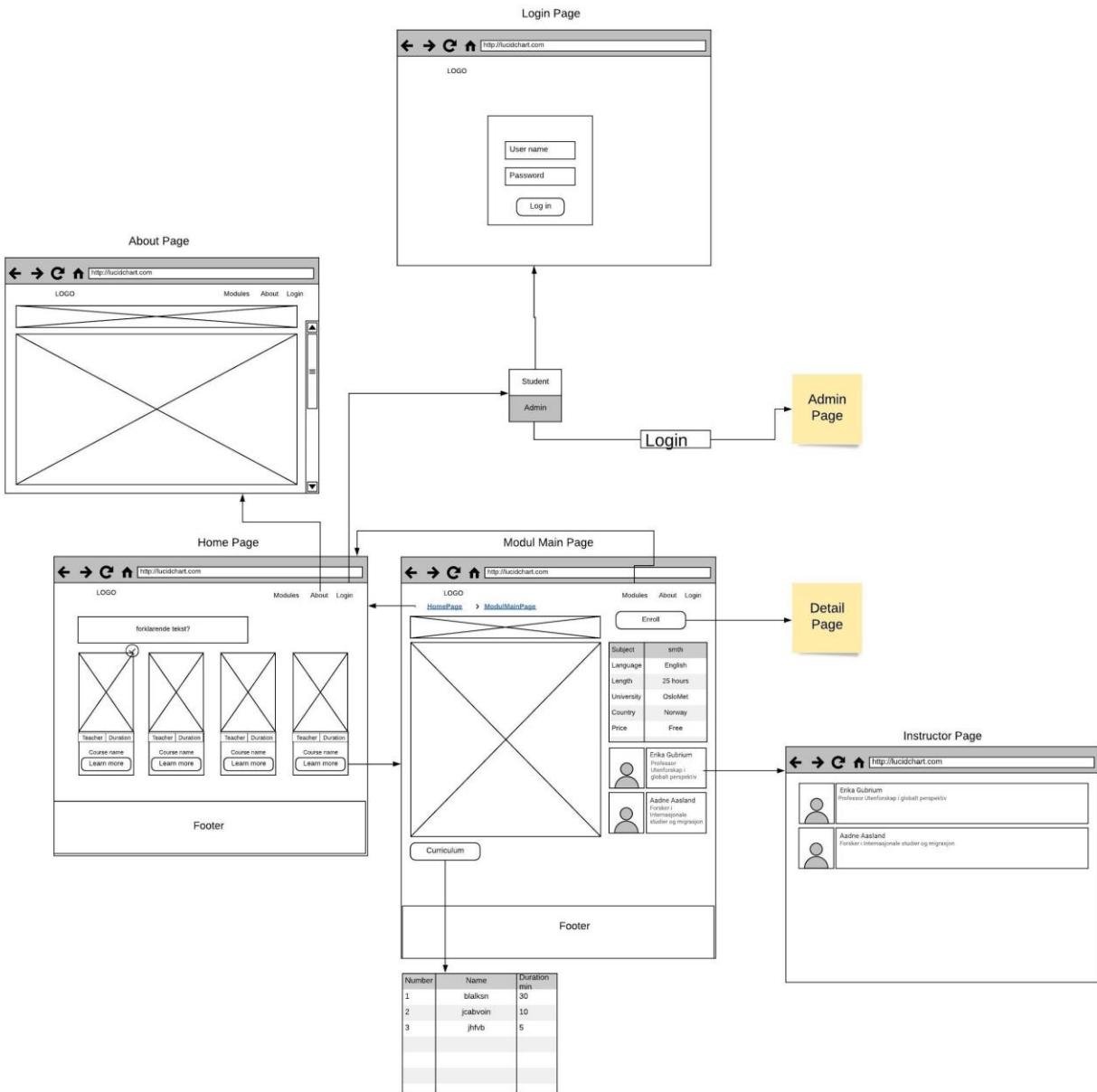
6.6 Wireframe



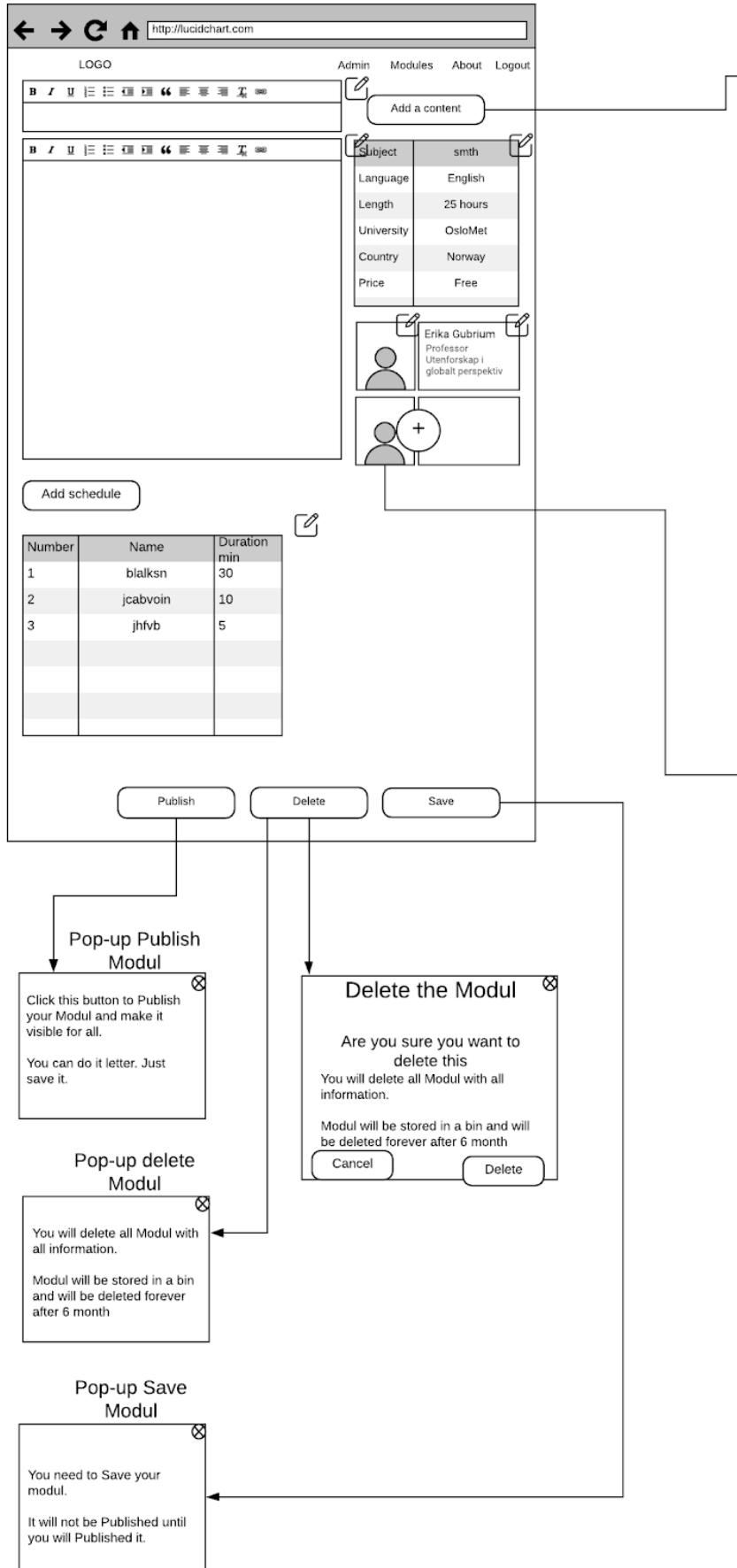


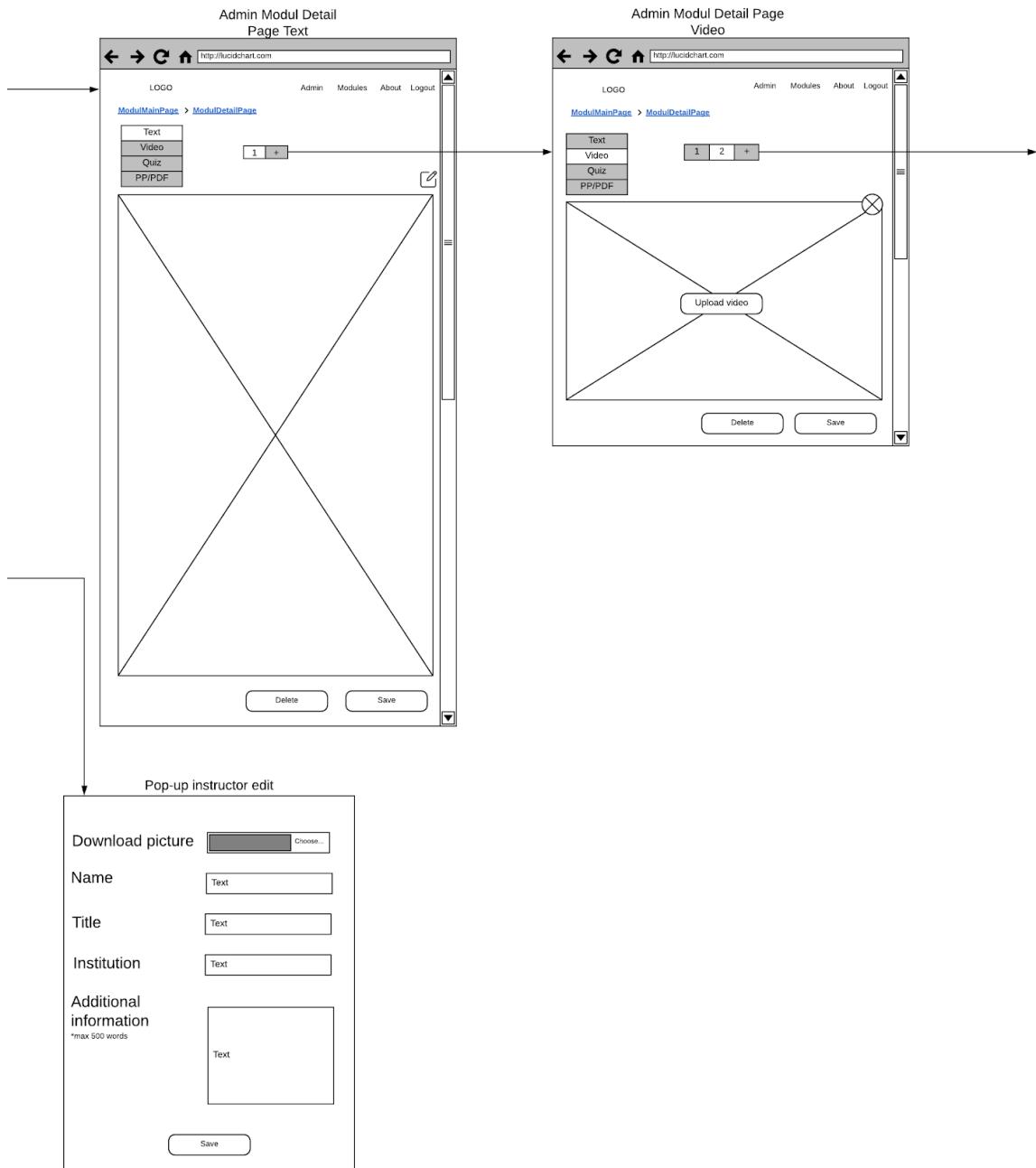




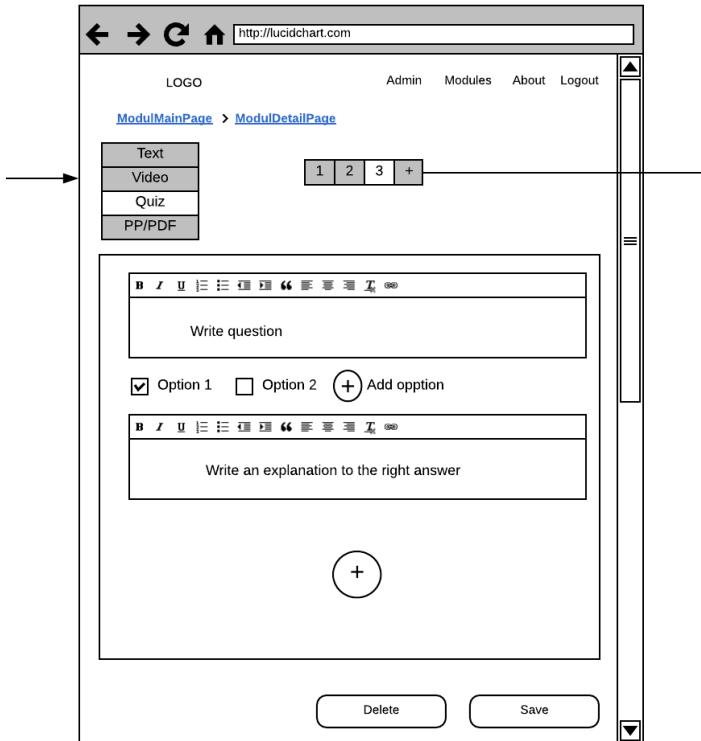


Admin Modul Main Page

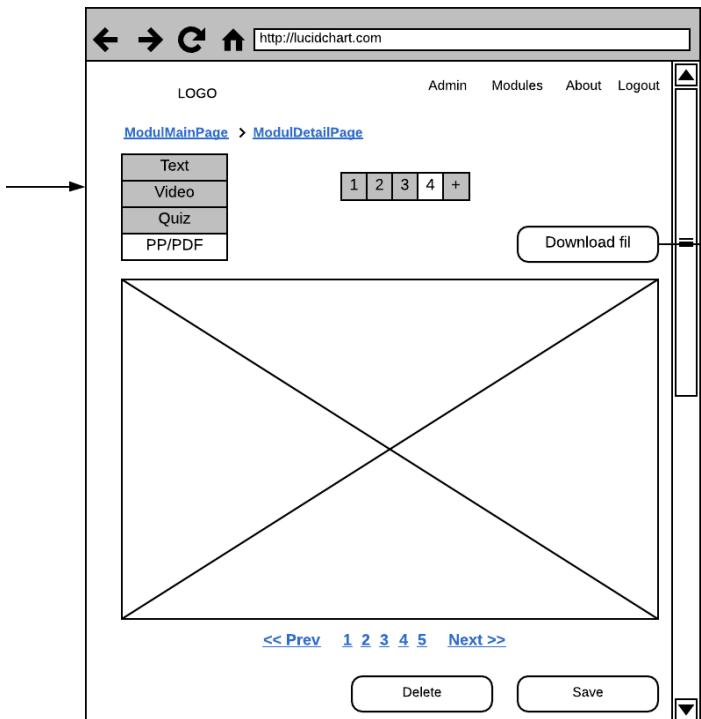




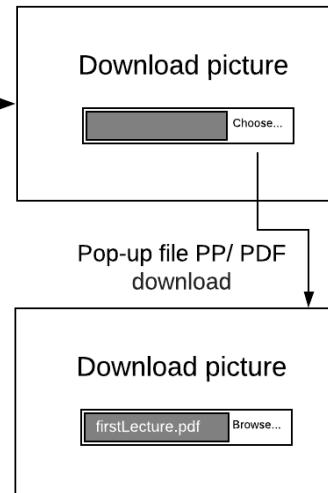
Admin Modul Detail Page Quiz



Modul Detail Page PowerPoint



Pop-up file PP/ PDF



6.7 Figma prototype

The image shows a Figma prototype of a website for 'WIX WORK INCLUSION OF MIGRANTS'. The top navigation bar includes 'MODULES', 'ABOUT', 'LOG IN', and 'REGISTER'. The main section is titled 'Modules' with the sub-instruction 'Select a module you would like to learn more about.' Below this, there are four module cards. The first card on the left is for 'Erika Gabrium' and 'Understanding and measuring poverty', with a duration of '60 min' and a 'LEARN MORE' button. The second card in the middle has a purple border and is labeled 'Kursholder' and 'Tittel' with a 'min' duration and a 'LEARN MORE' button. The third and fourth cards are identical, both labeled 'Kursholder' and 'Tittel' with a 'min' duration and a 'LEARN MORE' button.

About

Developed and led by experts in the fields of social policy, social work, and urban studies, this course provides an introduction to work and social inclusion of migrants in the comparative urban contexts of Bangalore, Moscow and Oslo.

In the course, we will use a visual and interactive approach to explore theories and methods for evaluating work inclusion measures, as well as the crucial strategies and factors that are often not considered. You will gain knowledge of how poverty may be understood and compared across the three settings and will learn alternative conceptual approaches to the more traditional income-focused one.

You will learn new ways to evaluate the work inclusion strategies beyond those currently used. Our focus is on moving beyond an income-only focus in order to improve the social inclusion of target groups through work. Using a "situated" work inclusion approach, the course will provide an appreciation of the historical, social-material and psychological factors shaping the success of inclusion methods targeted to diverse and often marginalized populations and identities.

We will focus on:

- Methods and theories for understanding and measuring poverty in comparative settings
- Policy strategies and necessary considerations developing inclusion measures
- Evaluation of current work inclusion measures using alternative approaches to a focus on income

The course will move between different levels of experiences shaping the success of work and social inclusion measures, including a biographical approach to the everyday life of the target group individuals, the factors of changing place and space before and after migration, as well as the structural and institutional factors shaping inclusion experiences. The concepts and ideas you will learn will help you to reflect upon current practices and improve practices for often marginalized communities.



OSLO METROPOLITAN UNIVERSITY
STORBYUNIVERSITETET



Understanding and measuring poverty

Overview

This sub-section starts with an outline of welfare rights and benefits in Norway, India and Russia.

- What does the welfare state look like in these countries?
- What sorts of public understandings lie behind the formation of their welfare systems?
- How do these understandings, as well as the actual welfare benefits, shape the experience of welfare recipients and especially those with the fewest resources?

Then we take a closer look at migrants:

- What are the welfare measures available to migrants in Oslo, Bangalore and Moscow?
- Do they differ from those available to non-migrants?
- What are the social and political factors behind these measures?

[START](#)

Length	60min
Institution	OsloMet
Subject	Social welfare
Price	Free
Language	English

Course Leader



Erika Gubrium
Researcher
OsloMet



Erika Gubrium
Researcher
OsloMet

Curriculum

Number	Name	Duration
1	The Norwegian welfare state	4min
2	Equality and Compromise	4min
3	Equality and Compromise	4min



< Prev 1 2 3 4 5 6 7 Next >

The Norwegian Welfare State

Ideals in tension

Norway is a thin, stretched country in Scandinavia. It has a population of only 5 million people. Oslo, the capital, has about 650,000 inhabitants. The metropolitan area, which includes the densely populated areas around Oslo proper, has 1.7 million people.

In Norway all political parties basically agree that the quality of health, education and social services should be as similar as possible throughout the country.

The Norwegian welfare state has the following general characteristics:

- Benefits and services are financed through a re-distributive, general tax system. Those with lower incomes pay relatively less and those with higher incomes (up to a limit) pay relatively more.
- The welfare system is founded on the ideals of equality, social justice, social security, solidarity and social integration. These ideals are subject to positive and negative tensions.
- The goals of economic redistribution and social integration require high employment and economically active people.
- The focus on employment and employability may clash with the ideals of equality, social justice and social security.

Resources

Wikipedia. Oslo.

World Population Review. Oslo population 2018. The World Population Review is a private, US based web site which presents demographic data in a convenient way.

Image

Lysefjord, Norway.

NEXT



< Prev 1 2 ③ 4 5 6 7 Next >

Early Reforms

1. When is the Norwegian constitution day?

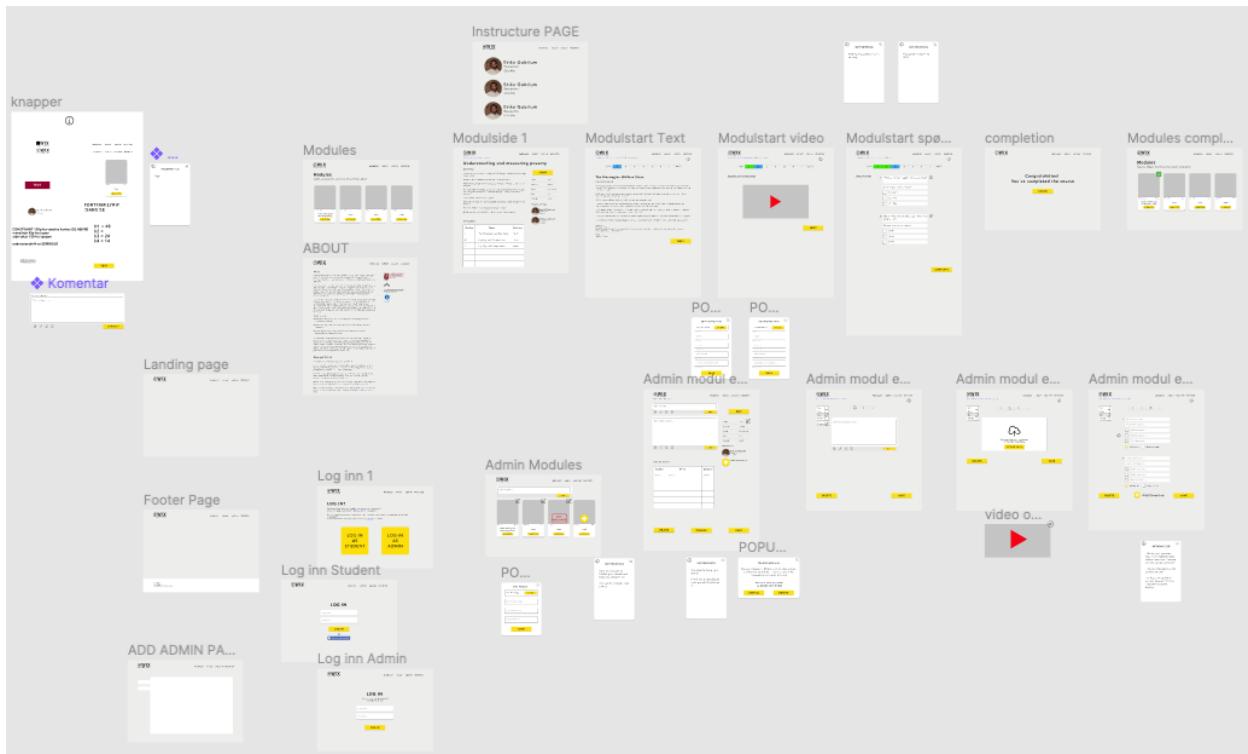
Choose one correct answer

- 16. April
- 16. May
- 17. May

2. Which year did Norway get there own constitution

Choose one correct answer

- 1818
- 1814
- 1816



6.8 Vedlegg for login funksjonalitet til Facebook og Google - første forsøk.

Vi legger den delen her selv om den ikke ble bygget i prosjektet, fordi den viser en annen måte at den kunne blitt laget på.

For å legge til logg inn funksjonaliteten med Facebook, så gikk vi til www.developers.facebook.com > My Apps > Create App > Set Up. I Settings > Basic kopierte vi App ID and App Secret, for å kunne bruke dem senere.

I VS code åpnet vi Kommando-vinduet og tastet npm install react-facebook-login.³⁷ Deretter går vi til src>LoginPage og åpner mappe "components". I denne mappen, åpner vi en fil med navnet "Facebook.js". Den inneholder følgende kode:

³⁷ For denne delen av prosjektet, etter å ha prøvd forskjellige modeller, så bestemte vi å bruke metoden fra denne linken: <https://medium.com/recraftrelic/login-with-facebook-and-google-in-reactjs-990d818d5dab>

```

import React, { Component } from 'react';
import '../LoginPage.css';
import FacebookLogin from 'react-facebook-login';

class Facebook extends Component {

  render() {

    const responseFacebook = (response) => {
      console.log(response);
    }

    return (
      <div className="Facebook">

        <FacebookLogin
          appId="1315775898620150"
          fields="name,email,picture"
          callback={responseFacebook}
        />

      </div>
    );
  }
}

export default Facebook;

```

I src>LoginPage>LoginPage.js, la vi til en ny rad, som vi skal bruke for login med sosial media knapper. Siden skal se slik ut:

```

<Row>
  <Col>
    <div className ="Facebook">

    </div>
  </Col>
  <Col>
    <div className ="Gmail">

    </div>
  </Col>
  <Col>
    <div className ="Microsoft">

    </div>
  </Col>
</Row>

```

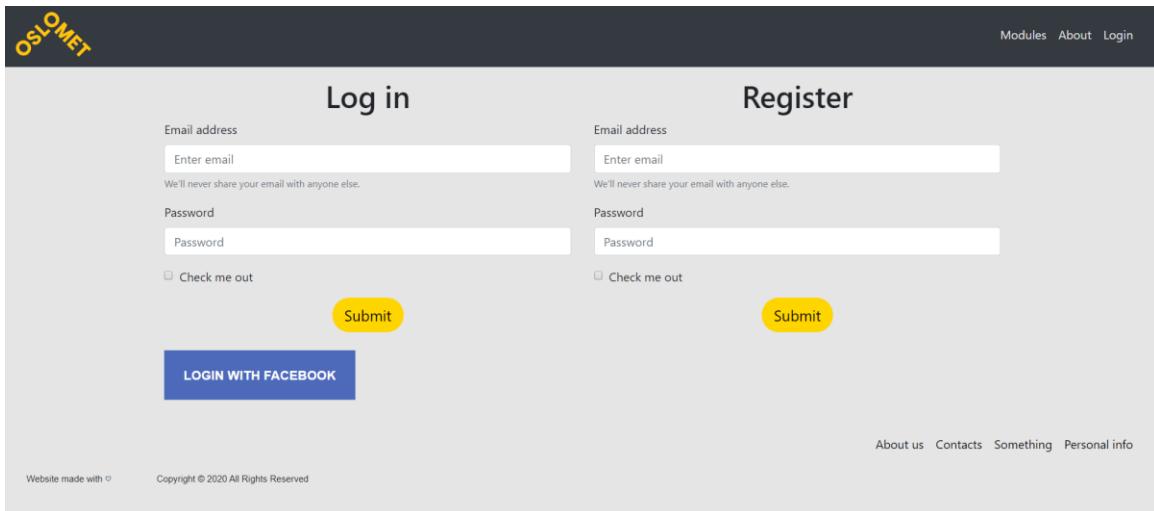
På samme side, legger vi til "import Facebook from './components/Facebook';". I <div> className "Facebook", så legger vi til "<Facebook/>":

```

<div className ="Facebook">
  <Facebook />
</div>

```

Når vi kjører programmet, så skal siden se slik ut:



Vi fortsetter med å legge til login funksjonaliteten for Google. Først går vi til Google API Console. Her lager vi client ID og klikker på "Create credentials".

I OAuth client ID, velger vi applikasjonstype "Web application".

I "Authorized JavaScript origins" og "Authorized redirect URLs" legger vi til hjemmeside URL-en, som var `https://localhost:3000/` da denne delen av dokumentasjonen ble skrevet. Så klikker vi på "Create Button".

På klientside fulgte vi de samme stegene som for å lage en knapp for Facebook. I kommandovindu på VS code, kjørte vi `npm install react-google-login`.

På selv koden, i `src>LoginPage>components`, åpnet vi en fil "Gmail.js", som så slik ut:

```
import React, { Component } from 'react';
import './LoginPage.css';
import GoogleLogin from 'react-google-login';

class Google extends Component {

  render() {

    const responseGoogle = (response) =>
      console.log(response);

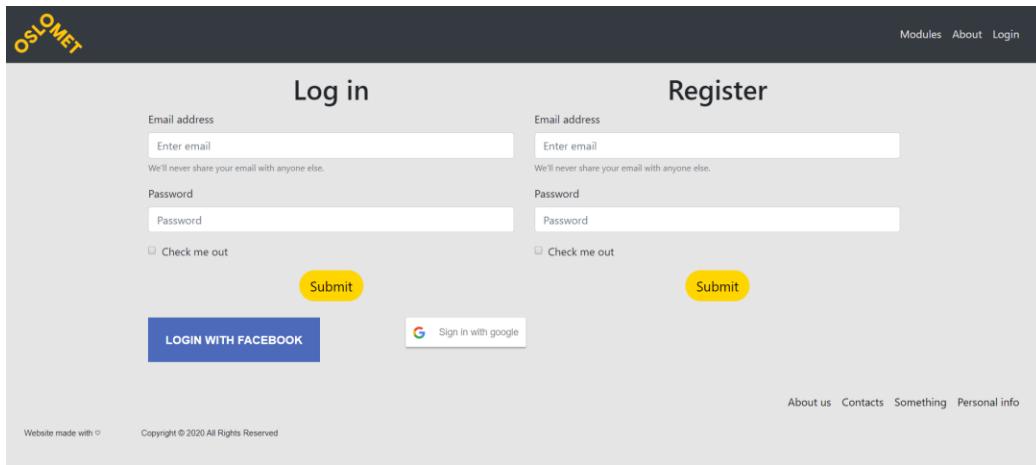
    return (
      <div className="Gmail">
        <GoogleLogin
          clientId="903553020968-hias98fifmunqoeq3o4da5jj8sjgl53v.apps.googleusercontent.com"
          buttonText="Sign in with google"
          onSuccess={responseGoogle}
          onFailure={responseGoogle}
        />
      </div>
    );
  }
}

export default Google;
```

I src>LoginPage>LoginPage.js, la vi til kodelinje “import Google from './components/Facebook'”. Deretter la vi til en <div> med className “Google”, som så slik ut:

```
<Col>
  <div className ="Google">
    <Google/>
  </div>
</Col>
```

Når vi kjørte koden med “Run npm start” i kommando vindu, så siden slik ut:

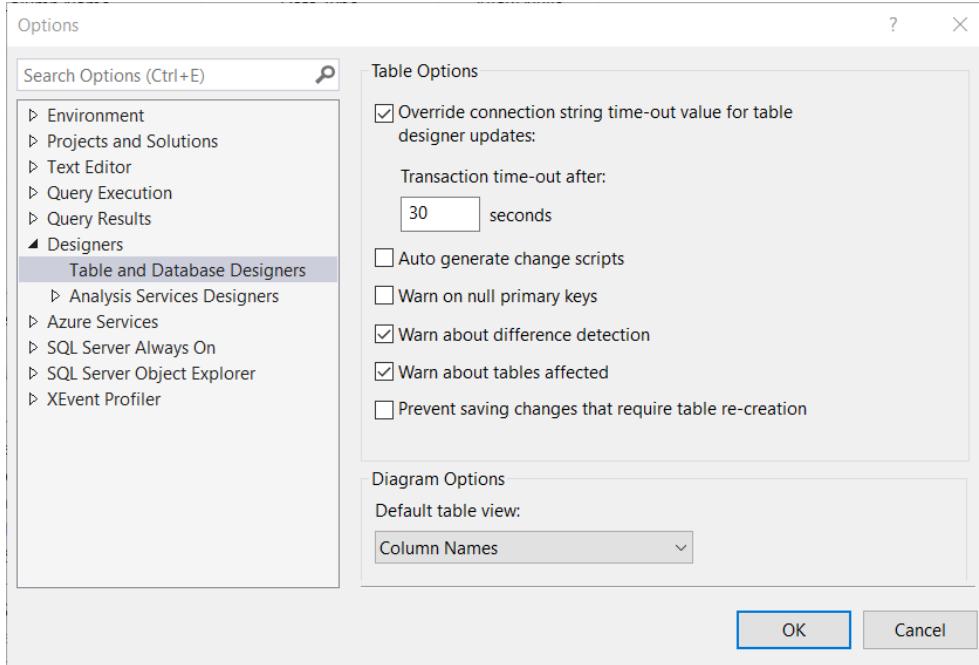


6.8 Vedlegg for backend

Første stegene i SQL Server Management Studio

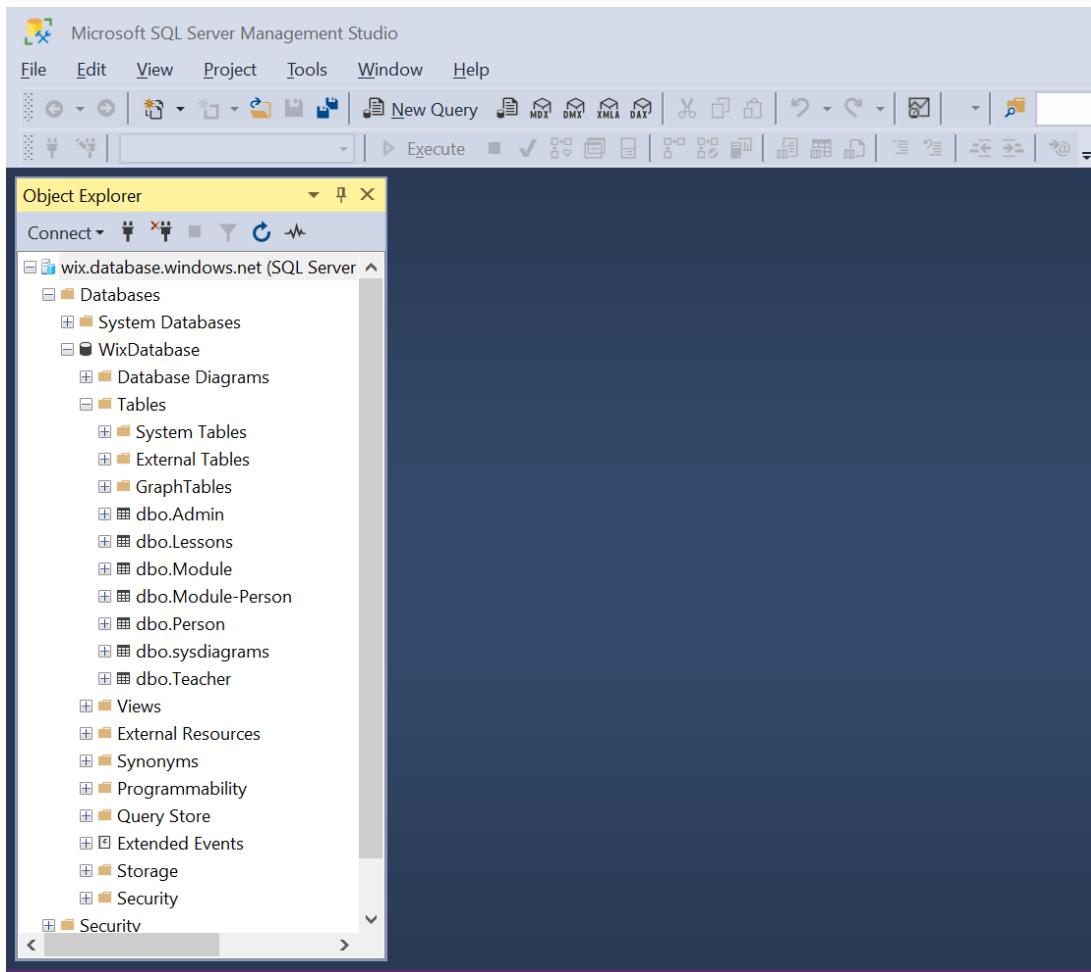
Når vi logger inn med SQL Server Management Studio (SSMS), før vi starter å bygge databasen, går vi i Tools → Options → Designers → Table and Database Designers, og velger ut “Prevent saving

changes that require table re-creation”, slik at det skal være mulig å lage database schemaer:



I Object Explorer, høyreklikk i Databases for å lage en ny database.

Etter at databasen og tabellene er bygget, skal databasen se slik ut når man kobler seg på SSMS:



Tabeller i SQL Server

Prosess for å lage tabeller i SQL server

(i) Person

Vi lagde først en tabell for vanlige personer som skal ta kurs, admins, og teachers:

Column Name	Data Type	Allow Nulls
personID	int	<input type="checkbox"/>
email	nvarchar(250)	<input checked="" type="checkbox"/>
password	nvarchar(20)	<input type="checkbox"/>
name	nvarchar(50)	<input type="checkbox"/>
surname	nvarchar(50)	<input type="checkbox"/>
postnr	nchar(4)	<input type="checkbox"/>
address	nvarchar(50)	<input checked="" type="checkbox"/>
username	nvarchar(50)	<input type="checkbox"/>
		<input type="checkbox"/>

Attributtet "personID" skal være autogenerert. For å få det til, så klikker vi i personID og, "Column Properties → Identity Specification → (Is Identity)", valgte vi "Yes", slik det vises på bildet nederst:

Column Properties	
	No
	No
	Yes
	<u>Yes</u>
	1
	1
	Yes
	No

På denne måten, så blir "personID" autogenerert og alltid unik. Derfor valgte vi den som primary key for denne tabellen.

(ii) Teacher:

Column Name	Data Type	Allow Nulls
teacherID	int	<input type="checkbox"/>
title	nchar(20)	<input type="checkbox"/>
institution	nchar(20)	<input type="checkbox"/>
personID	int	<input type="checkbox"/>
image	nvarchar(50)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Attributtet "teacherID" skal være primary key til Teacher tabellen. Teacher tabellen skal inneholde en "personID", og den skal hjelpe til at hver Teacher som blir laget er unik.

(iii) Admin:

Column Name	Data Type	Allow Nulls
adminID	int	<input type="checkbox"/>
birthnumber	nchar(11)	<input type="checkbox"/>
personID	int	<input type="checkbox"/>
telNr	nchar(12)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

“adminID” skal være primary key til *Admin* tabellen. *Admin* tabellen skal inneholde en “personID”, og den skal hjelpe til at hver *Admin* som blir laget er unik.

(iv) Module:

Samme prosess skal følges for Module:

Column Name	Data Type	Allow Nulls
moduleId	int	<input type="checkbox"/>
institution	nchar(15)	<input checked="" type="checkbox"/>
price	float	<input checked="" type="checkbox"/>
description	nchar(30)	<input checked="" type="checkbox"/>
language	nchar(20)	<input checked="" type="checkbox"/>
duration	float	<input checked="" type="checkbox"/>
title	nchar(20)	<input checked="" type="checkbox"/>
teacherID	int	<input checked="" type="checkbox"/>
adminID	int	<input checked="" type="checkbox"/>
picture	nvarchar(50)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

“moduleId” skal være primary key til *Module* tabellen. *Module* skal ha 2 foreign keys fra Teacher og Admin tabel. Vi gjør det slik at en admin og teacher er registrert til å redigere eller legge på enhver module som blir laget.

(v) Module-Person:

Column Name	Data Type	Allow Nulls
moduleId	int	<input type="checkbox"/>
personID	int	<input type="checkbox"/>
		<input type="checkbox"/>

Den tabellen har 2 primary keys som er hentet fra *Module* og *Person* tabellene. Meningen med den er å fungere som en “brua” mellom de to tabellene. På den måten vi kan ha en oversikt over hvilke personer er registrert på de *modulene* som etterhvert blir laget.

(vi) Lesson:

wix.WixDatabase - dbo.Lessons		
Column Name	Data Type	Allow Nulls
lessonID	int	<input type="checkbox"/>
type	nchar(15)	<input checked="" type="checkbox"/>
moduleId	int	<input checked="" type="checkbox"/>
name	nchar(20)	<input checked="" type="checkbox"/>
details	nchar(50)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Tabell for person,teacher og admin:

(i) Person:

Column Name	Data Type	Allow Nulls
personID	int	<input type="checkbox"/>
email	nvarchar(250)	<input checked="" type="checkbox"/>
password	nvarchar(20)	<input type="checkbox"/>
name	nvarchar(50)	<input type="checkbox"/>
surname	nvarchar(50)	<input type="checkbox"/>
postnr	nchar(4)	<input type="checkbox"/>
address	nvarchar(50)	<input checked="" type="checkbox"/>
username	nvarchar(50)	<input type="checkbox"/>
		<input type="checkbox"/>

(ii) Teacher:

Column Name	Data Type	Allow Nulls
teacherID	int	<input type="checkbox"/>
title	nchar(20)	<input type="checkbox"/>
institution	nchar(20)	<input type="checkbox"/>
personID	int	<input type="checkbox"/>
image	nvarchar(50)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

(iii) Admin:

Column Name	Data Type	Allow Nulls
adminID	int	<input type="checkbox"/>
birthnumber	nchar(11)	<input type="checkbox"/>
personID	int	<input type="checkbox"/>
telNr	nchar(12)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Tabel for Module, Lessons og Module-Person:

(iv) Module:

Column Name	Data Type	Allow Nulls
moduleId	int	<input type="checkbox"/>
institution	nchar(15)	<input checked="" type="checkbox"/>
price	float	<input checked="" type="checkbox"/>
description	nchar(30)	<input checked="" type="checkbox"/>
language	nchar(20)	<input checked="" type="checkbox"/>
duration	float	<input checked="" type="checkbox"/>
title	nchar(20)	<input checked="" type="checkbox"/>
teacherID	int	<input checked="" type="checkbox"/>
adminID	int	<input checked="" type="checkbox"/>
picture	nvarchar(50)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

(v) Lessons:

Column Name	Data Type	Allow Nulls
lessonID	int	<input type="checkbox"/>
type	nchar(15)	<input checked="" type="checkbox"/>
moduleID	int	<input checked="" type="checkbox"/>
name	nchar(20)	<input checked="" type="checkbox"/>
details	nchar(50)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

(vi) Module-Person:

Column Name	Data Type	Allow Nulls
moduleID	int	<input type="checkbox"/>
personID	int	<input type="checkbox"/>
		<input type="checkbox"/>

Stored Procedures - SQL Server

System Stored Procedures
dbo.CreateAdmin
dbo.CreateLesson
dbo.CreateModule
dbo.CreatePerson
dbo.CreateTeacher
dbo.DeleteAdmin
dbo.DeleteLesson
dbo.DeleteModule
dbo.DeletePerson
dbo.DeleteTeacher
dbo.FetchAdmin
dbo.FetchAdminByID
dbo.FetchLesson
dbo.FetchLessonByID
dbo.FetchModule
dbo.FetchModuleByID
dbo.FetchPerson
dbo.FetchPersonByID
dbo.FetchTeacher
dbo.FetchTeacherByID
dbo.sp_ alterdiagram
dbo.sp_creatediagram
dbo.sp_dropdiagram
dbo.sp_helpdiagramdefinition
dbo.sp_helpdiagrams
dbo.sp_renamediagram
dbo.sp_upgradediagrams
dbo.UpdateAdmin
dbo.UpdateLesson
dbo.UpdateModule
dbo.UpdatePerson
dbo.UpdateTeacher

Prosedyrene for Admin:

CreateAdmin:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[CreateAdmin]
    @birthnumber nchar(11), @personID int, @telNr nchar(12)
AS
BEGIN
    INSERT INTO [dbo].[Admin] (birthnumber, personID, telNr)
        VALUES (@birthnumber, @personID, @telNr)
END

```

UpdateAdmin:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[UpdateAdmin]
    @adminID int, @birthnumber nchar(11), @telNr nchar(12)
AS
BEGIN
    UPDATE dbo.[Admin]
        SET birthnumber = @birthnumber,
            telNr = @telNr
    WHERE adminID = @adminID
END

```

DeleteAdmin:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[DeleteAdmin]
    @adminID int
AS
BEGIN
    DELETE dbo.[Admin]
    WHERE adminID = @adminID
END

```

FetchAdminByID:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[FetchAdminByID]
    @adminID int
AS
BEGIN
    SELECT adminID, birthnumber, personID, telNr
    FROM dbo.[Admin]
    WHERE adminID = @adminID
END

```

FetchAdmin:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[FetchAdmin]

AS
BEGIN
    SELECT adminID, birthnumber, personID, telNr FROM dbo.[Admin]
END

```

Prosedyrene for Lesson:

CreateLesson:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [dbo].[CreateLesson]
    @type nchar(15), @moduleID int, @name nchar(20), @details nchar(50)

AS
BEGIN
    INSERT INTO [dbo].[Lessons] ([type], moduleID, [name], details)
    VALUES (@type, @moduleID, @name, @details)
END

```

UpdateLesson:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[UpdateLesson]
    @lessonID int, @type nchar(15), @moduleID int, @name nchar(20), @details nchar(50)

AS
BEGIN
    UPDATE [dbo].[Lessons]
        SET
            [type] = @type,
            moduleID = @moduleID,
            [name] = @name,
            details = @details
        WHERE lessonID = @lessonID
END

```

DeleteLesson:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[DeleteLesson]
    @lessonID int

AS
BEGIN
    DELETE dbo.[Lessons]
    WHERE lessonID = @lessonID
END

```

FetchLessonByID:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[FetchLessonByID]
    @lessonID int
AS
BEGIN
    SELECT lessonID, [type], moduleID, [name], details
    FROM dbo.[Lessons]
    WHERE lessonID = @lessonID
END

```

FetchLesson:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[FetchLesson]

AS
BEGIN
    SELECT lessonID, [type], moduleID, [name], details FROM dbo.[Lessons]
END

```

Procedure for Module:

```

ALTER PROCEDURE [dbo].[CreateModule]
    @title nchar(20), @teacherID int, @adminID int, @institution nchar (15), @price float, @description nchar(30), @language nchar(20), @picture nvarchar, @subject nchar(20)
AS
BEGIN
    INSERT INTO [dbo].[Module] (title, teacherID, adminID, institution, price, [description], [language], picture, subject)
    VALUES (@title, @teacherID, @adminID, @institution, @price, @description, @language, @picture, @subject)
END

```

UpdateModule:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [dbo].[updateModule]
    @moduleId int, @title nchar(20), @teacherID int, @adminID int, @institution nchar (15), @price float, @description nchar(30), @language nchar(20), @picture nvarchar, @subject nchar(20)
AS
BEGIN
    UPDATE dbo.Module
    SET title = @title,
        teacherID = @teacherID,
        adminID = @adminID,
        institution = @institution,
        price = @price,
        [description] = @description,
        [language] = @language,
        picture = @picture,
        subject = @subject
    WHERE moduleId = @moduleId
END

```

DeleteModule:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [dbo].[DeleteModule]
    @moduleID int
AS
BEGIN

    DELETE dbo.Module
    WHERE moduleID = @moduleID

END

```

FetchModuleByID:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [dbo].[FetchModuleByID]
    @moduleID int
AS
BEGIN
    SELECT moduleID, title, teacherID, adminID, institution, price, [description], [language], picture, subject
    FROM dbo.Module
    WHERE moduleID = @moduleID

END

```

FetchModule:

```

ALTER PROCEDURE [dbo].[FetchModule]
AS
BEGIN
    SELECT moduleID, title, teacherID, adminID, institution, price, [description], [language], picture, subject FROM dbo.Module
END

```

Prosedyrene i Person:

CreatePerson:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [dbo].[CreatePerson]
    @name nvarchar(50), @surname nvarchar(50), @username nvarchar(50), @email nvarchar(250), @password nvarchar(20), @postnr nchar(4), @address nvarchar(50)
AS
BEGIN
    INSERT INTO [dbo].[Person] ([name], surname, username, email, [password], postnr, [address])
    VALUES (@name, @surname, @username, @email, @password, @postnr, @address)
END

```

UpdatePerson:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[UpdatePerson]
    @personID int, @name nvarchar(50), @surname nvarchar(50), @username nvarchar(50), @email nvarchar(250), @password nvarchar(20), @postnr nchar(4), @address nvarchar(50)
AS
BEGIN
    UPDATE dbo.[Person]
    SET [name] = @name,
        surname = @surname,
        username = @username,
        email = @email,
        [password] = @password,
        postnr = @postnr,
        [address] = @address
    WHERE personID = @personID
END

```

DeletePerson:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [dbo].[DeletePerson]
    @personID int
AS
BEGIN
    DELETE dbo.[Person]
    WHERE personID = @personID
END

```

FetchPersonByID:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [dbo].[FetchPersonByID]
    @personID int
AS
BEGIN
    SELECT [personID],[email],[password],[name],[surname],[postnr],[address],[username]
    , (SELECT TOP 1 adminID FROM [Admin] AS A WHERE A.personID = P.personID) AS adminID
    , (SELECT TOP 1 teacherID FROM Teacher AS T WHERE T.personID = P.personID) AS teacherID
    FROM [Person] AS [P]
    WHERE personID = @personID
END

```

FetchPerson:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [dbo].[FetchPerson]
AS
BEGIN
SELECT [personID],[email],[password],[name],[surname],[postnr],[address],[username]
, (SELECT TOP 1 adminID FROM [Admin] AS A WHERE A.personID = P.personID) AS adminID
, (SELECT TOP 1 teacherID FROM Teacher AS T WHERE T.personID = P.personID) AS teacherID
FROM [Person] AS [P]
END

```

Prosedyrene i Teacher:

CreateTeacher:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [dbo].[CreateTeacher]
    @title nchar(20), @institution nchar(20), @personID int, @image nvarchar
AS
BEGIN
    INSERT INTO [dbo].[Teacher] (title, institution, personID, [image])
    VALUES (@title, @institution, @personID, @image)
END

```

UpdateTeacher:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[UpdateTeacher]
    @teacherID int, @title nchar(20), @institution nchar(20), @personID int, @image nvarchar(50)
AS
BEGIN
    UPDATE dbo.Teacher
    SET title = @title,
    institution = @institution,
    personID = @personID,
    [image] = @image
    WHERE teacherID = @teacherID
END

```

DeleteTeacher:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[DeleteTeacher]
    @teacherID int
AS
BEGIN
    DELETE dbo.Teacher
    WHERE teacherID = @teacherID
END

```

FetchTeacherByID:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[FetchTeacher]

AS
BEGIN
    SELECT teacherID, title, institution, personID, [image] FROM dbo.[Teacher]
END
```

FetchTeacher:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[FetchTeacher]

AS
BEGIN
    SELECT teacherID, title, institution, personID, [image] FROM dbo.[Teacher]
END
```

Models i Visual Studio

(i) Admin.cs:

```
1  namespace BachelorOppgave.Data.Models
2  {
3      9 references
4      public class Admin
5      {
6          //integers
7          1 reference
8          public int adminID { get; set; }
9          0 references
10         public int personID { get; set; }
11
12         //strings
13         0 references
14         public string birthnumber { get; set; }
15         0 references
16         public string telNr { get; set; }
17     }
18 }
```

(ii) Lesson.cs:

```
1 namespace BachelorOppgave.Data.Models
2 {
3     8 references
4     public class Lesson
5     {
6         //integers
7         1 reference
8         public int lessonID { get; set; }
9         0 references
10        public int moduleID { get; set; }
11        0 references
12        //strings
13        0 references
14        public string type { get; set; }
15        0 references
16        public string name { get; set; }
17        0 references
18        public string details { get; set; }
19    }
20}
```

(iii) Module.cs :

```
1 ✓ namespace BachelorOppgave.Data.Models
2 {
3     9 references
4     public class Module
5     {
6         //integers
7         1 reference
8         public int moduleID { get; set; }
9         1 reference
10        public int teacherID { get; set; }
11        1 reference
12        public int adminID { get; set; }
13        0 references
14        //strings
15        1 reference
16        public string institution { get; set; }
17        1 reference
18        public string description { get; set; }
19        1 reference
20        public string language { get; set; }
21        1 reference
22        public string title { get; set; }
23        1 reference
24        public string picture { get; set; }
25        0 references
26        //etc
27        1 reference
28        public float price { get; set; }
29        0 references
30        public float duration { get; set; }
31    }
32}
```

(iv) Person.cs:

```
1  namespace BachelorOppgave.Data.Models
2  {
3      5 references
4      public class Person
5      {
6          //integers
7          1 reference
8          public int personID { get; set; }
9
10         //strings
11         0 references
12         public string email { get; set; }
13
14         [JsonIgnore]
15         1 reference
16         public string password { get; set; }
17         0 references
18         public string name { get; set; }
19         0 references
20         public string surname { get; set; }
21         0 references
22         public string postnr { get; set; }
23         0 references
24         public string address { get; set; }
25         1 reference
26         public string username { get; set; }
27         1 reference
28         public string Token { get; set; }
29
30         1 reference
31         public int? adminID { get; set; }
32         1 reference
33         public int? teacherID { get; set; }
34     }
35 }
```

(v) Teacher.cs:

```
1  namespace BachelorOppgave.Data.Models
2  {
3      8 references
4      public class Teacher
5      {
6          //integers
7          2 references
8          public int teacherID { get; set; }
9          1 reference
10         public int personID { get; set; }
11
12         //strings
13         1 reference
14         public string title { get; set; }
15         1 reference
16         public string institution { get; set; }
17         1 reference
18         public string image { get; set; } //save as images url or file directory
19     }
20 }
```

(vi) Module-Person.cs:

```
1  namespace BachelorOppgave.Data.Models
2  {
3      0 references
4      public class Module_User
5      {
6          //integers
7          0 references
8          public int moduleID { get; set; }
9          0 references
10         public int personID { get; set; }
11     }
12 }
```

Repositories i Visual Studio

i. AdminRepository.cs:

1. Lage en Admin:

```
1 reference
public int CreateAdmin(CreateAdmin createAdmin)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();
        var result = connection.Execute(@"EXEC [dbo].[CreateAdmin]
            @birthnumber,
            @personID,
            @telNr,
            @createAdmin
        ");
        return result;
    }
}
```

2. Hente Admin, det vil si hente opplysninger til alle Admins:

```
1 reference
public IEnumerable<Admin> FetchAdmin()
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();
        return connection.Query<Admin>(@"EXEC[dbo].[FetchPerson]");
    }
}
```

3. Oppdatere Admin med nye opplysninger:

```

1 reference
public int UpdateAdmin(UpdateAdmin updateAdmin)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();

        var result = connection.Execute(@"EXEC [dbo].[UpdateAdmin]
            @adminID = @adminID,
            @birthnumber = @birthnumber,
            @telNr = @telNr",
            updateAdmin
        );
        return result;
    }
}

```

4. Fjerne en Admin fra databasen og system:

```

1 reference
public int DeleteAdmin(int adminID)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();

        var result = connection.Execute(@"EXEC [dbo].[DeleteAdmin]
            @adminID = @adminID",
            new
            {
                adminID,
            }
        );
        return result;
    }
}

```

5. Hent en bestemt Admin fra systemet:

```

//fetches individual admin by using his/her 's  adminID
1 reference
public Admin FetchAdminByID(int adminID)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();

        return connection.QueryFirstOrDefault<Admin>(@"EXEC [dbo].[FetchAdminByID] @adminID = @adminID",
            new
            {
                adminID,
            });
    }
}

```

ii. LessonRepository.cs:

1. Lage en Leksjon:

```

1 reference
public int CreateLesson(CreateLesson createLesson)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();

        var result = connection.Execute(@"EXEC [dbo].[CreateLesson]
            @type = @type,
            @moduleID = @moduleID,
            @name = @name,
            @details = @details",
            createLesson

        );
        return result;
    }
}

```

2. Hente en Leksjon:

```

1 reference
public IEnumerable<Lesson> FetchLessons()
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();

        return connection.Query<Lesson>(@"EXEC [dbo].[FetchLesson]");
    }
}

```

3. Hent en bestemt Leksjon:

```

public Lesson FetchLessonByID(int lessonID)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();

        return connection.QueryFirstOrDefault<Lesson>(@"EXEC [dbo].[FetchLessonByID] @lessonID = @lessonID",
            new
            {
                lessonID,
            });
    }
}

```

4. Oppdatere en Leksjon:

```
1 reference
public int UpdateLesson(Lesson lesson)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();

        var result = connection.Execute(@"EXEC [dbo].[UpdateLesson]
            @lessonID = @lessonID,
            @type = @type,
            @moduleID = @moduleID,
            @name = @name,
            @details = @details",
            lesson
        );
        return result;
    }
}
```

5. Fjerne leksjoner ved bruk av lessonID:

```
1 reference
public int DeleteLesson(int lessonID)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();

        var result = connection.Execute(@"EXEC [dbo].[DeleteLesson]
            @lessonID = @lessonID",
            new
            {
                lessonID,
            }
        );
        return result;
    }
}
```

iii. ModuleRepository.cs:

1. Lage Modul:

```

1 reference
public int CreateModule(CreateModule createModule)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();

        var result = connection.Execute(@"exec [dbo].[CreateModule]
            @title = @title,
            @teacherID = @teacherID,
            @adminID = @adminID,
            @institution = @institution,
            @price = @price,
            @description = @description,
            @language = @language,
            @picture = @picture,
            @subject = @subject,
            @duration = @duration",
            createModule
        );
        return result;
    }
}

```

2. Hente Moduler:

```

1 reference
public IEnumerable<Module> FetchModules()
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();
        return connection.Query<Module>(@"EXEC [dbo].[FetchModule]");
    }
}

```

3. Fjerne bestemt Modul:

```

1 reference
public int DeleteModule(int moduleID)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();

        var result = connection.Execute(@"EXEC [dbo].[DeleteModule]
            @moduleID = @moduleID",
            new
            {
                moduleID,
            });
        return result;
    }
}

```

4. Oppdatere Modul:

```

1 reference
public int UpdateModule(UpdateModule updateModule)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();

        var result = connection.Execute(@"exec [dbo].[UpdateModule]
            @moduleID = @moduleID,
            @title = @title,
            @teacherID = @teacherID,
            @adminID = @adminID,
            @institution = @institution,
            @price = @price,
            @description = @description,
            @language = @language,
            @picture = @picture,
            @subject = @subject,
            @duration = @duration",
            updateModule
        );
        return result;
    }
}

```

5. Hent en bestemt Modul:

```

1 reference
public Module FetchModuleByID(int moduleID)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();

        //Dapper QueryFirstOrDefault method is used to fetch one record.
        return connection.QueryFirstOrDefault<Module>(@"EXEC [dbo].[FetchModuleByID] @moduleID = @moduleID",
            new
            {
                moduleID,
            });
    }
}

```

iv. PersonRepository.cs:

1. Lage en Person:

```

1 reference
public int CreatePerson(CreatePerson person)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();
        var result = connection.Execute(@"EXEC [dbo].[CreatePerson]
            @name = @name,
            @surname = @surname,
            @username = @username,
            @email = @email,
            @password = @password,
            @postnr = @postnr,
            @address = @address",
            person
        );
        return result;
    }
}

```

2. Hent Person liste:

```

2 references
public IEnumerable<Person> FetchPersons()
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();
        return connection.Query<Person>(@"EXEC[dbo].[FetchPerson]");
    }
}

```

3. Hent en bestemt Person:

```

1 reference
public Person FetchPersonByID(int userID)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();
        return connection.QueryFirstOrDefault<Person>(@"EXEC[dbo].[FetchPersonByID] @personID = @personID");
    }
}

```

4. Oppdatere Person tabellen:

```
1 reference
public int UpdatePerson(UpdatePerson updatePerson)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();

        var result = connection.Execute(@"EXEC [dbo].[UpdatePerson]
            @personID = @personID
            @name = @name,
            @surname = @surname,
            @username = @username,
            @email = @email,
            @password = @password,
            @postnr = @postnr,
            @address = @address",
            updatePerson
        );
        return result;
    }
}
```

5. Fjern bestemt Person:

```
1 reference
public int DeletePerson(int personID)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();

        var result = connection.Execute(@"EXEC [dbo].[DeletePerson]
            @personID = @personID",
            new
            {
                personID,
            }
        );
        return result;
    }
}
```

v. TeacherRepository.cs:

1. Lage en Lærer:

```
1 reference
public int CreateTeacher(CreateTeacher createTeacher)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();
        var result = connection.Execute(@"EXEC [dbo].[CreateTeacher]
            @title = @title,
            @institution = @institution,
            @personID = @personID,
            @image = @image",
            createTeacher
        );
        return result;
    }
}
```

2. Hent en liste med Lærer:

```
0 references
public IEnumerable<Teacher> FetchTeacher()
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();
        return connection.Query<Teacher>(@"EXEC[dbo].[FetchTeacher]");
    }
}
```

3. Hent en bestemt Lærer:

```
1 reference
public Teacher FetchTeacherByID(int teacherID)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();
        return connection.QueryFirstOrDefault<Teacher>(@"EXEC [dbo].[FetchTeacherByID] @teacherID = @teacherID",
            new
            {
                teacherID,
            });
    }
}
```

4. Oppdatere tabellen med Lærer:

```

1 reference
public int UpdateTeacher(UpdateTeacher updateTeacher)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();

        var result = connection.Execute(@"EXEC [dbo].[UpdateTeacher]
            @teacherID = @teacherID,
            @title = @title,
            @institution = @institution,
            @image = @image",
            updateTeacher
        );
        return result;
    }
}

```

5. Fjern en bestemt Teacher:

```

1 reference
public int DeleteTeacher(int teacherID)
{
    using (var connection = new SqlConnection(_connectionString))
    {
        connection.Open();

        var result = connection.Execute(@"EXEC [dbo].[DeleteTeacher]
            @teacherID = @teacherID",
            new
            {
                teacherID,
            }
        );
        return result;
    }
}

```

Hjelpemodeller i Visual Studio:

CreateAdmin og UpdateAdmin:

```

using System.Text.Json.Serialization;

namespace BachelorOppgave.Controllers.AdminModels
{
    2 references
    public class UpdateAdmin
    {
        //integers
        [JsonIgnore]
        1 reference
        public int adminID { get; set; }

        //strings
        0 references
        public string birthnumber { get; set; }
        0 references
        public string telNr { get; set; }
    }
}

1 reference
2 references
public class CreateAdmin
{
    //integers
    0 references
    public int personID { get; set; }

    //strings
    0 references
    public string birthnumber { get; set; }
    0 references
    public string telNr { get; set; }
}

```

CreateLesson og UpdateLesson:

```

using System.Text.Json.Serialization;

namespace BachelorOppgave.Controllers.LessonsModels
{
    0 references
    public class UpdateLesson
    {
        //integers
        [JsonIgnore]
        0 references
        public int lessonID { get; set; }
        [JsonIgnore]
        0 references
        public int moduleID { get; set; }

        //strings
        0 references
        public string type { get; set; }
        0 references
        public string name { get; set; }
        0 references
        public string details { get; set; }
    }
}

1   using System.Text.Json.Serialization;
2
3   namespace BachelorOppgave.Controllers.LessonsModels
4   {
5       2 references
6       public class CreateLesson
7       {
8           //integers
9           [JsonIgnore]
10          1 reference
11          public int moduleID { get; set; }

12          //strings
13          0 references
14          public string type { get; set; }
15          0 references
16          public string name { get; set; }
17          0 references
18          public string details { get; set; }
    }
}

```

CreateModule og UpdateModule:

```

UpdateModule.cs  + X
@BachelorOppgave  ✉ BachelorOppgave.Controllers  ↗ moduleID
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text.Json.Serialization;
5   using System.Threading.Tasks;
6
7   namespace BachelorOppgave.Controllers.ModulesModels
8   {
9       2 references
10      public class UpdateModule
11      {
12          //integers
13          [JsonIgnore]
14          1 reference
15          public int moduleID { get; set; }
16          0 references
17          public int teacherID { get; set; }
18          0 references
19          public int adminID { get; set; }

20          //strings
21          0 references
22          public string institution { get; set; }
23          0 references
24          public string description { get; set; }
25          0 references
26          public string language { get; set; }
27          0 references
28          public string title { get; set; }
29          0 references
30          public string picture { get; set; }
31          0 references
32          public string subject { get; set; }

33          //etc
34          0 references
35          public float price { get; set; }
36          0 references
37          public float duration { get; set; }
    }
}

CreateModule.cs  + X
@BachelorOppgave  ✉ BachelorOppgave.Controllers.ModulesModels
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Threading.Tasks;
5
6   namespace BachelorOppgave.Controllers.ModulesModels
7   {
8       2 references
9       public class CreateModule
10      {
11          //integers
12          0 references
13          public int teacherID { get; set; }
14          0 references
15          public int adminID { get; set; }

16          //strings
17          0 references
18          public string institution { get; set; }
19          0 references
20          public string description { get; set; }
21          0 references
22          public string language { get; set; }
23          0 references
24          public string title { get; set; }
25          0 references
26          public string picture { get; set; }
27          0 references
28          public string subject { get; set; }

29          //etc
30          0 references
31          public float price { get; set; }
32          0 references
33          public float duration { get; set; }
    }
}

```

CreatePerson og UpdatePerson:

```

CreatePerson.cs
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5
6  namespace BachelorOppgave.Controllers.PersonsModels
7  {
8      public class CreatePerson
9      {
10         //strings
11         public string email { get; set; }
12         public string password { get; set; }
13         public string name { get; set; }
14         public string surname { get; set; }
15         public string postnr { get; set; }
16         public string address { get; set; }
17         public string username { get; set; }
18     }
19 }
20

UpdatePerson.cs
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text.Json.Serialization;
5  using System.Threading.Tasks;
6
7  namespace BachelorOppgave.Controllers.PersonsModels
8  {
9      public class UpdatePerson
10     {
11         [JsonIgnore]
12         public int personID { get; set; }
13
14         public string email { get; set; }
15         public string password { get; set; }
16         public string name { get; set; }
17         public string surname { get; set; }
18         public string postnr { get; set; }
19         public string address { get; set; }
20         public string username { get; set; }
21     }
22 }
23

```

CreateTeacher og UpdateTeacher:

```

CreateTeacher.cs
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5
6  namespace BachelorOppgave.Controllers.TeacherModels
7  {
8      public class CreateTeacher
9      {
10         //integers
11         public int personID { get; set; }
12
13         //strings
14         public string title { get; set; }
15         public string institution { get; set; }
16         public string image { get; set; } //save as images url or file directory
17     }
18 }
19

UpdateTeacher.cs
1  using System.Text.Json.Serialization;
2
3  namespace BachelorOppgave.Controllers.TeacherModels
4  {
5      public class UpdateTeacher
6      {
7          //integers
8          [JsonIgnore]
9          public int teacherID { get; set; }
10
11          //strings
12          public string title { get; set; }
13          public string institution { get; set; }
14          public string image { get; set; } //save as images url or file directory
15      }
16 }
17

```

Controllers i Visual Studio:

For Admin:

```
namespace BachelorOppgave.Controllers
{
    [Route("api/[controller]")] //api/admin
    [ApiController]
    [Authorize(Roles = "Admin")]
    [EnableCors("CORS")]
    1 reference
    public class AdminController : Controller
    {
        private readonly AdminRepository _adminRepository;

        0 references
        public AdminController(AdminRepository adminRepository)
        {
            _adminRepository = adminRepository;
        }

        [HttpGet] //api/admin
        //to get a list of admins
        0 references
        public IEnumerable<Admin> GetAdmins()
        {
            return _adminRepository.FetchAdmin();
        }

        //to get one specific admin
        [HttpGet("{adminID}")]
        0 references
        public IActionResult FetchAdminByID(int adminID)
        {
            var admin = _adminRepository.FetchAdminByID(adminID);

            if (admin != null)
            {
                return Ok(admin);
            }

            return NotFound(new { Message = $"Module with id {adminID} is not available." });
        }
    }
}
```

```

//to create admin
[HttpPost]
0 references
public IActionResult CreateAdmin(CreateAdmin createAdmin)
{
    var result = _adminRepository.CreateAdmin(createAdmin);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

//to update admin
[HttpPut("{adminID}")]
0 references
public IActionResult UpdateAdmin(int adminID, UpdateAdmin updateAdmin)
{
    updateAdmin.adminID = adminID;
    var result = _adminRepository.UpdateAdmin(updateAdmin);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

//to delete admin
[HttpDelete("{adminID}")]
0 references
public IActionResult DeleteAdmin(int adminID)
{
    var result = _adminRepository.DeleteAdmin(adminID);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

```

For Lesson:

```
namespace BachelorOppgave.Controllers
{
    [Route("api/module/{moduleID}/{controller}")] //api/module/{moduleID}/Lessons
    [ApiController]
    [Authorize]
    [EnableCors("CORS")]
    1 reference
    public class LessonsController : Controller
    {
        private readonly LessonRepository _lessonRepository;

        0 references
        public LessonsController(LessonRepository lessonRepository)
        {
            _lessonRepository = lessonRepository;
        }
        [HttpGet] //api/lessons
        [AllowAnonymous]
        //to get a list of lessons
        0 references
        public IActionResult GetLessons()
        {
            return Ok(_lessonRepository.FetchLessons());
        }

        //to get one specific lesson
        [HttpGet("{lessonID}")]
        [AllowAnonymous]
        0 references
        public IActionResult FetchLessonByID(int moduleID, int lessonID)
        {
            var lesson = _lessonRepository.FetchLessonByID(lessonID);

            if (lesson != null)
            {
                return Ok(lesson);
            }

            return NotFound(new { Message = $"Lesson with id {lessonID} is not available." });
        }
    }
}
```

```

//to create lesson
[HttpPost]
[Authorize(Roles = "Admin")]
0 references
public IActionResult CreateLesson(int moduleID, CreateLesson createLesson)
{
    createLesson.moduleID = moduleID;
    var result = _lessonRepository.CreateLesson(createLesson);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

//to update lesson
[HttpPut("{lessonID}")]
[Authorize(Roles = "Admin")]
0 references
public IActionResult UpdateLesson(int moduleID, int lessonID, Lesson lesson)
{
    lesson.moduleID = moduleID;
    lesson.lessonID = lessonID;
    var result = _lessonRepository.UpdateLesson(lesson);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

//to delete lesson
[HttpDelete("{lessonID}")]
[Authorize(Roles = "Admin")]
0 references
public IActionResult DeleteLesson(int moduleID, int lessonID)
{
    var result = _lessonRepository.DeleteLesson(lessonID);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

```

For Modules:

```
namespace BachelorOppgave.Controllers
{
    [Route("api/[controller]")] //api/modules
    [ApiController]
    [Authorize]
    [EnableCors("CORS")]
    1 reference
    public class ModulesController : Controller
    {
        private readonly ModuleRepository _moduleRepository;

        0 references
        public ModulesController(ModuleRepository moduleRepository)
        {
            _moduleRepository = moduleRepository;
        }

        [HttpGet] //api/modules
        //to get a list of modules
        [AllowAnonymous]
        0 references
        public IEnumerable<Module> GetModules()
        {
            return _moduleRepository.FetchModules();
        }

        //to get one specific module
        [HttpGet("{moduleID}")]
        [AllowAnonymous]
        0 references
        public IActionResult FetchModuleByID(int moduleID)
        {
            var module = _moduleRepository.FetchModuleByID(moduleID);

            if (module != null)
            {
                return Ok(module);
            }

            return NotFound(new { Message = $"Module with id {moduleID} is not available." });
        }
    }
}
```

```

//to create module
[HttpPost]
[Authorize(Roles = "Admin")]
0 references
public IActionResult CreateModule(CreateModule createModule)
{
    var result = _moduleRepository.CreateModule(createModule);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

//to update module
[HttpPut("{moduleID}")]
[Authorize(Roles = "Admin")]
0 references
public IActionResult UpdateModule(int moduleID, UpdateModule updateModule)
{
    updateModule.moduleID = moduleID;
    var result = _moduleRepository.UpdateModule(updateModule);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

//to delete module
[HttpDelete("{moduleID}")]
[Authorize(Roles = "Admin")]
0 references
public IActionResult DeleteModule(int moduleID)
{
    var result = _moduleRepository.DeleteModule(moduleID);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

```

For Persons:

```
namespace BachelorOppgave.Controllers

{
    [Route("api/[controller]")]
    [ApiController]
    [Authorize]
    [EnableCors("CORS")]
    1 reference
    public class PersonsController : ControllerBase
    {
        private readonly PersonRepository _personRepository;
        private readonly PersonService _personService;

        0 references
        public PersonsController(PersonRepository personRepository, PersonService personService)
        {
            _personRepository = personRepository;
            _personService = personService;
        }

        //to get list of persons
        [HttpGet]
        0 references
        public IActionResult GetUsers()
        {
            return Ok(_personRepository.FetchPersons());
        }

        //to get one specific person
        [HttpGet("{personID}")]
        0 references
        public IActionResult FetchPersonByID(int personID)
        {
            var person = _personRepository.FetchPersonByID(personID);

            if (person != null)
            {
                return Ok(person);
            }

            return NotFound(new { Message = $"Person with id {personID} is not available." });
        }
    }
}
```

```

//to create person
[AllowAnonymous]
[HttpPost]
0 references
public IActionResult CreatePerson(CreatePerson person)
{
    var result = _personRepository.CreatePerson(person);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

//to update person
[HttpPut("{personID}")]
0 references
public IActionResult UpdatePerson(int personID, UpdatePerson updatePerson)
{
    updatePerson.personID = personID;
    var result = _personRepository.UpdatePerson(updatePerson);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

//to delete person
[HttpDelete("{personID}")]
0 references
public IActionResult DeletePerson(int personID)
{
    var result = _personRepository.DeletePerson(personID);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

//to authenticate person
[AllowAnonymous]
[HttpPost("authenticate")]
0 references
public IActionResult Authenticate([FromBody]AuthenticateModel model)
{
    var person = _personService.Authenticate(model.Username, model.Password);

    if (person == null)
        return Unauthorized(new { message = "Username or password is incorrect" });

    return Ok(person);
}

```

For Teachers:

```
namespace BachelorOppgave.Controllers
{
    [Route("api/[controller]")] //api/teacher
    [ApiController]
    [Authorize]
    [EnableCors("CORS")]
    1 reference
    public class TeacherController : Controller
    {
        private readonly TeacherRepository _teacherRepository;

        0 references
        public TeacherController(TeacherRepository teacherRepository)
        {
            _teacherRepository = teacherRepository;
        }

        [HttpGet] //api/teachers
        [AllowAnonymous]
        //to get a list of teachers
        0 references
        public IEnumerable<Teacher> GetTeacher()
        {
            return _teacherRepository.FetchTeacher();
        }

        //to get one specific teacher
        [HttpGet("{teacherID}")]
        [AllowAnonymous]
        0 references
        public IActionResult FetchTeacherByID(int teacherID)
        {
            var teacher = _teacherRepository.FetchTeacherByID(teacherID);

            if (teacher != null)
            {
                return Ok(teacher);
            }

            return NotFound(new { Message = $"Teacher with id {teacherID} is not available." });
        }
    }
}
```

```

[HttpPost]
[Authorize(Roles = "Admin")]
0 references
public IActionResult CreateTeacher(CreateTeacher createTeacher)
{
    var result = _teacherRepository.CreateTeacher(createTeacher);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

//to update teacher
[HttpPut("{teacherID}")]
0 references
public IActionResult UpdateTeacher(int teacherID, UpdateTeacher updateTeacher)
{
    //Admin can change every teacher, but teacher can only modify their own data.
    //Run check, in case you are not admin, to see if we own the teacher item we try to modify.
    if(!base.User.IsInRole("Admin"))
    {
        //Get personId from claims:
        var personId = base.User.Claims
            .Single(claim => claim.Type == ClaimTypes.Name)
            .Value;

        //Check the teacher in database that we want to change
        var teacherItem = _teacherRepository.FetchTeacherByID(teacherID);

        //If teacher is not the logged in person, then we are not allowed to modify the teacher item we don't own.
        if (teacherItem?.personID.ToString() != personId)
        {
            throw new UnauthorizedAccessException();
        }
    }

    updateTeacher.teacherID = teacherID;
    var result = _teacherRepository.UpdateTeacher(updateTeacher);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

//to delete teacher
[HttpDelete("{teacherID}")]
[Authorize(Roles = "Admin")]
0 references
public IActionResult DeleteTeacher(int teacherID)
{
    var result = _teacherRepository.DeleteTeacher(teacherID);
    if (result > 0)
    {
        return Ok(result);
    }

    return StatusCode(500, new { Message = "Some error happened." });
}

```

Stegene for publisering av API i Azure

I Azure portal – portal.azure.com – valgte vi «App Services», som vist bildet under:

portal.azure.com/#home

Azure Search resources, services, and docs (G+/)

Azure services

Create a resource

SQL databases

App Services

SQL managed instances

Managed databases

A regist

Recent resources

Name	Type
WixDatabase (wix/WixDatabase)	SQL database

Så kommer vi til denne siden og klikket på Create app service:

No app services to display

Create, build, deploy, and manage powerful web, mobile, and API apps for employees or customers using a single platform.

Build standards-based web apps and APIs using .NET, Java, Node.js, PHP, and Python.

Learn more about App Service ↗

Create app service

I Prosjekt detaljer, kalte vi API-en "WixAPI. Vi valgte «Free Trial» og satt Resource Group lik «WixDatabase», som er databasen vi laget i SQL server og hosted hos Azure. Når vi skulle velge område (Region), vi kunne ikke forandre fra Central US uten å måtte betale. Det vil si at serveren ligger i USA.

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

▼

Resource Group * ⓘ

▼

[Create new](#)

Instance Details

Name * ✓
.azurewebsites.net

Publish * Code Docker Container

Runtime stack * ▼

Operating System * Linux Windows

Region * ▼

ⓘ Not finding your App Service Plan? Try a different region.

I App Service Plan > Sku and Size, valgte vi Free F1, med shared infrastructure og 1 GB minne, fordi den var gratis.

App Service Plan

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app.
[Learn more](#) ↗

Windows Plan (Central US) * ⓘ ▼

[Create new](#)

Sku and size * Free F1
Shared infrastructure, 1 GB memory
[Change size](#)

I WebApp > Monitoring, valgte vi ja til «enable application insights», valgte vi “yes”, fordi det er et nyttig verktøy for å holde oversikt over, blant annet, hvor mange som bruker appen, noe som prosjekt eieren hadde lyst til.

Web App

[Basics](#) [Monitoring](#) [Tags](#) [Review + create](#)

Azure Monitor gives you full observability into your applications, infrastructure, and network. [Learn more](#) ↗

Application Insights

Enable Application Insights * No Yes

Application Insights * ▼

[Create new](#)

Region

I slutten, klikker vi på “Review + create” og så “Create” igjen. Så blir vi sendt til denne siden:

✓ Your deployment is complete

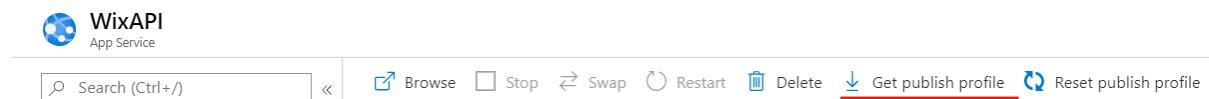
Deployment name: Microsoft.Web-WebApp-Portal-24adc536-bf76
Subscription: Free Trial
Resource group: WixDatabase

Start time: 5/3/2020, 4:47:49 PM
Correlation ID: e916d3e8-59c3-4e16-b5a3-7277a086a651

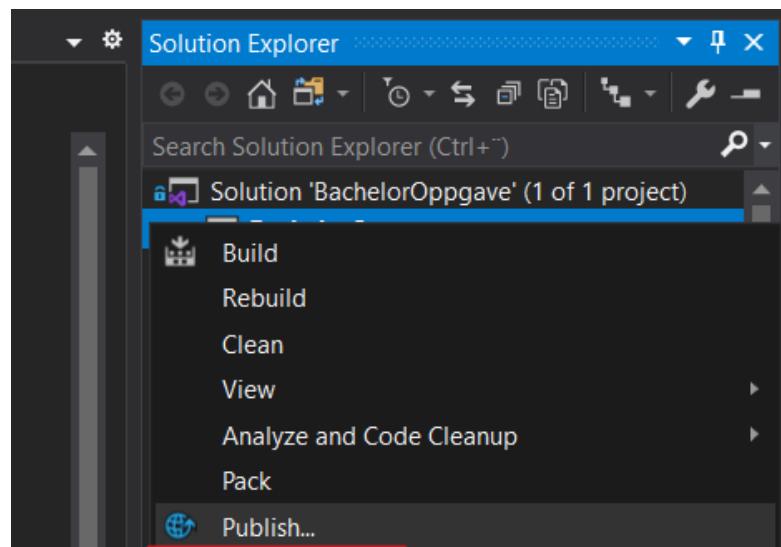
▼ Deployment details [\(Download\)](#)
^ Next steps

[Go to resource](#)

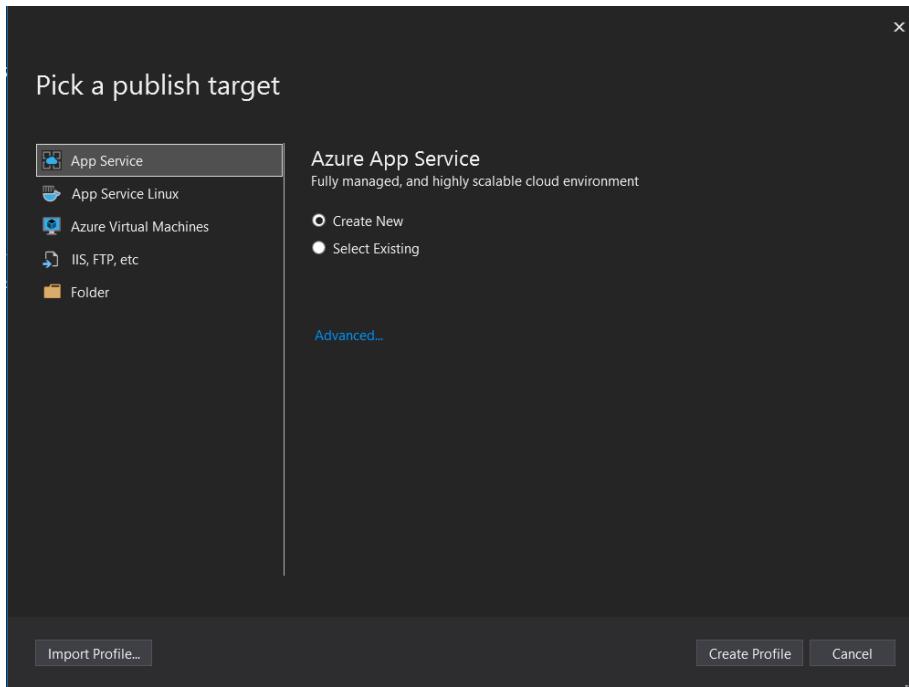
Vi klikker i "Go to resource" og ender opp i første siden til WixAPI. Når vi først er på WixAPI, så klikker vi i "Get publish profile":



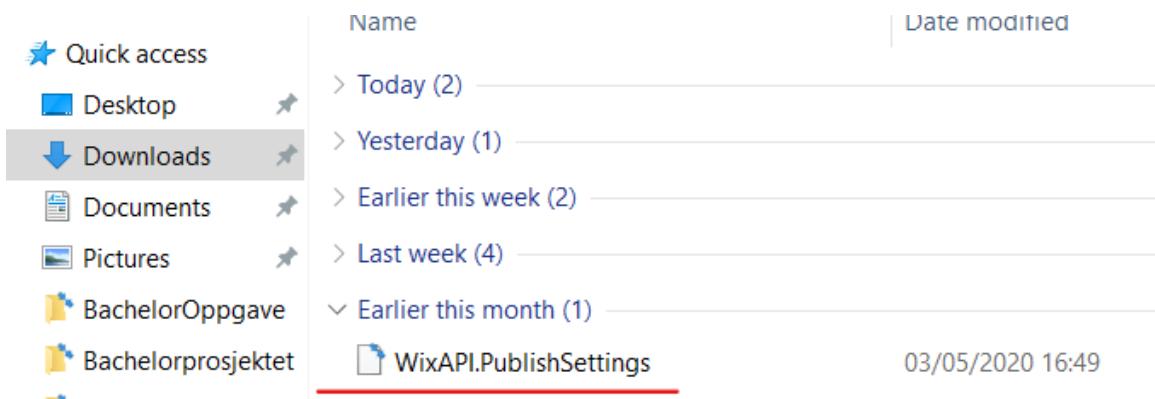
Etter å ha gjennomført de stegene i Azure, så måtte vi fikse det også i Visual Studio. Vi høyreklikker på prosjektnavnet i Solution Explorer og klikker på «publish».



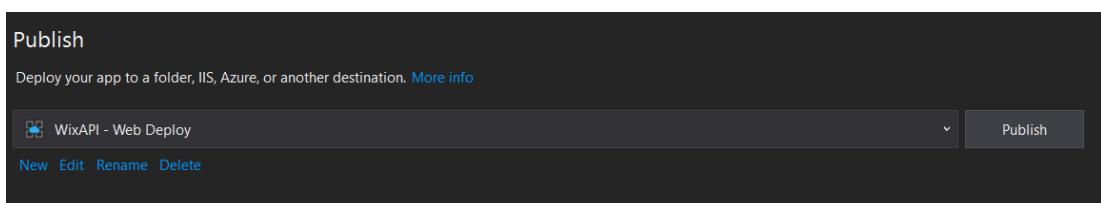
Deretter klikker vi på "import profile", nederst på venstre hjørnet.



Vi velger filen som ble lastet ned fra Azure:



Så klikker vi på "Publish":



Hvis det ikke dukker opp noen feilmeldinger i koden, så skal API-en være publisert og er tilgjengelig med bruk av linken <http://wixapi.azurewebsites.net>.

Testing av autentisering i Postman og Swagger:

For å teste funksjonaliteten, så brukte vi Postman for å lage en bruker med brukernavn og passord. For å logge oss inn, må vi bruke HTTP-metode POST mot /api/Persons/authenticate, gå til «Body», velge «raw» og så JSON format, for å legge til brukernavn og passord, slik som i figuren under:

The screenshot shows the Postman interface with a single request named "Untitled Request". The method is set to POST, and the URL is https://localhost:5001/api/Persons/authenticate. The "Body" tab is selected, showing a JSON payload:

```

1 {
2   "username": "silje",
3   "password": "asd123"
4 }

```

Postman returnerer brukeren med et token:

The screenshot shows the Postman interface after sending the POST request. The response status is 200 OK, and the response body is displayed in JSON format:

```

1 {
2   "personID": 2,
3   "email": "dfgfdgd",
4   "name": "hdfhdh",
5   "surname": "gfhgfh",
6   "postnr": "shdf",
7   "address": "fdhdhf",
8   "username": "silje",
9   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmJxdWVfbmFtZSI6IjIiLCJuYmYjOjE1ODk1NTA4ODUsImV4cCI6MTU5MDE1NTY4NSviaWF0IjoxNTg5NTUwODg1fQ.jQkKEWCj2o_B7j7qL2BQo81np60QgN_E8xUvtW13DEI",
10  "adminID": 5,
11  "teacherID": 5
12

```

Dette token-et skal brukes for å teste om man er nå autorisert til å få tilgang til de andre funksjonalitetene i applikasjonen. For eksempel, for å hente alle personene, så bruker vi GET mot /api/Persons/, velger vi «Authorization», type «Bearer Token» og så token-et som vi fikk når brukeren ble autentisert. Denne testen ble gjort før metoden GetPersons ble satt til [AllowAnonymous].

The screenshot shows the Postman interface with a GET request to https://localhost:5001/api/Persons/. The "Authorization" tab is selected, and the type is set to "Bearer Token". A token value is entered in the "Token" field:

TYPE
Bearer Token

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Token
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmJxdWVfbmFtZSI6IjIiLCJuYmYjOjE1ODk1NTA4ODUsImV4cCI6MTU5MDE1NTY4NSviaWF0IjoxNTg5NTUwODg1fQ.jQkKEWCj2o_B7j7qL2BQo81np60QgN_E8xUvtW13DEI

Og applikasjonen bekrefter at vi har tillatelse til å se liste med brukerne:

Body Cookies Headers (4) Test Results Status: 200 OK Time: 249 ms Size: 655 B

Pretty Raw Preview Visualize JSON

```

1  [
2   {
3     "personID": 2,
4     "email": "dfgfdgd",
5     "name": "hdfhdf",
6     "surname": "gfhgfh",
7     "postnr": "shdf",
8     "address": "fdhdfh",
9     "username": "silje",
10    "token": null,
11    "adminID": 5,
12    "teacherID": 5
13  },
14  {
15    "personID": 3,
16    "email": "email@email.com",
17    "name": "string",
18    "surname": "string",
19    "postnr": "stri",
20    "address": "string",
21    "username": "usertest",
22    "token": null,
23    "adminID": null,
24    "teacherID": null
25  },
26  {
27    "personID": 4,
28    "email": "string",
29    "name": "string",
30    "surname": "string",
31    "postnr": "string",
32    "address": "string",
33    "username": "string"
34  }
]

```

Testing i Swagger:

For å teste om autentisering løsningen har fungert i Swagger, først laget vi en bruker.

POST /api/Persons

Parameters Cancel

No parameters

Request body application/Json

```
{
  "email": "string",
  "password": "string",
  "name": "string",
  "surname": "string",
  "postnr": "string",
  "address": "string",
  "username": "string"
}
```

Execute Clear

Og så, i metoden authenticate, tester vi funksjonaliteten med å legge til brukernavnet og passord:

POST /api/Persons/authenticate

Parameters

No parameters

Request body

application/json

```
{
  "username": "string",
  "password": "string"
}
```

Execute Clear

Og vi får svar fra serveren med token:

Responses

Curl

```
curl -X POST "https://wixapi.azurewebsites.net/api/Persons/authenticate" -H "accept: /*" -H "Content-Type: application/json" -d "{\"username\":\"string\",\"password\":\"string\"}"
```

Request URL

<https://wixapi.azurewebsites.net/api/Persons/authenticate>

Server response

Code Details

200 Response body

```
{
  "personID": 4,
  "email": "string",
  "name": "string",
  "surname": "string",
  "postnr": "string",
  "address": "string",
  "username": "string",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbmxwdWVmFtZSI6IjQlLCJuYmY1OjE1ODk1MzgzNjAsImV4cCI6MTU5MDMzMzE2MCviaWF0IjoxNTg5NTMzMzYwFQ.gkIZ59pZYq-vQkkVNEYkTRUzdhjKRLOWPkoIU1eEAo0",
  "adminID": null,
  "teacherID": null
}
```

Download

Response headers

```
content-encoding: gzip
content-type: application/json; charset=utf-8
date: Fri, 15 May 2020 10:25:59 GMT
server: Microsoft-IIS/10.0
transfer-encoding: chunked
vary: Accept-Encoding
x-powered-by: ASP.NET
```

Download

6.9 GitHub

Frontend

Search or jump to... Pull requests Issues Marketplace Explore

Unwatch 3 Star 0 Fork 1

Code Issues 0 Pull requests 0 Actions Projects 0 Security 0 Insights Settings

Branch: dev

Commits on May 15, 2020

- updateModul
alinkaZ committed 7 days ago
- Merge branch 'master' into dev
MarijaShakhrai committed 7 days ago
- DeleteModul
alinkaZ committed 7 days ago

Commits on May 14, 2020

- api
alinkaZ committed 8 days ago

Commits on May 13, 2020

- authorization
alinkaZ committed 9 days ago

Commits on May 10, 2020

- Some changes in UI for users pages
alinkaZ committed 12 days ago

Commits on May 4, 2020

- Add question on Admin Quiz Page
alinkaZ committed 18 days ago

Commits on May 3, 2020

- Form for adding answer on the quiz admin page
alinkaZ committed 19 days ago

Commits on May 1, 2020

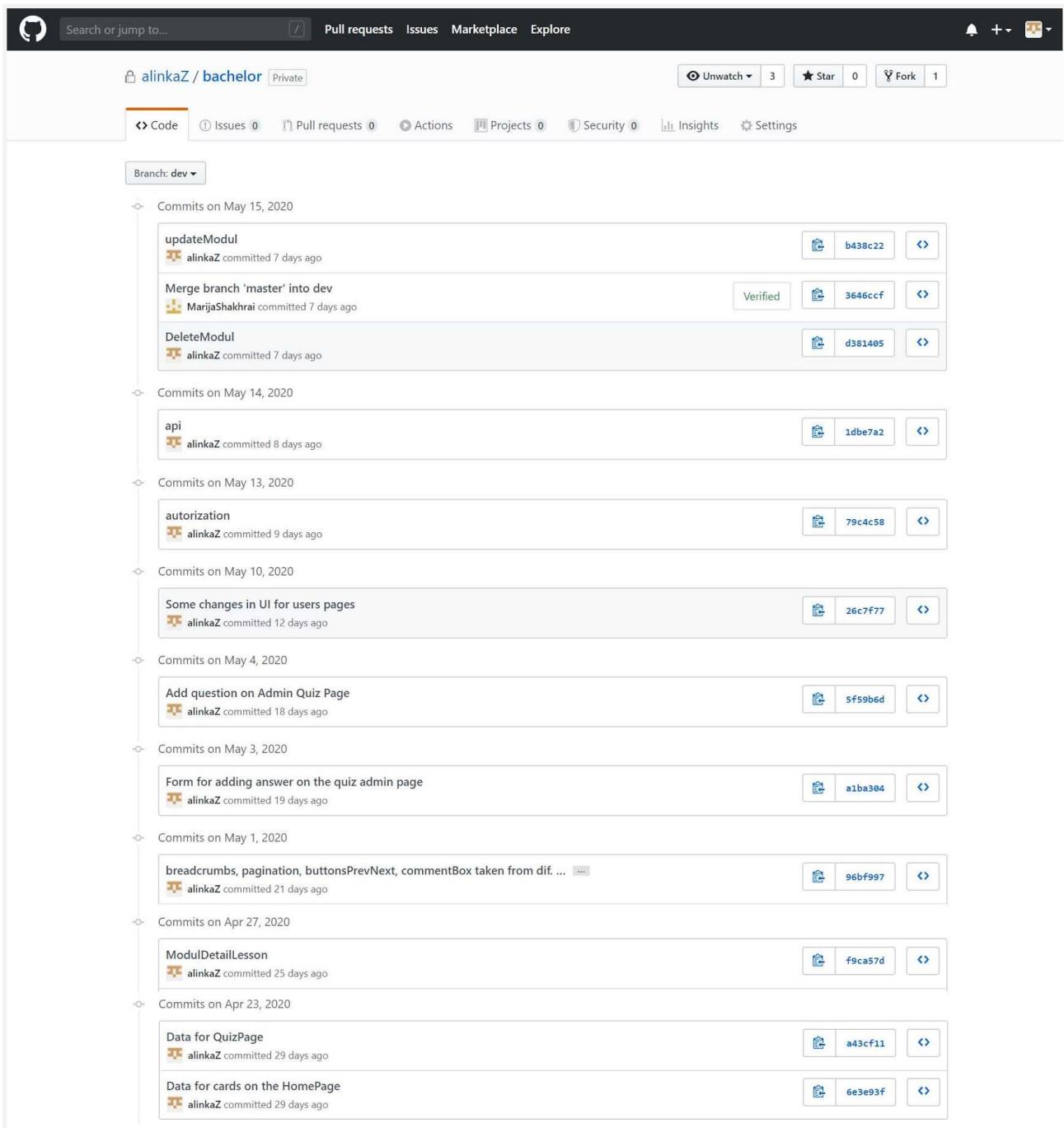
- breadcrumbs, pagination, buttonsPrevNext, commentBox taken from dif. ...
alinkaZ committed 21 days ago

Commits on Apr 27, 2020

- ModulDetailLesson
alinkaZ committed 25 days ago

Commits on Apr 23, 2020

- Data for QuizPage
alinkaZ committed 29 days ago
- Data for cards on the HomePage
alinkaZ committed 29 days ago



Commits on Apr 22, 2020
Json for CoursePage  alinkaZ committed on Apr 22
Commits on Apr 18, 2020
Info icon on the quiz page  alinkaZ committed on Apr 18
Commits on Apr 17, 2020
from bachelor2  alinkaZ committed on Apr 17
Commits on Apr 16, 2020
clear project2  alinkaZ committed on Apr 16
Commits on Apr 15, 2020
Merge pull request #2 from alinkaZ/dev ...  alinkaZ committed on Apr 15
AdminMainModulPage, TextPage  alinkaZ committed on Apr 15
Commits on Apr 11, 2020
Merge pull request #1 from alinkaZ/Database ...  alinkaZ committed on Apr 11
Merge branch 'master' of https://github.com/alinkaZ/bachelor  alinkaZ committed on Apr 11
Admin Home and Main Modul Page  alinkaZ committed on Apr 11
Commits on Apr 10, 2020
Login frontend funct  silb90 committed on Apr 10
Commits on Apr 9, 2020
Structure reorganising  alinkaZ committed on Apr 9
Commits on Mar 10, 2020
Commit from Silje  silb90 committed on Mar 10
second commit  alinkaZ committed on Mar 10
Commits on Mar 4, 2020
First commit  alinkaZ committed on Mar 4
first commit  alinkaZ committed on Mar 4

Search or jump to... Pull requests Issues Marketplace Explore

Unwatch 3 Star 0 Fork 1

Code Issues 0 Pull requests 0 Actions Projects 0 Security 0 Insights Settings

Branch: master

- Commits on May 15, 2020
 - DeleteModul
alinkaZ committed 7 days ago
- Commits on May 14, 2020
 - api
alinkaZ committed 8 days ago
- Commits on May 13, 2020
 - authorization
alinkaZ committed 9 days ago
- Commits on May 10, 2020
 - Some changes in UI for users pages
alinkaZ committed 12 days ago
- Commits on May 4, 2020
 - Add question on Admin Quiz Page
alinkaZ committed 18 days ago
- Commits on May 3, 2020
 - Form for adding answer on the quiz admin page
alinkaZ committed 19 days ago
- Commits on May 1, 2020
 - breadcrumbs, pagination, buttonsPrevNext, commentBox taken from dif. ...
alinkaZ committed 21 days ago
- Commits on Apr 27, 2020
 - ModulDetailLesson
alinkaZ committed 25 days ago
- Commits on Apr 23, 2020
 - Data for QuizPage
alinkaZ committed 29 days ago
 - Data for cards on the HomePage
alinkaZ committed 29 days ago
- Commits on Apr 22, 2020
 - Json for CoursePage
alinkaZ committed on Apr 22
- Commits on Apr 18, 2020
 - Info icon on the quize page
alinkaZ committed on Apr 18
- Commits on Apr 17, 2020
 - from bachelor2
alinkaZ committed on Apr 17
- Commits on Apr 16, 2020
 - clear project2
alinkaZ committed on Apr 16

Commits on Apr 15, 2020
Merge pull request #2 from alinkaZ/dev ... alinkaZ committed on Apr 15
AdminMainModulPage, TextPage alinkaZ committed on Apr 15
Commits on Apr 11, 2020
Merge pull request #1 from alinkaZ/Database ... alinkaZ committed on Apr 11
Merge branch 'master' of https://github.com/alinkaZ/bachelor alinkaZ committed on Apr 11
Admin Home and Main Modul Page alinkaZ committed on Apr 11
Commits on Apr 10, 2020
Login frontend funct silb90 committed on Apr 10
Commits on Apr 9, 2020
Structure reorganising alinkaZ committed on Apr 9
Seeded values with new migrations AlexMess1996 committed on Apr 9
Commits on Apr 8, 2020
Added database WIX.Db AlexMess1996 committed on Apr 8
Commits on Mar 24, 2020
Created wix.db AlexMess1996 committed on Mar 24
Added more to dotnet core AlexMess1996 committed on Mar 24
Establishing dotnet in project AlexMess1996 committed on Mar 24
Commits on Mar 11, 2020
Revert "Laging av api folder som skal lages av backend team" ... AlexMess1996 committed on Mar 11
Laging av api folder som skal lages av backend team AlexMess1996 committed on Mar 11
Commits on Mar 10, 2020
Commit from Silje silb90 committed on Mar 10
second commit alinkaZ committed on Mar 10
Commits on Mar 4, 2020
First commit alinkaZ committed on Mar 4
first commit alinkaZ committed on Mar 4

Backend

- Commits on May 21, 2020
 - FetchPerson to FetchAdmin**
AlexMess1996 committed yesterday
- Commits on May 20, 2020
 - deleting unused things**
silb90 committed 2 days ago
- Commits on May 19, 2020
 - deleting useless stuff**
silb90 committed 3 days ago
 - adding code to modules**
silb90 committed 3 days ago
- Commits on May 18, 2020
 - deleted unnecessary folders**
silb90 committed 4 days ago
 - get all teachers**
silb90 committed 4 days ago
- Commits on May 16, 2020
 - created models for all controllers**
silb90 committed 6 days ago
 - Merge branch 'master' into feature/roles-as-claims**
silb90 committed 6 days ago
 - Added roles**
silb90 committed 6 days ago
- Commits on May 15, 2020
 - Modified modules**
silb90 committed 7 days ago
 - Create PersonsModel and modifications in CreatePerson**
silb90 committed 7 days ago
- Commits on May 14, 2020
 - Small fix in Teacher Repository**
silb90 committed 8 days ago
- Commits on May 12, 2020

-o	Commits on May 12, 2020																
	<table border="1"> <tbody> <tr> <td>Added roles.</td> <td> silb90 committed 10 days ago</td> <td> df987e5</td> <td></td> </tr> <tr> <td>fixing cors issue</td> <td> silb90 committed 10 days ago</td> <td> 7d18481</td> <td></td> </tr> <tr> <td>Added auth</td> <td> silb90 committed 10 days ago</td> <td> 888474f</td> <td></td> </tr> <tr> <td>Revert IdentityServer</td> <td> silb90 committed 11 days ago</td> <td> a3e6526</td> <td></td> </tr> </tbody> </table>	Added roles.	 silb90 committed 10 days ago	 df987e5		fixing cors issue	 silb90 committed 10 days ago	 7d18481		Added auth	 silb90 committed 10 days ago	 888474f		Revert IdentityServer	 silb90 committed 11 days ago	 a3e6526	
Added roles.	 silb90 committed 10 days ago	 df987e5															
fixing cors issue	 silb90 committed 10 days ago	 7d18481															
Added auth	 silb90 committed 10 days ago	 888474f															
Revert IdentityServer	 silb90 committed 11 days ago	 a3e6526															
-o	Commits on May 11, 2020																
	<table border="1"> <tbody> <tr> <td>Revert "Added frontend files for testing purposes + small additions t..."</td> <td> silb90 committed 11 days ago</td> <td> ef02c0c</td> <td></td> </tr> <tr> <td>Merge branch 'AddCors'</td> <td> silb90 committed 11 days ago</td> <td> 99aae10</td> <td></td> </tr> <tr> <td>Revert "EF migration + login functionality"</td> <td> silb90 committed 11 days ago</td> <td> 22e7448</td> <td></td> </tr> <tr> <td>add cors</td> <td> silb90 committed 11 days ago</td> <td> 5390b1d</td> <td></td> </tr> </tbody> </table>	Revert "Added frontend files for testing purposes + small additions t..."	 silb90 committed 11 days ago	 ef02c0c		Merge branch 'AddCors'	 silb90 committed 11 days ago	 99aae10		Revert "EF migration + login functionality"	 silb90 committed 11 days ago	 22e7448		add cors	 silb90 committed 11 days ago	 5390b1d	
Revert "Added frontend files for testing purposes + small additions t..."	 silb90 committed 11 days ago	 ef02c0c															
Merge branch 'AddCors'	 silb90 committed 11 days ago	 99aae10															
Revert "EF migration + login functionality"	 silb90 committed 11 days ago	 22e7448															
add cors	 silb90 committed 11 days ago	 5390b1d															
-o	Commits on May 10, 2020																
	<table border="1"> <tbody> <tr> <td>EF migration + login functionality</td> <td> silb90 committed 12 days ago</td> <td> 65a1df8</td> <td></td> </tr> </tbody> </table>	EF migration + login functionality	 silb90 committed 12 days ago	 65a1df8													
EF migration + login functionality	 silb90 committed 12 days ago	 65a1df8															
-o	Commits on May 9, 2020																
	<table border="1"> <tbody> <tr> <td>Added frontend files for testing purposes + small additions to backend</td> <td> silb90 committed 13 days ago</td> <td> ecdfe015</td> <td></td> </tr> </tbody> </table>	Added frontend files for testing purposes + small additions to backend	 silb90 committed 13 days ago	 ecdfe015													
Added frontend files for testing purposes + small additions to backend	 silb90 committed 13 days ago	 ecdfe015															
-o	Commits on May 7, 2020																
	<table border="1"> <tbody> <tr> <td>Trying to implement authentication and authorization</td> <td> silb90 committed 15 days ago</td> <td> e236795</td> <td></td> </tr> </tbody> </table>	Trying to implement authentication and authorization	 silb90 committed 15 days ago	 e236795													
Trying to implement authentication and authorization	 silb90 committed 15 days ago	 e236795															
-o	Commits on May 4, 2020																
	<table border="1"> <tbody> <tr> <td>Updating database and API for testing in swagger</td> <td> silb90 committed 18 days ago</td> <td> 38af380</td> <td></td> </tr> <tr> <td>Added controller and repository for admin and teacher</td> <td> silb90 committed 18 days ago</td> <td> 4323fa8</td> <td></td> </tr> <tr> <td>Small fix module repository; update controller and repository for les...</td> <td> silb90 committed 18 days ago</td> <td> d62e96f</td> <td></td> </tr> </tbody> </table>	Updating database and API for testing in swagger	 silb90 committed 18 days ago	 38af380		Added controller and repository for admin and teacher	 silb90 committed 18 days ago	 4323fa8		Small fix module repository; update controller and repository for les...	 silb90 committed 18 days ago	 d62e96f					
Updating database and API for testing in swagger	 silb90 committed 18 days ago	 38af380															
Added controller and repository for admin and teacher	 silb90 committed 18 days ago	 4323fa8															
Small fix module repository; update controller and repository for les...	 silb90 committed 18 days ago	 d62e96f															

Commits on May 4, 2020
Updating database and API for testing in swagger silb90 committed 18 days ago
Added controller and repository for admin and teacher silb90 committed 18 days ago
Small fix module repository; update controller and repository for les... silb90 committed 18 days ago
Updating controllers and repositories for Modules and Persons silb90 committed 18 days ago
Delete weatherforecast.cs silb90 committed 19 days ago
Commits on May 3, 2020
Added swagger, deleted datarepository, modified code silb90 committed 19 days ago
Inserted data to be able to publish the API in azure silb90 committed 19 days ago
Commits on May 2, 2020
Added methods that will be utilized on storing data to the database AlexMess1996 committed 20 days ago
Commits on May 1, 2020
Re- implement the diagram AlexMess1996 committed 21 days ago
Created all available models from SQL SMS AlexMess1996 committed 21 days ago
Commits on Apr 30, 2020
small fix in data repository silb90 committed 22 days ago
Updating tables with new attributes silb90 committed 22 days ago

-o	Commits on Apr 30, 2020
	Update datarepository  silb90 committed 22 days ago Modified course to module and added lesson ...  silb90 committed 23 days ago Revert "Delete WeatherForecastController.cs" ...  silb90 committed 23 days ago
-o	Commits on Apr 22, 2020
	Merge pull request #1 from silb90/Database_Diagram_Implement ...  AlexMess1996 committed on 22 Apr Delete WeatherForecastController.cs  AlexMess1996 committed on 22 Apr
-o	Commits on Apr 20, 2020
	changing connection strings to azure  silb90 committed on 20 Apr Update course/user  silb90 committed on 20 Apr
-o	Commits on Apr 19, 2020
	Delete user/course  silb90 committed on 19 Apr Create course  silb90 committed on 19 Apr Create users  silb90 committed on 19 Apr
-o	Commits on Apr 18, 2020
	fetchcourseid, fetchuserid  silb90 committed on 18 Apr
-o	Commits on Apr 16, 2020
	implementing restful api  silb90 committed on 16 Apr Added update and delete user  silb90 committed on 16 Apr small fix

Commits on Apr 16, 2020

implementing restful api silb90 committed on 16 Apr	fb0b1f3	🔗
Added update and delete user silb90 committed on 16 Apr	16897ed	🔗
small fix silb90 committed on 16 Apr	16bfd24	🔗
small fix silb90 committed on 16 Apr	a30b397	🔗
add separate controller user/course silb90 committed on 16 Apr	bb5821d	🔗
update + delete kurs silb90 committed on 16 Apr	87cb04b	🔗
to be continued 2... silb90 committed on 16 Apr	70ecdf1	🔗
To be continued... silb90 committed on 16 Apr	500eefd	🔗

Commits on Apr 14, 2020

vs commit silb90 committed on 14 Apr	d41caa1	🔗
First commit silb90 committed on 14 Apr	deb50dc	🔗
first commit silb90 committed on 14 Apr	b6d2ebf	🔗

Commits on Mar 17, 2020

lagt til database for CRUD DESKTOP-4NCN8H1\Amina committed on 17 Mar	91295a3	🔗
Initial Commit DESKTOP-4NCN8H1\Amina committed on 17 Mar	5cc5556	🔗

6.10 Integrasjonstesting

The screenshot shows the Postman interface for testing an API. At the top, it displays a GET request to <https://wixapi.azurewebsites.net/api/Modules>. A red arrow points from the URL to the 'API Modules' button in the top right corner. Below the request, the 'Headers' tab is selected, showing 8 headers. Another red arrow points from this tab to the 'Status av response' section. The status is listed as 200 OK. The 'Body' tab is also visible, stating 'This request does not have a body'. The 'Test Results' tab is shown at the bottom. In the bottom half of the interface, there is a history of requests. A red arrow points from the 'Authorization' tab of a DELETE request to a note about sensitive data. The note says: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables.' It also links to 'Learn more about variables'. The 'Token' field contains a long string of characters. The 'Status' for this request is 200 OK.

6.11 Systemtesting

By priority	WIX Testbook	Date	16.05
-------------	--------------	------	-------

	Critical	9	Total tests	95	Web deployment	1.0.0
	High	59	Passed	82	Environment	Local
	Medium	13	Failed	6	Tester Name	Alina Zielinska
	Low	12	NA	7	Device	HP 14-al174n0
Num	Feature	Objective	Priority	Steps to Reproduce	Expected Behavior	Result
1.1	Home Page	Verify top menu on the Home Page	High	1. Go to the Home Page 2. Observe	Menu observed on the top of the Home Page	OK
1.2	Home Page	Verify Logo on the Home Page	Medium	1. Go to the Home Page 2. Observe	Logo observed on the top-left of the Home Page	OK
1.3	Home Page	Verify main description on the Home Page	High	1. Go to the Home Page 2. Observe	Description observed on the top-left under logo of the Home Page	OK
1.4	Home Page	Verify footer on the Home Page	Low	1. Go to the Home Page 2. Observe	Footer observed on the bottom of the Home Page	OK
1.5	Home Page	Verify Learn More button on the Modul Card	Critical	1. Go to the Home Page 2. Observe	Learn more button observed on the bottom of the Modul Card	OK
1.6	Home Page	Learn More button redirect to the ModulPage of the particular Module	Critical	1. Go to the Home Page 2. Click on the Learn More button 3. Observe	User redirected to the Modul Page of the particular Module	OK
1.7	Home Page	Verify picture on the Modul Card that was send during creating Modul	High		Picture observed on the top of the Modul Card	Bug

1.8	Home Page	Verify default picture on the Modul Card if there is no picture was send during creating Modul	Medium		Epty space observed on the top of the Modul Card	OK
1.9	Home Page	Verify title on the Modul Card	High	1. Go to the Home Page 2. Observe	Description observed above the learn more button of the Modul Card	OK
1.10	Home Page	Verify lecturer on the left side of the Modul Card	High	1. Go to the Home Page 2. Observe	Name and surname of the lecturer observed under picture in the top-left corner of the Modul Card	OK
1.11	Home Page	Verify timing on the right side of the Modul Card	High	1. Go to the Home Page 2. Observe	Timing observed under picture in the top-right corner of the Module Card	OK
1.12	Home Page	Clicking on the button Learn more user goes to the particular description of the particular module	Critical	1.Go to the Home Page 2. Click Learn More button 3. Observe	User redirects to the Modul Main Page of particular Modul	OK
1.13	Home Page	Clicking on the Login in the Menu redirect user to the Login Page	Critical	1.Go to the Home Page 2. Click Login button 3. Observe	User redirects to the Login Page	OK
1.14	Home Page	Clicking on the About in the Menu redirect user to the About Page	Critical	1.Go to the Home Page 2. Click About button 3. Observe	User redirects to the About Page	OK
1.15	Home Page	Clicking on the Module in the Menu redirect user to the same Page	Critical	1.Go to the Home Page 2. Click Module button 3. Observe	User update Modul Page	OK

2.1	About Page	Verify top menu on the Home Page	High	1. Go to the About Page 2. Observe	Menu observed on the top of the Home Page	OK
2.2	About Page	Verify Logo on the Home Page	Medium	1. Go to the About Page 2. Observe	Logo observed on the top-left of the Home Page	OK
2.3	About Page	Verify Logo of the Institutions	Medium	1. Go to the About Page 2. Observe		OK
2.4	About Page	Verify link from the institutions logo	Low	1. Go to the About Page 2. Click on the every institution's logo 3. Observe	Cliking on the logo should redirekt user to the Home Page of the particular institution in the separete tab	OK
2.5	About Page	Verify text about project	Low	1. Go to the About Page 2. Observe		OK
2.6	About Page	Verify pictures in about project	Low	1. Go to the About Page 2. Observe		OK
2.7	About Page	Verify footer on the Home Page	Low	1. Go to the Home Page 2. Observe	Footer observed on the bottom of the Home Page	OK

3.1	Modul Main Page	Verify top menu on the MainModul Page	High	1. Go to the MainModul Page 2. Observe	Menu observed on the top of the MainModul Page	OK
3.2	Modul Main Page	Verify Logo on the MainModul Page	Medium	1. Go to the MainModul Page 2. Observe	Logo observed on the top-left of the MainModul Page	OK
3.3	Modul Main Page	Verify Title of the Modul	High	1. Go to the MainModul Page 2. Observe		OK
3.4	Modul Main Page	Verify Description of the Modul	High	1. Go to the MainModul Page 2. Observe		OK
3.5	Modul Main Page	Verify Modules schedule	High	1. Go to the MainModul Page 2. Observe		OK

3.6	Modul Main Page	Verify Modules summary	High	1. Go to the MainModul Page 2. Observe	All fields in the summary should be filled out	OK
3.7	Modul Main Page	Verify Modules lecturer	High	1. Go to the MainModul Page 2. Observe	Lecturer should include picture, name and title	OK
3.8	Modul Main Page	Verify button Start the course	High	1. Go to the MainModul Page 2. Observe		OK
3.9	Modul Main Page	User redirects to the first lesson of the course clicking on the Start the course button	Critical	1. Go to the MainModul Page 2. Click the "Start the course" button 3. Observe	User should be redirected to the first lesson of the particular course	OK
4.1	Text Lesson	Verify top menu on the Text Lesson Page	High	1. Go to the Text Lesson Page 2. Observe	Menu observed on the top of the Text Lesson Page	OK
4.2	Text Lesson	Verify Logo on the Text Lesson Page	Medium	1. Go to the Text Lesson Page 2. Observe	Logo observed on the top-left of the Text Lesson Page	OK
4.3	Text Lesson	Verify title on the Text Lesson Page	Critical	1. Go to the Text Lesson Page 2. Observe		OK
4.4	Text Lesson	Verify description of the lesson	Critical	1. Go to the Lesson Page 2. Observe		OK
4.5	Text Lesson	Verify Next button on the Text Lesson Page	High	1. Go to the Lesson Page 2. Observe	Next button should be on the bottom-right under the description of the lesson	OK
4.6	Common elements Lessons	Next button lead to the next lesson of the modul	High	1. Go to the Lesson Page 2. Click on the Next Button 3. Observe	Next button should lead to the particular next lesson after text lesson	OK
4.7	Common elements	Verify Pagination	Medium	1. Go to the Lesson Page 2. Observe	Pagination should be in on the center-top above the title	OK

	Lesson s					
4.8	Comm on eleme nts Lesso n s	Pagination functionality	High	1. Go to the Lesson Page 2. Observe	Pagination lead to the Lesson that represent of the particular number	NA
4.9	Comm on eleme nts Lesso n s	Breadcrumb	Medium	1. Go to the Lesson Page 2. Click on the links from breadcrumb 3. Observe	Breadcrumb lead to the particular page according to the link	OK
4.10	Comm on eleme nts Lesso n s	Previous button on the first lesson of the Module	Medium	1. Go to the Lesson Page 2. Observe	Previous button should be unactive on the first lesson of the Module	OK
4.11	Comm on eleme nts Lesso n s	Title	Medium	1. Go to the Lesson Page 2. Observe	Title of the lesson should appear	OK
5.1	Log in	Log in with correct credentials and access the module-page	High	1. Fill Email 2.Fill Password 3.Press submit 4. Redirect to modules-page	Giving right credentials should let the user access to modules-page	OK
5.2	Log in	Attempt to log in with wrong credentials and get a pop-up window that will show error message	High	1. Fill wrong credentials in email and password areas 2. Press submit 3. Pop-up window appears press ok to continue	An error message appears explain the error that the user has caused.	OK
5.3	Log in	Attempting to fill in something else than a email and get error message	Medium	1. fill in email wrong email or something that doesnt have "@" 2. Press submit	An error message appears explain the error that the user has caused.	OK

5.4	Log in	Attempting to fill an password that is longer than the character (20) limit	Medium	1. Fill in password with more than 20 characters 2. press on the surface of the website	An error message appears beneath the password-input	OK
5.5	Log in	After initial log in, have the website remember my credentials	Low	1. Fill email 2. Fill password 3. Check "Check me out" 4. Press Submit 5. Press log out 6. Press Log in 7. Press submit	Both email and password input-fields should automatically have the credentials previously used and be able to log in	OK
5.6	Log in	Log in with Facebook	Low	1. Press "Sign in with Facebook" 2. Fill email 3. Fill password 4. Press connect	User will be able to log in with his/her Facebook profile	OK
5.7	Log in	Log in as Admin in User log in page	High	1. Fill email with admin email 2. Fill password with admin password, Press submit	Error message appears explaining that email or password is wrong and not let the admin access from user log in	OK

6.1	Register	Fill in right credentials in the input-fields and have a user registered	High	1. Fill all the input-fields with correct info 2. Press Save button	User should be created and appear in module-page upon filling all the input-fields and pressing the Save button.	Ok
6.2	Register	Fill in wrong credentials and get an error message when you press Save button	High	1. Fill all the input-fields with wrong info 2. Press Save button	Error message appears explaining the error caused by the user upon registering.	OK
6.3	Register	Fill in individual input-fields with wrong credentials	High	1. Fill input field with wrong info	Error message appears right beneath that input-field	OK

				2. Press anywhere else but that input field		
6.4	Register	Fill in wrong format of phone number, email, name,surname	High	1. Fill in wrong format of characters for Name,surname,email and phone input fields 2. Press Save button	Error messages appears beneath every wrong input-field	OK
6.5	Register	Field are not filled in	High	1. Do not fill in info into fields 2. Press Save button	Error messages appears beneath every empty input-field	OK
7.1	Text Lesson	Text content	High	1. Go to the Text Lesson Page 2. Observe	Text content should be observed in the middel of the cscreen ander title	OK
7.2	Text Lesson	Picture in the text content	Medium	1. Go to the Text Lesson Page 2. Observe	Picture can be observed in the content	NA
7.3	Text Lesson	Formatting	High	1. Go to the Text Lesson Page 2. Observe	Text content should be formatted	Bug
7.4	Video Lesson	Video	High	1. Go to the Video Lesson Page 2. Observe	Video should be observed in the middle of the screen under title	NA
7.5	Video Lesson	Video functionality	High	1. Go to the Text Lesson Page 2. Observe	It should be possible to play/ pause/ rewind/ forward video	NA
7.6	Last Page	Last Page	Medium	1. Go to the last Page 2. Click on button Next 3. Observe	Clicking Next button will redirect to the Finish Page	OK
7.7	Finish	Finish Page	Low	1. Go to the Finish Lesson Page 2. Observe	Observe text with congratulations about finishing the course in the middle of the screen	OK

8.1	Quiz	Question blokk	High	1. Go to the Quiz Page 2. Observe	Verify question blokk with question and answer alternatives	OK
8.2	Quiz	Check out boks	High	1. Go to the Quiz Page 2. Check out answer 3. Observe	Check box can be check out and check in	OK
8.3	Quiz	Colour for the answer	Low	1. Go to the Quiz Page 2. Check out answer 3. Observe	Red indicator for the wrong answer, green indicator for the right answer	NA
8.4	Quiz	Tips	Low	1. Go to the Quiz Page 2. Click on the questioned mark 3. Observe	Answer and tips should appear	OK
9.1	Admin Main	Adding text field	Low	1. Go to the AdminMain Page 2. Observe	Observ field for adding text on the top of the screen	OK
9.2	Admin Main	Adding text function	Low	1. Go to the AdminMain Page 2. Observe	Adding text in the field chaging text ont hte main User Page	NA
9.3	Admin Main	Create card	High	1. Go to the AdminMain Page 2. Observe	Create card should be the first element under adding text field	OK
9.4	Admin Main	Create button function	High	1. Go to the AdminMain Page 2. Click on the button "Create" 3. Observe	Pushing the button admin redirected to the admin Modul Page	OK
9.5	Admin Main	Modules card	High	1. Go to the AdminMain Page 2. Observe	All existing modules cards observed on the page	OK
9.6	Admin Main	Eddit button function	High	1. Go to the AdminMain Page 2. Click on the button "Eddit"	Pushing the button admin redirected to the admin Modul Page	OK

				3. Observe		
Test Cases for Admin Modul Page						
10.1	Admin Modul	Verify field for title	High	1. Go to the Admin Modul Page 2. Observe		OK
10.2	Admin Modul	Verify field for description	High	1. Go to the Admin Modul Page 2. Observe		OK
10.3	Admin Modul	Verify summary field	High	1. Go to the Admin Modul Page 2. Observe	Duration, Institution, Subject, Price, Language, Picture	OK
10.4	Admin Modul	Verify field for adding lecturer	High	1. Go to the Admin Modul Page 2. Observe	Picture, title	BUG
10.5	Admin Modul	Verify field for schedule of the course	High	1. Go to the Admin Modul Page 2. Observe	Fields for number, name and duration	OK
10.6	Admin Modul	Title functionality	High	1. Go to the Admin Modul Page 2. Type in title of the course 3. Go to ModulPage	Title should appear on the Modul Page	OK
10.7	Admin Modul	Description functionality	High	1. Go to the Admin Modul Page 2. Type in description of the course 3. Go to ModulPage	Description should appear on the Modul Page	OK
10.8	Admin Modul	Duration	High	1. Go to the Admin Modul Page 2. Type in duration of the course 3. Go to ModulPage	Duration should appear on the Modul Page	OK
10.9	Admin Modul	Institution	High	1. Go to the Admin Modul Page 2. Type in institution of the course 3. Go to ModulPage	Institution should appear on the Modul Page	OK
10.10	Admin Modul	Subject	High	1. Go to the Admin Modul Page	Subject should appear on the Modul Page	OK

				2. Type in subject of the course 3. Go to ModulPage		
10.1 1	Admin Modul	Price	High	1. Go to the Admin Modul Page 2. Type in price of the course 3. Go to ModulPage	Price should appear on the Modul Page	OK
10.1 2	Admin Modul	Language	High	1. Go to the Admin Modul Page 2. Type in Language of the course 3. Go to ModulPage	Language should appear on the Modul Page	OK
10.1 3	Admin Modul	Picture	High	1. Go to the Admin Modul Page 2. Type in URL for picture of the course 3. Go to HomePage 4. Go to the AdminMain Page	Picture should appear on the Card on the Home Page for Admin And for the user	BUG
10.1 4	Admin Modul	Title on the card	High	1. Go to the Admin Modul Page 2. Type in title of the course 3. Go to Home Page	Title should appear on the Card on the Home Page	OK
10.1 5	Admin Modul	Edit card	High	1. Go to the Admin Main Page 2. Click on the Edit button 3. Change existing data 4. Go to the AdminMain Page 5. Go to the Home Page	Data should be changed on the Card and on the Main ModulPage on the Home Page for Admin And for the user	OK
11.1	Admin Lesson Comm on	Header	High	1. Go to the Admin Lesson Page 2. Observe	Field for fil inn header should appear	OK
11.2	Admin Lesson Text	Text	High	1. Go to the Admin Lesson Page 2. Observe	Field for fil inn header should appear	OK

11.3	Admin Lesson Text	Text functionality	High	1. Go to the Admin Text Lesson Page 2. Type in text in the field for text 3. Go to the User Lesson Page 4. Observe	Text should appear	OK
11.4	Admin Lesson Video Text	Video functionality	High	1. Go to the Admin Video Lesson Page 2. Download video 3. Go to the User Video Page 4. Observe	Video should appear	NA
11.5	Admin Lesson Quiz	Quiz question	High	1. Go to the Admin Quiz Lesson Page 2. Type in question 3. Go to the User Quiz Page 4. Observe	Question appear	OK
11.6	Admin Lesson Quiz	Quiz answers	High	1. Go to the Admin Quiz Lesson Page 2. Type in answers alternativs 3. Go to the User Quiz Page 4. Observe	Answers alternatives appear	Bug
11.7	Admin Lesson Quiz	Right answer in a check box	High	1. Go to the Admin Quiz Lesson Page 2. Tick right answer from the alternatives in the check box 3. Go to the User Quiz Page 4. Observe	Indikasion of the right answer appear	Bug
11.8	Admin Lesson Quiz	Right answer as a hint	High	1. Go to the Admin Quiz Lesson Page 2. Type in right answer as a hint 3. Go to the User Quiz Page 4. Observe	Hint appear	OK

11.9	Admin Lesson Quiz	Add one more question	High	<ol style="list-style-type: none">1. Go to the Admin Quiz Lesson Page2. Click on the "Add one more question"3. Observe	Block for new question appear	OK
------	-------------------------	--------------------------	------	--	-------------------------------	----