Bachelor oppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for IKT og realfag

Bakken, Aleksander
Tranvik, Brage Aspli
Vaaland, Trym

CT-Wood Arkivsystem

Bacheloroppgave i Dataingeniør
Veileder: Styve, Arne

Mai 2021

NTNU
Kunnskap for en bedre verden

Bakken, Aleksander
Tranvik, Brage Aspli
Vaaland, Trym

CT-Wood Arkivsystem

Bacheloroppgave i Dataingeniør
Veileder: Styve, Arne
Mai 2021

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for IKT og realfag

NTNU
Kunnskap for en bedre verden

Forord

Denne bacheloroppgaven er den avsluttende oppgaven på studiet Bachelor dataingeniør våren 2021 ved NTNU i Ålesund. Oppgaven er skrevet av:
Aleksander Bakken, Brage Aspli Tranvik og Trym Vaaland

Denne bacheloroppgaven er gitt som oppdrag fra Luleå tekniska universitet i Skellefteå som har en ledende plassering innen stokkskanning, tørking, varmebehandling, fuktdynamikk og mer. Oppgaven går ut på å utvikle et arkivsystem som kan anvendes gjennom en nettside. Dette arkivsystemet skal bli brukt hovedsakelig av forskere som benytter en CT-skanner og et klimakammer for å forske på tre og deres egenskaper.

Denne oppgaven ble valgt fordi den var interessant og ganske unik hvor det blir tatt i bruk interessant teknologi. Gruppen har erfaring innen webutvikling og databaser, slik at oppgaven ikke blir alt for fremmed samtidig som at den blir litt utfordrende. Denne oppgaven virket også som en god mulighet for å lære om nye teknologier og strategier som gruppen kunne ta i bruk i det framtidige jobblivet.

Vi vil veldig gjerne takke:
Arne Styve for fantastisk veiledning gjennom hele prosjektet.
Lars Hansson hos Luleå tekniska universitet for god og lærerik hjelp, samt et artig og lærerikt prosjekt.

Innhold

Figurer **viii**

Sammendrag **xii**

Terminologi **xiii**

1 Innledning **1**

1.1 Bakgrunn 1

1.2 Problemstilling 1

1.3 Dagens løsning 1

1.3.1 Systemskisser 3

1.4 Mål 4

1.5 Kravspesifikasjon 4

1.5.1 Interessenter 4

1.5.2 Krav til filer systemet skal håndtere 6

1.5.3 Avgrensninger 7

1.5.4 Dokumentasjon 7

1.6 Rapportens videre innhold 7

2 Teoretisk Grunnlag **8**

2.1 CT skanning 8

2.2 Filtyper 8

2.2.1 DICOM (ima) 8

2.2.2 Tiff 8

2.3 Brukergrensesnitt 8

2.4 Design 9

2.4.1 Trådramme 9

2.4.2 Brukersentrert design 9
2.4.3 Model-Visning-Kontroller Designmønster 11
2.5 Relasjondatabase 12
2.6 Versjonskontroll 12
2.7 Prosjektplanlegging/gjennomføring 13
2.7.1 Kravspesifikasjon 13
2.7.2 Flytskjema 13
2.8 Virtualisering 14
2.9 Cross-Origin Resource Sharing (CORS) 14
2.10 Agile metoder 14
2.11 Scrum 15
2.11.1 Scrum team 15
2.11.2 Scrum handlinger 16

3 Materialer og metode **19**

3.1 Prosjektorganisasjon 19
3.1.1 Prosjektgruppen 19
3.1.2 Oppdragsgiver 19
3.1.3 Veileder 19
3.1.4 Sprint og møter 20
3.1.5 Prosjektorganisering 20
3.2 Planlegging 20
3.2.1 Prosjektstyringsverktøy 20
3.2.2 Kravspek og møtenotater 25
3.2.3 Trådrammer 25
3.2.4 Flytskjema 25

3.3 Utviklingsverktøy 25
3.3.1 IntelliJ Idea 25
3.3.2 WebStorm Ide 26
3.3.3 dbdiagram.io 27
3.3.4 Nettleser utviklingsverktøy 27
3.4 Distribuering av applikasjon 28
3.4.1 Virtuell maskin 29
3.4.2 Maven 29
3.4.3 Docker 29
3.5 Programmeringsspråk 30
3.5.1 Java 30
3.5.2 Hypertext Markup Language (HTML) 30
3.5.3 Cascading Style Sheet (CSS) 31
3.5.4 JavaScript 31
3.6 Rammeverk & biblioteker 32
3.6.1 React 32
3.6.2 Spring & Spring Boot 33
3.6.3 Lombok 34
3.6.4 Eclipse Jersey 34
3.6.5 imgscalr - Java Image-Scaling Library 35
3.6.6 JSON Web Token 35
3.6.7 Jasypt Spring Boot 36
3.6.8 EclipseLink JPA (Java Persistence API) 37
3.6.9 Apache Commons IO 38
3.6.10 jcifs-ng 39
3.7 Testing 39

3.7.1 Postman . . . 39
3.7.2 SonarLint . . . 40
3.7.3 Brukertesting . . . 40
3.7.4 Integrasjonstester . . . 41
3.8 Kommunikasjon . . . 41
3.8.1 HTTP-statuskoder . . . 41

4 Resultater **42**

4.1 Konsept . . . 42
4.2 GUI - Design . . . 43
4.3 GUI-Brukergrensesnitt . . . 51
4.3.1 Hjemmesiden . . . 52
4.3.2 Login siden . . . 52
4.3.3 Framsiden til en bruker . . . 53
4.3.4 Opprett nytt prosjekt . . . 55
4.3.5 Prosjekt siden . . . 56
4.3.6 Admin kontrollpanel . . . 64
4.4 Prosjektgjennomføring . . . 71
4.4.1 Scrum . . . 71
4.4.2 Generelt . . . 72
4.5 Brukertesting . . . 72
4.5.1 Testresultat . . . 73
4.6 Kjente feil og mangler . . . 73
4.6.1 Kjente mangler . . . 73
4.6.2 Kjente feil . . . 73
4.7 Backend . . . 73

4.7.1 Database 74
4.7.2 REST kontroller 75
4.8 Installasjon og konfigurering av applikasjonen 76

5 Drøfting **78**
5.1 Resultat evaluering 78
5.1.1 Sikkerhet 78
5.1.2 Brukervennlighet 79
5.1.3 Brukergrensesnitt 81
5.1.4 Backend 82
5.1.5 Vår bruk av HTTP-statuskoder 82
5.1.6 Feil og mangler 83
5.1.7 Brukertest resultatet 84
5.1.8 Database 85
5.1.9 Samarbeid 85
5.1.10 Scrum 85
5.2 Problemer gjennom prosjektet 86
5.2.1 Docker 86
5.2.2 CORS feilmeldinger 86
5.3 Arbeidsfordeling 86

6 Konklusjon **88**

Bibliografi **89**

Vedlegg **a**
A Prosjekt-avhengigheter . . . . . a
A.1 Frontend . . . . . a

A.2 Backend . . . . . b

B SQL c

B.1 Generering av tabeller (PostgreSQL) c

B.2 Generering av roller og en bruker (PostgreSQL) . . . . . e

C Brukertest . . . . . f

D API - forespørseler i

E Backend - klassediagram . . . . . o

F Forprosjektrapport . . . . . p

Figurer

1 Gamle fil løsningen . . . 3
2 Ønsket løsning . . . 3
3 Use-Case diagram . . . 5
4 Trådramme eksempel . . . 9
5 Consistency prinsipp . . . 11
6 MVC . . . 11
7 Versjonskontroll . . . 13
8 Scrum rammeverket flytskjema . . . 17
9 Prosjektgruppen . . . 19
10 Git-flow . . . 23
11 gitKraken . . . 24
12 IntelliJ Idea . . . 26
13 dbdiagram.io . . . 27
14 Google Chrome sitt utviklingsverktøy . . . 28
15 Git code percentages . . . 31
16 Lombok . . . 34
17 ForbiddenException . . . 35
18 imgscalr . . . 35
19 Jwt Eksempel . . . 36
20 ENC Eksempel . . . 36
21 Jasypt Kryptering Eksempel . . . 37
22 JPA . . . 38
23 Commons IO eksempel . . . 38
24 Postman . . . 40

25 Hvordan resultatet skal bli 42
26 Hvordan resultatet skal bli 43
27 Framside konsept 44
28 Lag prosjekt side konsept 44
29 Fil opplasting side konsept 45
30 Prosjekt detaljer side konsept 45
31 Endre på prosjekt tag side konsept 46
32 Prosjekt filer side konsept 46
33 Prosjekt bilder side konsept 47
34 Prosjekt medlemmer side konsept 47
35 Prosjekt tillatelse side konsept 48
36 Ny bruker side konsept 48
37 Finn en bruker til å endre side konsept 49
38 Endre på en bruker side konsept 49
39 Slett en bruker side konsept 50
40 Slett en tag side konsept 50
41 Eksempel på dialog konsept 51
42 Navigasjonsbar når brukeren ikke er logget inn 51
43 Navigasjonsbar som en admin bruker 51
44 Hjemmeside 52
45 Login side resultat 53
46 Framsiden som en user bruker 54
47 Framsiden som en academic bruker 54
48 Opprett nytt prosjekt siden 55
49 Prosjekt navigasjonsbar 56
50 Prosjekt detaljer siden 57

51 Oppretting av tag 57
52 Endring av tags i prosjekt 58
53 Opplasting siden 59
54 Prosjekt filer 60
55 Prosjekt bilder 60
56 Prosjekt medlemmer 61
57 Prosjekt medlemmer 62
58 Bekreftelse boks 62
59 Prosjekt spesial tillatelse som user 63
60 Prosjekt spesial tillatelse 63
61 Prosjekt spesial tillatelse bekreftelse 64
62 Admin forside 64
63 Create User 65
64 Find user 66
65 Edit user 67
66 Errormelding 67
67 Delete User 68
68 Delete User dialogboks 68
69 Delete Tags 69
70 Delete Tags Modal 69
71 Omstart Server side 70
72 Omstart dato 70
73 Omstart tid 71
74 Omstart tidssone 71
75 Gantt diagram 72
76 Backend mode 74

77 databaseResultat 75

78 application.properties 77

79 Rolle sikkerhet 79

80 Backend klasse diagram . . . . . o

Sammendrag

Ved Luleå tekniska universitet (LTU) så utføres materialstudier på tre. Til disse studiene anvendes en tomograf (CT-skanner) og et klimakammer. For hvert prosjekt hvor tomografen blir benyttet blir det generert en stor mengde med informasjon. Denne informasjonen kommer i form av data og et stort antall bilder som de ansatte bruker i sin forskning ved universitetet. Fram til nå har de håndterte disse dataene manuelt på en måte som er lite brukervennlig. LTU ønsker nå å ha en bedre måte å organisere disse dataene på.

Målet med dette prosjektet er å effektivisere arbeidet til de ved LTU som jobber med stokkskanning, tørking, varmebehandling, fuktdynamikk og mer. Dette blir oppnådd ved å skape et arkivsystem som skal kunne holde på deres filer på en filserver, en database som kan holde på deres prosjekter, informasjon om prosjektene og informasjon om brukere, og en webapplikasjon som skal være ett enkelt brukergrensesnitt slik at det er mulig å hente informasjon og filer fra ett prosjekt.

Resultatet vårt ble en nettside som snakker med en API som snakker med en filserver. Brukere kan ut ifra resultatet vårt lage prosjekt, holde styr på medlemmer og hvem som kan se prosjektene, laste opp og laste ned filer til filserveren gjennom en enkel nettside.

I frontend har vi brukt biblioteket React for å lage et brukergrensesnitt. Vi brukte Spring Boot for å skape en Spring applikasjon til vår backend. Spring applikasjoner benytter språket Java for å sette opp konfigurasjonen og gjøre det enkelt å lage applikasjoner. Vi var kjent med PostgreSQL fra før og derfor valgte vi det til databasen vår.

Terminologi

Begreper

Backend	Backend er programvaren som ligger nærmest databasen og utfører tyngre operasjoner.
Frontend	Frontend er programvaren som ligger nærmest brukeren og det er den som bygger opp brukergrensesnittet. Frontend kommuniserer med backend ved hjelp av REST forespørsler.
DICOM	Digital Imaging and Communications in Medicine er et bilde format som inneholder forskjellig informasjon. Dette bildeformatet er ofte brukt av CT-skannere.
Tiff	Tiff er et bilde format som blir brukt til fotografi og elektroniske dokumenter. Tiff kan bli lagret som en stabel som inneholder flere bilder. Tiff er også brukt av CT-skannere.
Parameter	Et parameter er et argument som blir brukt innen koding. Det blir brukt slik at en verdi/variabel inni koden kan bli endret etter situasjonen.
Full Stack	Et begrep som blir brukt når det er snakk om håndtering av databaser, servere, systemer og klienter. Altså alt som kan inngå i ett IT prosjekt.
Pull request	Innenfor versjonsstyring så brukes pull requests for å vise arbeidet som er gjort på en grein. Dette arbeidet diskuteres og kan bli godkjent for å bli med inn på en annen grein.
User story	En user story er en kort enkel beskrivelse av en funksjonalitet, fortalt fra synspunktet til personen som ønsker funksjonaliteten.
Url	En url er en lenke som blir brukt til å koble seg til et nettsted. Eksempel: *https://www.eksempel.no*
Internet Protocol	En unik adresse som brukes til å identifisere en enhet på internettet eller et lokalt nettverk.

Forkortelser

NTNU	Norges teknisk-naturvitenskapelige universitet
LTU	Luleå tekniska universitet
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
IDE	Integrated Development Environment

API	Application Programming Interface
XML	Extensible Markup Language
JSX	JavaScript XML
DOM	Document Object Model
SQL	Structured Query Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
SMB	Server Message Block
REST	Representational State Transfer
GUI	Graphical User Interface
CT	Computertomografi
JWT	JSON Web Token
IP	Internet Protocol

Verktøy

GitKraken	Grafisk brukergrensesnitt for Git
GitHub	Git repository tjeneste for å holde på prosjekt filer
IntelliJ	Integrert utviklingsmiljø laget for Java
WebStorm	Integrert utviklingsmiljø laget for JavaScript
Java	Objekt orientert programmeringsspråk
JavaScript	Programmering språk brukt for klient-side logikk og rendering
HTML	Brukt til tagging av tekst filer for å styre font, farger, grafikk og hyperkoblinger
JSX	Utvidelse til JavaScript syntaksen som brukes til å strukturere komponentene på nettsiden
CSS	Verktøy som blir brukt for å endre på utseende til et HTML-dokument
PostgreSQL	Relasjonsdatabase brukt som en sikker måte å holde på informasjon
Maven	Verktøy for å bygge Java prosjekter likt uansett plattform
Postman	Verktøy for å teste API forespørsler fra serveren
Discord	Brukt for å kommunisere over nettet og dele informasjon mellom gruppemedlemmer
Jira	Issue sporings-verktøy for Agile prosjekt gjennomføring
Confluence	Plattform for å samle gruppens ideer, notater og diagrammer på en plass

1 Innledning

1.1 Bakgrunn

Luleå tekniska universitet (LTU) utfører materialstudier på tre. En tomograf (CT-skanner) og et klimakammer blir ofte benyttet for disse studiene. Når tomografen blir benyttet så blir det skapt store mengder med informasjon på filer. Filene består av både data og bilder, som anvendes til forskningen. Disse filene blir behandlet hovedsakelig for manuelt og blir lagret på disk og filserver.

1.2 Problemstilling

Når det blir utført materialstudier og en tomograf blir tatt i bruk, så vil det bli skapt mye data og mange bilder. Denne informasjonen håndteres for tiden svært manuelt. Filene vil bli liggende på enten en harddisk eller en filserver, med ingen form for standardisert organisering.

Uten noen form for standard organisering kombinert med vanskeligheter for uthenting av relevant informasjon så leder det til unødvendig sløsing med tid. Problemstillingen vår er å redusere den unødvendige sløsingen med tid og reduser dobbeltarbeid utført.

1.3 Dagens løsning

Den løsningen LTU bruker nå er veldig manuell. Prosessen fra skanning til arkivering kan beskrives slik:

1. Gjenstanden går gjennom CT-skanneren
2. Bildene lagres på en datamaskin som er tilkoblet CT-skanneren
3. Bildene kopieres manuelt over til en filserver

Den type data som et prosjekt kan inneholde er CT-bilder, loggfiler, andre bilder, rapporter og lenker til aktuelle journalartikler. Alt dette blir lagt inn på filserveren. De som er involverte i et prosjekt er ansvarlige for de bildene og dataene som hører til. Dette blir beskrevet som en veldig sårbar løsning. Se tabell 4 for mer detaljer.

Tabell 4: Oversikt over hvilke typer data som lagres

Type informasjon	Format	Kommentar
Bilder fra Siemes CT-scan	DICOM	Se https://en.wikipedia.org/wiki/DICOM Storleken på bilderna är 527 kB. Antal filer kan variera från 1 upp till 15000 st kanske fler. Olika varianter av skann-projekt. • Ett skann, dvs en position, en tidpunkt. • Ett skann, dvs en position, i flera tidspunkter. • Flera skann, dvs flera positioner, en tidpunkt. • Flera skann, dvs flera positioner, i flera tidspunkter. I bild-huvudet finns information ombland annat, position, upplösning (x, y, z)-led, skann-tidspunkt, olika konfigurationer av CT-skannern.
Bilder från Microtec CT-skannern.	TIFF (Stack)	Vi utarbetar för närvarande formatet med Microtec, eftersom vi är inte nöjda med hur det är utformat nu. Filstorleken är ca 500 - 1000 MB
Loggfiler	xlsx, txt	Temperatur och processparametrar från klimatanläggningarna
Bilder	tiff, jpg	Bilder på försöksuppställningar, med mera.
Rapporter	MS Word, PDF	
Länkar		Länkar till journalartiklar som anknyter till projektet.

Noen problemer med denne løsningen er:

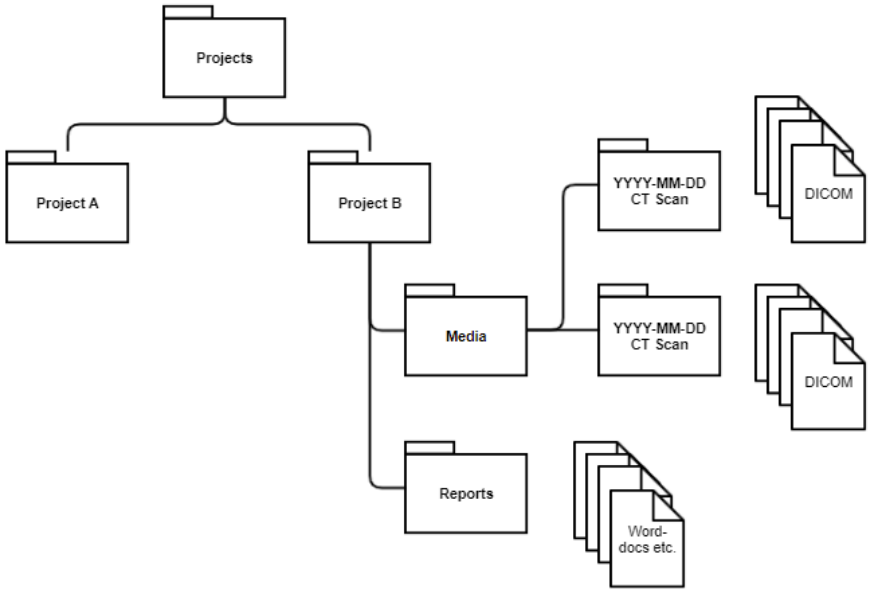
Søk
Når dataene ligger som en mappestruktur på filserveren er det ingen måte å søke gjennom innholdet hvis det er noe spesifikt du leter etter. Dataene er ikke tagget og det eneste som er mulig å søke etter er mappe- og filnavn.

Brukervennlighet
Som løsningen er i dag, har ikke de ansatte tilgang til data som andre personer har scannet og lagret. Dette er dårlig for brukervennligheten. For å dele informasjonen må de antageligvis sende den gjennom andre kanaler, som er en omvei å gå.

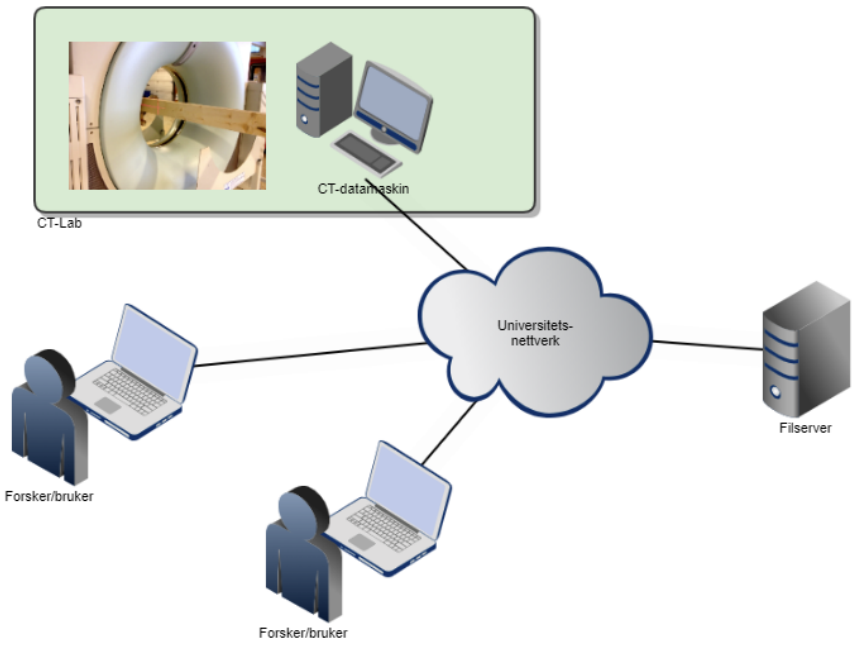
Organisering
Dette er en uoversiktlig løsning. Ansatte har forskjellige måter å organisere dataene de er ansvarlig for og dette gjør at det kan være stor variasjon på mappestrukturen for hvert prosjekt.

Skisser og eksempel om hvordan dataene blir sortert og hvordan ønsket resultat ser ut.

1.3.1 Systemskisser



Figur 1: Dagens løsningen på filstruktur



Figur 2: Hvordan dagens løsning er

1.4 Mål

Målet med denne oppgaven er å skape ett sluttprodukt som møter kravspesifikasjonene fra oppdragsgiveren. Sluttproduktet må kunne brukes av forskere og studenter for å se og håndtere filer i en trestruktur, og sluttproduktet burde derfor være enkelt og intuitivt å bruke.

Vi vil forstå og tolke kravspesifikasjoner og ønskene fra kunden på en måte hvor kunden blir fornøyd med vår atferd i forhold til prosjektet og vårt sluttprodukt.

1.5 Kravspesifikasjon

I startfasen utviklet vi i samarbeid med oppdragsgiver hvilke krav som prosjektet skal ha. Noen av kravene vi fikk var at det skal være en web basert løsning, hvilke brukere systemet skal ha, hvilke type bilder systemet skal håndtere, det skal være mulig å tagge prosjekter, det skal være mulig å overføre filer til filserveren og mer. Oppdragsgiver var også åpen for tilbakemelding når det kom til krav vi som utviklerne foreslo. Kravene handlet i hovedsak om hva forskjellige brukergruppen skal kunne gjøre i den nye løsningen, hvilke filer systemet skal kunne håndtere og om at løsningen skal være brukervennlig og enkel å bruke.

1.5.1 Interessenter

I kravspekket er det beskrevet fire type brukere som skal bruke systemet: systemeier, systemadministrator, forsker/medarbeider og student. Ut ifra møter har vi derimot funnet ut at system eier og system administrator kan være den samme rollen og endte opp med rollene: administrator, akademiker og bruker. Figur 3 viser de forskjellige rollene i et Use-Case diagram.

Figur 3: Use-case diagram for systemet

Beskrivelse som går mer i dybden:

Som en bruker skal jeg kunne:

- Se og gå inn på alle prosjekter
- Se og laste ned filer fra et prosjekt som ikke er privat eller brukeren har fått spesialtillatelse for å se
- Se på bilder fra et prosjekt som ikke er privat eller brukeren har fått spesialtillatelse for å se

Som en akademiker skal jeg kunne:
Alt som brukere kan **og**:

- Opprette nye tagger
- Legge til og fjerne tagger fra sine prosjekter
- Opprette nye prosjekt
- Legge til og fjerne brukere fra prosjektets medlemsliste som prosjekt eier
- Legge til og fjerne brukere som har spesielle rettigheter på et prosjekt som prosjekt medlem
- Laste opp filer til et prosjekt som medlem
- Velge å gjøre prosjekter private som medlem/eier
- Legge til tagger på filer og fjerne dem som medlem
- Slette filer til et prosjekt som medlem
- Legge til ikon til et prosjekt som eier
- Arkivere lenker innpå et prosjekt som et medlem

Som en administrator skal jeg kunne:
Alt som akademikere kan **og**:

- Opprette nye brukere
- Se alle brukernes detaljer og endre detaljer som epost og passord
- Slette brukere og tagger
- Planlegge dato og tid som serveren skal bli automatisk restartet på
- Se på statistikk om server detaljer

1.5.2 Krav til filer systemet skal håndtere

Det er et krav at løsningen vår skal håndtere alle filtypene som vanligvis brukes i sammenheng med et prosjekt. Disse er: skanninger (tiff, ima), loggfiler (xlsx, txt), bilder (jpg) og rapporter (docx, pdf).

1.5.3 Avgrensninger

Det ble ikke satt særlig mange avgrensninger til oppgaven, men det ble satt en på grunn av at vi kunne se at det var mye arbeid å gjøre, og at vi måtte prioritere det som var viktig. Oppdragsgiver ville gjerne ha en sikkerhetskopi løsning for systemet slik at det skulle være enkelt å ta sikkerhetskopier av filserveren som holdte på filene i prosjekt. I starten av prosjektet ble vi ganske enige om at det ville bli en tidskrevende funksjonalitet å utvikle og at vi derfor skulle unngå å jobbe med det før vi visste at vi hadde god tid.

1.5.4 Dokumentasjon

Ett av kravene som ble stilt til oss i dette prosjektet er at det skal være enkelt å kunne bygge på og videreutvikle vårt sluttprodukt. Derfor har vi prøvd å være nøye med å dokumentere alle filer som krever dokumentasjon, vi skal lage diagrammer som skal fortelle flyten gjennom nettstedet, flyten i databasen og hvilke API-forespørsler som frontenden kan kalle på.

1.6 Rapportens videre innhold

Kapittel 2: Teoretisk Grunnlag
Kapittel to inneholder informasjon om relevant teori som blir brukt igjennom rapporten.

Kapittel 3: Materialer og metode
Kapittel tre går igjennom materialer og metoder som er brukt igjennom oppgaven som hvordan vi har jobbet og hvilke verktøy, rammeverk og biblioteker vi har brukt til gjennomføring av prosjektet. Vi går også igjennom litt relevant viktig teori til de forskjellige emnene.

Kapittel 4: Resultater
I kapittel fire går vi igjennom resultatet av prosjektet. Vi starter med konseptet og designet før vi viser frem resultatet over hvordan nettsiden har blitt. Til slutt går vi igjennom feil, mangler og installasjon av applikasjonen.

Kapittel 5: Drøfting
I kapittel fem tar vi en evaluering over forskjellige resultater i prosjektet. Vi går også igjennom de mest tidskrevende problemene vi hadde i prosjektet og hvordan vi fordelte arbeidet til gruppen.

Kapittel 6: Konklusjon
I kapittel seks er det om kort konklusjon om prosjektet.

2 Teoretisk Grunnlag

2.1 CT skanning

CT som betyr *computertomografi*, er en radiologisk undersøkelsesmetode for snittfotografering. En CT-maskin fungerer med at et røntgenrør og diametralt monterte røntgendetektorer roterer rundt et objekt og tar et bildeopptak. Røntgenstrålene blir svekket av forskjellige stoffer i objektet som blir målt i forskjellige vinkler, og dermed blir det bygget opp forskjellige bilder i forskjellige vinkle/snitt. I prosjektene som arkivsystemet skal organisere blir det skannet tre, men CT-maskiner er normalt brukt på sykehuspasienter. (Brekke mfl. 2018)

2.2 Filtyper

2.2.1 DICOM (ima)

Digital **I**maging and **C**ommunications in **M**edicine (DICOM) med filtype *.ima* eller *.dcm* er en standard innen håndtering, lagring og overføring av digitale bilder og informasjon relatert til bildene. Et DICOM bilde kan inneholde en drøss av informasjon som: dato når bildet ble tatt, kroppsdel som er skannet, maskin id, pasient informasjon og mer. DICOM er et produkt av en CT-skanning og er dermed et røntgenbilde. DICOM 2020

2.2.2 Tiff

Tag **I**mage **F**ile **F**ormat med filtype *.tiff* eller *.tif* er et rastegrafikkbilde. Tiff kan også bli brukt som et produkt av en CT-skanning og kan inneholde flere bilder i samme fil fra en skanning. Dette blir kaldt en *Tiff stack*.

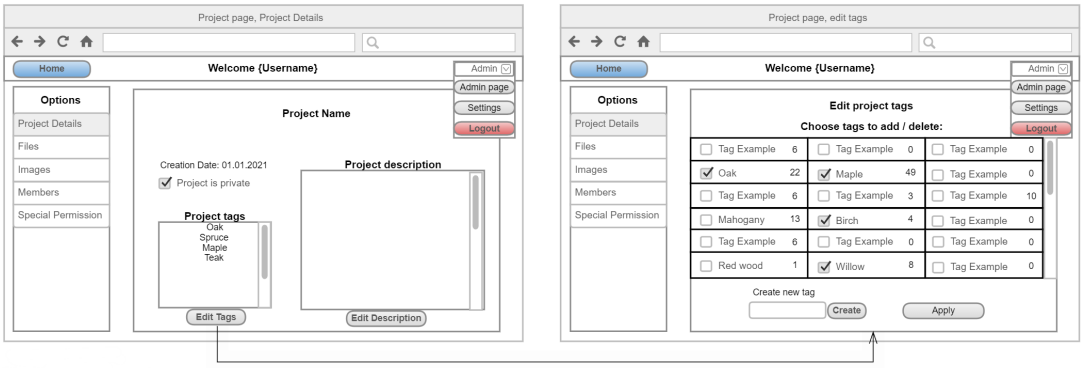
2.3 Brukergrensesnitt

Brukergrensesnitt er det som blir brukt av en bruker for å kommunisere og styre et operativsystem eller program. Et brukergrensesnitt kan enten være grafisk eller tekst basert. På et grafisk brukergrensesnitt blir det brukt en mus eller lignende for å styre, mens på en tekst-basert skjer kommunikasjonen ved hjelp av satte kommandoer. (Rossen 2020)

2.4 Design

2.4.1 Trådramme

En trådramme er en enkel skisse av selve «skjelettet» til nettsiden. Den skal vise det grunnleggende ved nettsiden som strukturen, utformingen, funksjonaliteten og navigasjonen til en nettside (Hannah 2021). Trådrammen tegnes vanligvis i gråskala og skal derfor ikke vise fram stylingen, man bare det viktigste. Dette er et veldig vanlig verktøy innen web-design som brukes for å enkelt visualisere ideer og komme fram til et endelig design. Siden skissene ikke har for mange detaljer, egner de seg godt i planleggingsfasen til prosjekter der det kan forkomme endringer underveis. Det er viktig med trådrammer fordi det gir utviklerne og oppdragsgiver en enkel måte å jobbe sammen med tanke på hvordan de ønsker nettsiden skal være. Et eksempel på hvordan trådrammer kan se ut er illustrert i Figur 4.



Figur 4: To trådrammer som er knyttet sammen

2.4.2 Brukersentrert design

Brukersentrert design handler om å lage produkter som er intuitive og enkle å bruke for alle. Dette gjelder ikke bare for web-design, men også fysiske gjenstander og produkter. Det handler om hva man bør tenke på under utviklingen og hva som er gode og dårlige sider med tanke på interaksjon for brukeren.
Et sentralt konsept innen brukersentrert design er Don Norman's 6 prinsipper:

- **Visibility**
  Dette handler om at det skal være enkelt å finne ting. Finner man ikke en funksjon enkelt blir den sannsynligvis ikke brukt. Samtidig kan du ikke vise alt samtidig, siden det kan bli rotete.

- **Feedback**
  Dette handler om å gi brukeren en indikasjon på at noe har skjedd. Det kan være for eksempel en knapp som gråes ut hvis noe gjør at den ikke kan trykkes på eller en dialogboks som sier ifra når du gjør noe.

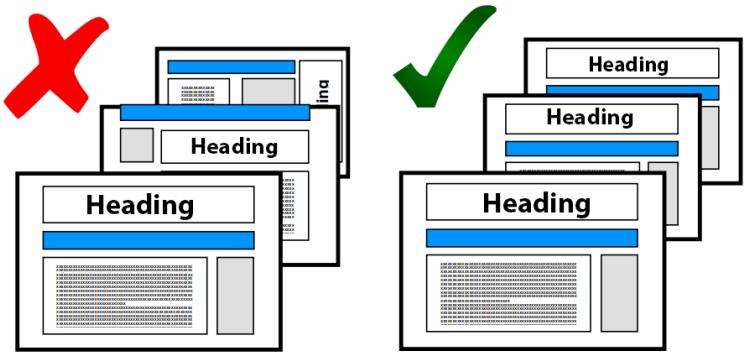
- **Constraints**
  Dette handler om å ikke gjøre brukergrensesnittet for komplekst. Det kan man oppnå ved å begrense samtidige valgmuligheter så det er enklere å forstå hva som kan gjøres. Ifølge «Hick's Lov» så øker beslutningstiden logaritmisk med antall valg. (Hick's law 2021)

- **Mapping**
  Dette handler om å ha innlysende sammenhenger mellom elementer og hva de faktisk gjør. Brukeren skal enkelt kunne forstå hvilken handling fører til den han vil gjøre. Et eksempel kan være scroll bar. Det er god mapping hvis nettsiden går i samme retning som du drar scroll-baren, og er det motsatt blir det forvirrende.

- **Consistency**
  Dette handler om å bruke like elementer for lik funksjonalitet. Det gjør det enklere for brukeren, som slipper å undersøke om et element bare ser annerledes ut eller om det faktisk har en annen funksjonalitet. Et område som det kan være smart å være konsekvent på er farger på knapper eller layout av nettsiden. Se Figur 5 for eksempel på dette prinsippet.

- **Affordance**
  Dette handler om at elementer bør gi brukeren en indikasjon på hva dens funksjon er. Spesielt viktig er dette hvis funksjonaliteten ikke er selvinnlysende av seg selv. Et eksempel på dette er lenker i en tekst. Hvis den ikke ser annerledes ut enn resten av teksten vil ikke brukeren oppdage at den er klikkbar.

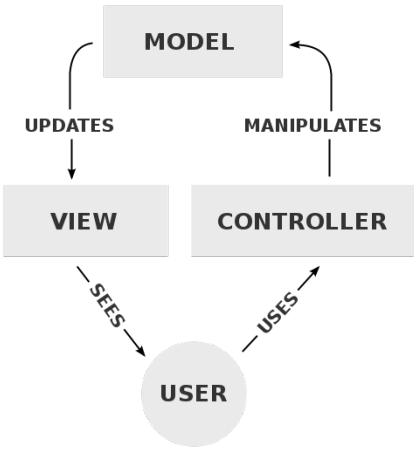
(Rekhi 2018)



Figur 5: Eksempel som illustrerer forskjellen mellom dårlig og god consistency mtp. sideoppsett

2.4.3 Model-Visning-Kontroller Designmønster

MVC *(engelsk: Model View Controller)* designmønsteret deler et program inn i tre separate deler: modell, visning og kontroller. Designmønsteret er illustrert i Figur 6. Dette designmønsteret er ikke egentlig for en hel applikasjon, men illustrerer heller den delen som en bruker er i kontakt med, dvs. front end. For en 'full stack' web applikasjon er det vanlig å ha flere deler/lag som tjenestelaget (service) og data-tilgang-laget (data access). (Kumar 2017)



Figur 6: Model View Controller designmønster. (RegisFrey 2010)

Modell

Ansvarlig for applikasjons-data. Den trenger ikke vite noe om hvordan informasjonen skal presenteres for brukeren i brukergrensesnittet.

Visning

Ansvarlig for å vise fram informasjonen som er hentet fra Modellen til brukeren. Dette laget trenger ikke vite noe om forretningslogikken til applikasjonen eller noe om informasjonen selv, bare hvordan den skal vises fram.

Kontroller
Ansvarlig for å fange opp handlinger som brukeren gjør og sende kommandoer videre til logikken i programmet. Dette laget ligger mellom Visning og Modell og kaller vanligvis funksjonalitet i Modell-laget.

Fordeler & ulemper

En stor fordel ved å bruke MVC designmønsteret er at programmet får lav kobling mellom de forskjellige lagene; de trenger ikke vite hvordan andre lag fungerer, bare hvordan de kan tas i bruk. Hvis man beholder input og output av et lag, kan man endre på den indre funksjonaliteten uten at det starter en kaskade av endringer som må gjøres i resten av programmet.

En annen fordel er at det gjør det enkelt å jobbe som utviklere i en gruppe, da flere personer kan jobbe i forskjellige lag samtidig. De trenger bare vite hva de andrelagene forventer å motta og hva de kan sende i retur.

En ulempe kan være at prosjektet blir komplekst. Blir det mange lag med forskjellige ansvarsområder kan det bli mye å sette seg inn i. En person som ikke er vant til å bruke dette designmønsteret for prosjekter blir nødt til å sette seg inn i konseptet før han starter.

2.5 Relasjondatabase

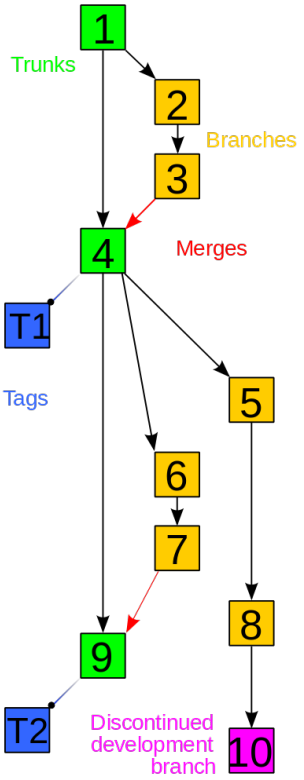
En relasjonsdatabase blir brukt for å lagre informasjon der data blir organisert som rader i tabeller inni en database. I en database vil data bli lagret som attributter der en av attributtene blir lagret som primærnøkkelen. En primærnøkkel må være unik. Data vil også være en tydelig identifiserbar enhet samt vil den ha relasjoner til relevante data fra andre tabeller. Denne koblingen mellom data og relasjoner realiseres gjennom en verdikopling. I en slik kobling vil det bli skapt en ny tabell som kobler sammen primærnøkkelen fra begge tabellene for å vise relasjonen mellom dem. (Bratbergsengen 2019)

2.6 Versjonskontroll

Versjonskontroll er et verktøy som er veldig mye brukt innen IT-utvikling. Målet er å kunne samarbeide om å utvikle kode, spore endringer enkelt og unngå vanskelige konflikter. Versjonskontroll har også oversikt over historien til koden og gjør det tryggere å eksperimentere siden du ikke trenger å bekymre deg for å ødelegge noe permanent. Utviklerne i en gruppe kan arbeide på egne kopier av koden og deretter slå den sammen med resten etter hvert for å unngå konflikter underveis. Og hvis det oppstår konflikter får du beskjed om hvor problemet er, så utviklerne kan løse det. Siden historien er lagret er det enkelt å gå tilbake for å se hvorfor konflikten har oppstått.

Ofte kan versjonskontroll-systemet også automatisk fikse enkle kode-konflikter. Versjonskontroll er heller ikke bare et godt verktøy for grupper, men også individuelle som ønsker bedre oversikt over utviklingen og endringer over tid. Figur 7 illustrerer hvordan et versjonskontrollert prosjekt med flere grener kan se ut.

De mest populære versjonskontroll-verktøyene er Git, Subversion og Mercurial. Git er det desidert mest brukte av de tre.
(GitLab 2021)



Figur 7: Eksempel på versjonskontrollert prosjekt (Echion2 2010)

2.7 Prosjektplanlegging/gjennomføring

Prosjektplanlegging er en av de første delene av et prosjekt der vi må velge hvilke verktøy vi trenger, hvor lang tid vi skal bruke på prosjektet, hva vi trenger å få gjort, hvordan resultatet skal bli og mer. For å få gjort dette blir det satt opp en Kravspesifikasjon, trådrammer, flytskjema og lignende.

2.7.1 Kravspesifikasjon

Kravspesifikasjon beskriver et programvaresystem som skal bli laget og etablerer grunnlaget for en avtale mellom arbeidsgiver og leverandøren. Den legger frem både funksjonelle og ikke-funksjonelle krav til programvaren. Kravspesifikasjon kan også inneholde noen brukertilfeller som beskriver hva en bruker skal kunne gjøre i systemet. (Software requirements specification 2021)

2.7.2 Flytskjema

Flytskjema er et diagram som bruker bokser og piler for å beskrive en prosess i et system. Boksene er et steg mens en pil binder sammen boksene i en satt rekkefølge. En boks vil da ha et spørsmål eller en handling og pilen vil peke der dette spørsmålet eller handlingen fører til.

2.8 Virtualisering

Virtualisering er å ta i bruk teknologi for å lage et virtuelt miljø
hvor det er mulig å kontrollere og fordele ressurser, skape et miljø
som er lett å reprodusere, og skape et stabilt miljø som er stabilt for programvare. Programvaren vil forholde seg til en virtuell prosessor og en virtuell lagringsenhet slik at det er mulig å ha full kontroll over hvor mye ressurser som er tilgjengelig i det virtuelle miljøet. virtualisering – IT 2020

2.9 Cross-Origin Resource Sharing (CORS)

CORS er en mekanisme som brukes for å øke sikkerheten til nettsteder. Gjennom CORS bestemmer en server hvilke kilder den vil ta imot informasjon fra. Hvis en server trenger å motta informasjon fra en kilde som er utenfor sitt eget domene/port, så må serveren være konfigurert slik at den tillater å laste inn data fra det domenet/porten den vil motta fra. Cross-Origin Resource Sharing (CORS) - HTTP — MDN 2021

2.10 Agile metoder

Agile metoder er en betegnelse som blir brukt innenfor programvare utvikling. Å jobbe med agile metoder betyr å være fleksibel og smidig når det kommer til planlegging og utvikling. Et kjennetegn til Agile metoder er at de oppfordrer til å tilpasse planer fortløpende, tidlig levering av ett fungerende produkt, kontinuerlig forbedring og fleksibilitet framfor endringer (Agile software development 2021).

Agile metoder innenfor software utvikling ble først popularisert av *The Manifesto for Agile Software Development* (Manifesto for Agile Software Development 2021). I manifestet som ble signert framheves fire verdier:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

De verdiene som står i vanlig skrift er ikke verdiløse, men verdiene som står i fet skrift har mer verdi, ifølge manifestet. Altså når en jobber med agile metoder så verdsetter man samhandling med team, levere et fungerende produkt, samhandling med kunde, og fleksibilitet framfor endringer.

Det er flere rammeverk som bruker agile metoder som hovedfokus. De to mest kjente og brukte rammeverkene er Scrum og Kanban. I dette prosjektet brukte vi Scrum.

2.11 Scrum

Scrum er et smidig prosessrammeverk som tar i bruk agile metoder for å hjelpe folk, teams og organisasjoner med å jobbe sammen for å utvikle et produkt. Scrum er en av de mest brukte agile rammeverkene og brukes som oftest innenfor programvare utvikling, men metodikken kan anvendes for all slags lagarbeid. Scrum implementerer empiriske prosesser, som betyr at en jobber iterativt ved å trinnvis levere anvendbar verdi til oppdragsgiveren.

Scrum har også fokus på å erstatte den algoritmiske tankegangen med en heuristisk en. Altså istedenfor å ha faste rammer og liten fleksibilitet, så fokuserer Scrum metodikken at folk i ett team selv-organiserer for å håndtere uforutsigbarhet og komplekse problemer på en fleksibel måte. (Schwaber og Sutherland 2021)

2.11.1 Scrum team

I et prosjekt som bruker Scrum-rammeverket så gjøres arbeidet av ett Scrum team. I ett team er det tre forskjellige roller:

- Utviklingsteam
- Produkteier
- Scrum Master

Scrum teamet burde ikke være større enn ti personer, ettersom det kan forårsake problemer med kommunikasjon. Teamet burde være en liten nok størrelse slik at de kan være fleksible ovenfor endringer, men fremdeles være stor nok til å kunne få gjort betydningsfulle mengder med arbeid.

2.11.1.1 Utviklingsteamet

Det er utviklingsteamet sin oppgave å utvikle alle aspekter av prosjektet og sørge for at hver sprint så har prosjektet kommet seg lengre enn den forrige iterasjonen av prosjektet. Utviklingsteamet skal lage en plan for sprinten, tilpasse seg endringer fra dag til dag og sørge for at godt arbeid blir gjort.

2.11.1.2 Produkteieren

Det er produkteieren sin oppgave å sørge for å maksimere verdien til arbeidet som blir gjort i Scrum teamet. Ofte så representerer produkteieren en eller flere interessenters ønsker for prosjektet. For at disse ønskene skal oppnås må produkteieren:

- Skape og definere ett klart og tydelig mål for sprintene
- Skape og definere forståelige produkt backlog oppgaver
- Rangere hvilke oppgaver i produkt backloggen som er viktige og hvilke er mindre viktige

2.11.1.3 Scrum Master

En Scrum Master er ansvarlig for all ting Scrum. De har ansvaret for å holde Scrum gående på riktig vis, ansvaret for at alle vet hvordan Scrum fungerer i teori og praksis og ansvaret for Scrum teamet sin effektivitet. Alt dette oppnås ved at Scrum Masteren:

- Prøve å fjerne hindringer for teamet sin framgang
- Forsikre seg om at alle Scrum handlinger tar plass og gjennomføres riktig
- Hjelpe de i teamet om de trenger hjelp i forhold til gjennomføring av Scrum handlinger
- Hjelpe å formidle hva som er viktig og hva som bør prioriteres
- Hjelper til slik at produkt backloggen er klar og tydelig

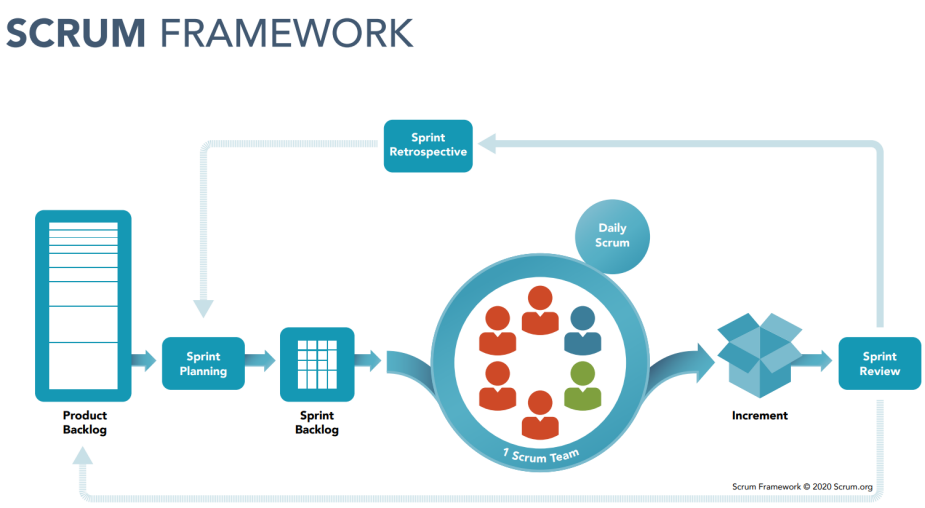
(Schwaber og Sutherland 2021)

2.11.2 Scrum handlinger

I scrum så går teamet gjennom 5 viktige handlinger.

- Backlog raffinering
- Sprint
- Sprint planlegging
- Sprint gjennomgang
- Sprint tilbakeblikk

Bortsett fra backlog raffinering så er alle de andre handlingene møter som teamet har. De er faste møter, med et klart og tydelig mål, som varer en fast mengde med tid. (Scrum Ceremonies and Artifacts 2020)



Figur 8: Flytskjema for Scrum rammeverket.

2.11.2.1 Backlog raffinering

Backlog raffinering handlingen gjøres en gang helt i starten av et prosjekt, og handler om å fylle opp produkt backloggen til prosjektet. I produkt backloggen ligger alt som skal bli utviklet for prosjektet, og skal inneholde nye funksjonaliteter, bug fikser, optimaliseringer eller hva som helst annet som prosjekteier vil tjene av. Dette gjøres ut ifra blant annet gitte kravspesifikasjoner og samarbeid og diskusjon med produkt eier. Produkt backloggen er i konstant endring gjennom prosjektets levetid og vil bli fylt inn ettersom teamet får tilbakemeldinger, nye idéer, møte resultater, markedsendringer og så videre. (Scrum Ceremonies and Artifacts 2020)

2.11.2.2 Sprintene

Sprintene i et prosjekt er hvor idéer blir gjort om til verdifulle løsninger. En sprint kan vare fra en uke til en måned, og en ny sprint blir startet rett etter den forrige blir ferdig. I starten av en sprint vil teamet gå gjennom en sprint planleggings fase, hvor de definerer et mål for sprinten og definerer arbeidet som skal bli gjort. I slutten av hver sprint så skal teamet ha et produkt som er en forbedret versjon av den fra forrige sprint, som de skal vise fram til oppdragsgiveren.

2.11.2.3 Sprint planlegging

Sprint planlegging er det første som skjer i en sprint. Hær definerer teamet hva målet med sprinten er, og hvilket arbeid som kan bli utført for å nå dette målet. Etter at målet for sprinten er satt så vil teamet gå gjennom produkt backloggen å se etter oppgaver som vil hjelpe til med å oppfyllet målet som ble satt. Det er normalt å endre og justere disse oppgavene for å klarere og øke forståelse for hva som trenger å bli gjort. Disse oppgavene blir da lagt til i sprint backloggen.

2.11.2.4 Sprint gjennomgang

En Sprint gjennomgang blir gjort i slutten av en sprint før teamet gjør sprint tilbakeblikket. I gjennomgangen så ser teamet på utfallet til sprinten sammen med oppdragsgiveren og sammen så bestemmer de om noen tilpasninger trengs. Teamet viser fram den nyeste iterasjonen av produktet deres til oppdragsgivere og eventuelt andre nøkkel interessenter, får tilbakemeldinger og diskuterer hvordan teamet skal gå videre med prosjektet. På denne måten får teamet ofte tilbakemeldinger på deres arbeid, og jobber fleksibelt rundt hva oppdragsgiveren faktisk er ute etter.

2.11.2.5 Sprint tilbakeblikk

I et sprint tilbakeblikk så diskuterer teamet hvordan sprinten gikk. I tilbakeblikkene så skal teamet diskutere hva som gikk bra og hva som gikk dårlig med tanke på individer, interaksjoner mellom individer, prosesser for utføring av sprinten og verktøy som ble brukt. Tilbakeblikket handler om å identifisere hva som var hindringer for sprintens suksess og deretter finne løsninger på hvordan unngå hindringene i de neste sprintene. Det er også viktig å ta opp det som gikk bra i sprinten, og hvordan problemer ble løst. (Schwaber og Sutherland 2021)

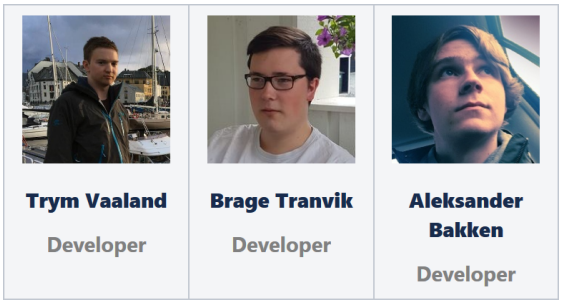
3 Materialer og metode

3.1 Prosjektorganisasjon

3.1.1 Prosjektgruppen

Alle på prosjekt gruppen er avgangsstudenter på studiet Bachelor i Ingeniørfaglig data ved NTNU Ålesund og består av følgene studenter som også vist på figur 9:

- Trym Vaaland
- Brage Tranvik
- Aleksander Bakken



Figur 9: Prosjektgruppen.

3.1.2 Oppdragsgiver

Oppdragsgiveren vår er Luleå universitet i Sverige der Lars Hansson har vært vår kontaktperson. Lars er professor innen tre-forskning og ingeniørfag ved Luleå universitet.

3.1.3 Veileder

Veilederen vår er Arne Styve. Arne er universitetslektor for institutt for IKT og realfag. Han er utdannet som Sivilingeniør og har mer enn 20 års erfaring innen norsk IT-industri.

3.1.4 Sprint og møter

Vi bestemte oss for å arbeide i 1 ukes sprinter etter ønske av veilederen vår. Sprintene skulle starte på mandag og slutte på fredag. Sprintplanleggingen skulle vi gjøre på mandag morgen. Sprint gjennomgang og sprint tilbakeblikk hadde vi planlagt å gjennomføre hver fredag. I lag med oppdragsgiver og veileder bestemte vi også oss for at vi skal ha møte med oppdragsgiver og veileder annenhver uke for å vise frem ny funksjonalitet.

3.1.5 Prosjektorganisering

I prosjektet har vi brukt agile metoder og jobber etter Scrum. Scrum er beskrevet i kapitel 2.11. Dette var et krav av veilederen vår og siden vi også ville holde prosjektet organisert valgte vi å bruke Scrum. Vi ga ingen på gruppen rollen som Scrum master, så denne rollen brukte vi ikke. Ellers så er Trym, Brage og Aleksander utviklere, mens Lars som er oppdragsgiver produkteieren. Uansett om vi ikke valgt en Scrum master hjalp veilederen vår oss med å arbeide etter Scrum der vi fikk tilbakemeldinger om vi gjorde noe feil. For møtene var Trym møteleder mens Aleksander og Brage vekslet mellom å være sekretær og vise frem resultater.

3.2 Planlegging

Til planlegging av vi hovedsaklig brukt Jira og Confluence, men vi har også brukt dbdiagram.io for planlegging av database og brukte Microsoft Teams til litt planlegging i starten av prosjektet.

3.2.1 Prosjektstyringsverktøy

3.2.1.1 Jira

Jira er et produkt for å organisere og styre prosjekter. Jira brukes til å gjennomføre prosjekter med Agile metode. Noen av funksjonaliteten i Jira for utvikling er: Sporing av issues, estimering av arbeid, opprette rapporter, Scrum brett, backlog og organisere sprinter. (Atlassian 2021)

Det var allerede bestemt fra starten av at vi skulle bruke Jira av veilederen. Vi har brukt Jira til å lage brukerhistorier, oppgaver, sprint og utgivelser. Brukerhistorier har vi brukt til å planlegge en oppgave som skal være mulig å utføre på applikasjonen. For eksempel en brukerhistorie kan være: *Som en bruker skal jeg kunne opprette et nytt prosjekt.* Denne historien kan da inneholde flere oppgaver som kreves for å utføre historien, som for eksempel: *Lag en REST forespørsel til å*

opprette et nytt prosjekt. Vi har også laget oppgaver utføre en brukerhistorie om oppgaven ikke passer som en historie. Et eksempel på dette er: *Lag fil ikoner til nettsiden*. Forskjellen er altså en brukerhistorie skal inneholde en full funksjonalitet som en bruker skal kunne utføre når den er ferdig, mens en oppgave er bare en enkel funksjonalitet eller oppgave som skal bli gjort.

Vi har også brukt Jira til sprint og utgivelseplanlegging. På sprintplanlegging har vi satt av litt tid på mandags morgen for å bestemme oss for hva vi skal få gjort den uken. Vi tar da de viktigste funksjonalitetene og legger de innpå sprinten. Det er da disse oppgavene vi jobber med resten av uken. Utgivelseplanlegging derimot har vi tatt tiden vi har igjen og planlagt en utgivelse hver andre uke. Grunnen til vi valgte annenhver uke er slik at vi har en utgivelse klar til møte med arbeidsgiver og veileder. Når vi har planlagt utgivelser har vi tatt og fordelt brukerhistorier og oppgaver på alle utgivelsene, der vi tar de viktigste funksjonalitetene på de første.

3.2.1.2 Confluence

Confluence er et wiki-sidesystem og arbeidsområde for grupper. Det skal være en sentral plass der man kan dele informasjon effektivt og det er bra integrert med Jira. Confluence har funksjonalitet for samarbeid om redigering av sider og inkluderer maler for ting som møtenotater, sprint tilbakeblikk og UML diagram.

Vi har bruk Confluence til det meste av planlegging til prosjektet; hvilken funksjonalitet som kreves, hvordan nettsiden skal se ut og hvordan nettsiden skal fungere. I tillegg har vi brukt det underveis i utviklingen for å notere og logge hva vi har gjort.
Her er en full liste:

- Kravspesifikasjon
- Trådrammer
- Flytskjema
- Møtenotater
- Andre notater
- Sprint tilbakeblikk
- Sprint gjennomgang
- Diagrammer

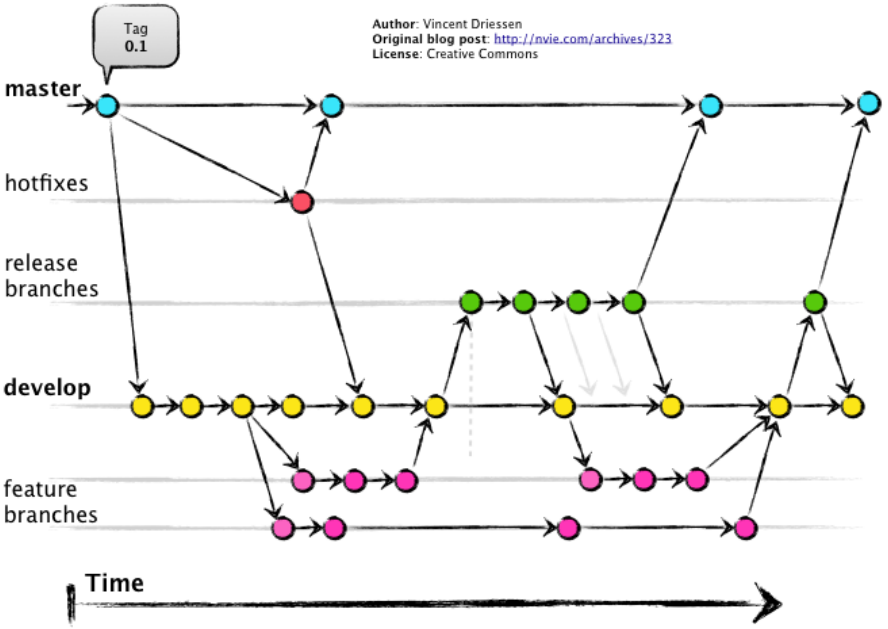
3.2.1.3 GitHub

GitHub er nettbasert tjeneste som lar personer lagre og dele kode på en plattform med versjonskontroll. Som navnet foreslår, brukes Git, den mest brukte formen for versjonskontroll. På GitHub kan brukere laste opp repositories, samarbeide, og ta imot forslag fra andre utviklere.

Vi har opprettet og brukt to repositories for dette prosjektet. Ett for frontend og ett for backend. Under utviklingen satte vi de som private så bare de med tillatelse kunne lese/endre koden. GitHub har funksjonalitet for oppgaver (issues), men vi benyttet oss ikke av den og brukte Jira i stedet.
For både backend og frontend startet vi med `.gitignore` generert for IntelliJ og WebStorm fra gitignore.io. Etterhvert har vi utvidet filene med det som trengs for rammeverk og andre verktøy som Docker, React og npm.

Git Flow
På starten av prosjektet bestemte vi oss for å bruke *Git Flow* til prosjektet. Vi hadde brukt denne metoden tidligere og hadde erfart at det var en grei måte å jobbe på og gjorde arbeidet oversiktlig. Måten vi implementerte *Git Flow* var lik som illustrert av Figur 10.

Med Git Flow så jobbet vi underveis med utgangspunkt ut fra develop. Mens vi utviklet en spesiell funksjonalitet eller jobbet med en *User Story* brukte vi separate greiner. Når den begynte å bli ferdig startet vi en «pull request», så de andre kunne kontrollere koden og se etter feil eller komme med forslag på endringer. Samtidig som den er åpen kan vi komme med nye commits. En godkjent *pull request* ble så ført inn i *develop* etter at eventuelle kode-konflikter ble løst. Når vi er klar for en release så dyttes develop inn på en release grein. Vi har ikke brukt å komme med flere commits på release greinen. Etter det satte vi på versjons-taggen og dyttet det over på main-greinen. Det er den greinen vi har tatt fra når vi har bygget prosjektet for å vise fram til oppdragsgiver.



Figur 10: Eksempel på bruk at git-flow i software-prosjekt. (Driessen 2010)

3.2.1.4 GitKraken

GitKraken er et versjonskontrollverktøy med et grafisk brukergrensesnitt (GUI). Vi har valgt å bruke GitKraken for dens enkle GUI og hjelp med å bruke *Gitflow*. Som studenter har vi også tilgang til *Pro* versjonen. Dette gir oss tilgang til å bruke GitKraken på private prosjekter.

1. Gitflow har vi brukt til å lage nye greiner med en funksjonalitet eller lage en ny utgivelse av applikasjonen. Gitflow må bli manuelt skrudd på inni innstillingene til GitKraken for å fungere.

2. Remote er en liste over alle greiner vi har i prosjektet på Git repositoryet. I vårt prosjekt har vi valgt å starte hver grein sitt navn med nummeret på brukerhistorien eller oppgaven den hører til med.

3. Dette er en visualisering av Gitflow treet. Her kan vi se hvem som har sendt ut en endring til Git repositoryet, hvilken grein den er fra og om to greiner er slått sammen.

4. Denne menyen har en del nyttige funksjoner. De vi har brukt er *Pull*, *Push*, *Stash* og *Pop*.

   (a) Pull bruker vi til å hente nye endringer av koden som har blitt sendt ut til den greinen vi er innpå.

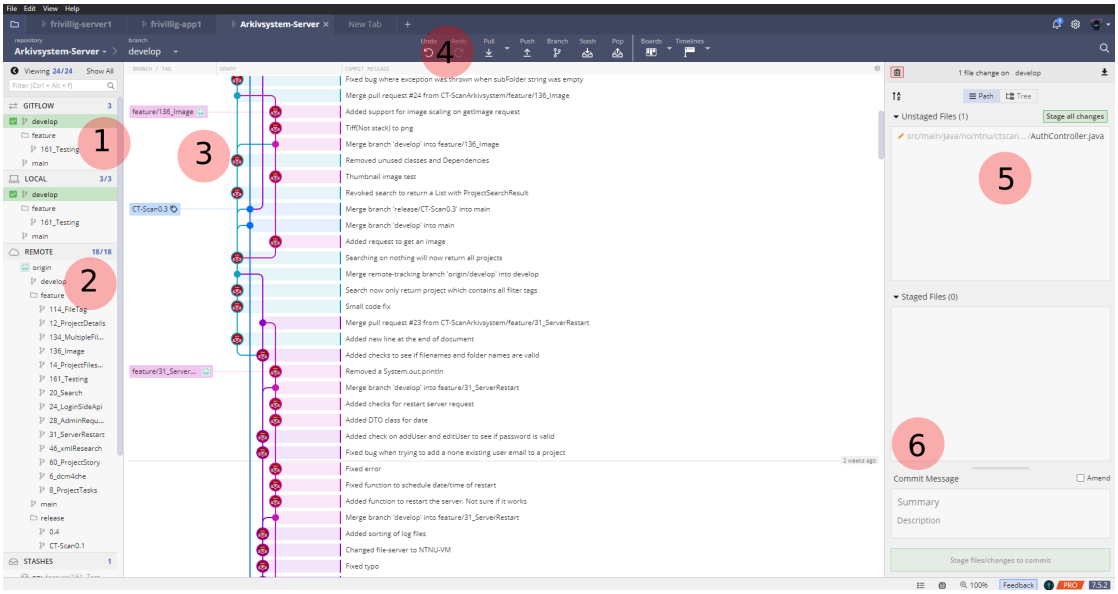
(b) Push bruker vi til å sende ut nye endringer i koden som vi har jobbet med på greinen vi er innpå.

(c) Stash bruker vi får å legge til side endringer lokalt om vi ikke er klare til å bruke *Push*, eller vi er usikre om vi skal beholde endringene.

(d) Pop bruker vi til å få tilbake endringene fra en *Stash* og fjerne den. Om vi ikke vil fjerne den kan vi også høyre klikke på en *Stash* og bruke *Apply*.

5. Unstaged Files viser endringene vi har gjort i koden. Som nye filer eller endringer i en eksisterende fil.

6. Staged Files er det samme som *Unstaged Files* utenom filene her er klar til å bli sendt ut til repositoryet. Vi må også sette på en tittel med en valgfri tekst.



Figur 11: GitKraken GUI

3.2.1.5 Dbdiagram.io

Dbdiagram.io har vi brukt til planlegging av database, hvilke variabler vi skal ha i de forskjellige database klassene i Java og forholdet mellom dem. Mer om hvordan nettsiden fungerer i kapitel 3.3.3.

3.2.2 Kravspek og møtenotater

Kravspek har vi brukt til å holde litt orden på hva funksjonalitet applikasjonen trenger. Vi laget et kravspek helt i starten av prosjektet der vi la inn mål med prosjektet, krav, hva en bruker skal kunne gjøre og spørsmål vi stilte med svarene. Etter en stund lagde arbeidsgiver og veileder et nytt kravspek til oss som vi har brukt videre. Vi har brukt kravspeke til å se igjennom om vi lurer på om det er noe vi har gløymt eller trenger. Møte notater har for det meste blitt brukt til å huske hva vi har snakket om i et møte, men vi har også brukt det til å sette opp avkrysningsbokser med forskjellig vi må huske å gjøre.

3.2.3 Trådrammer

Trådrammer har vi brukt til å planlegge brukergrensesnittet til nettsiden. Her har vi laget en tegning over hvordan vi tenker en side skal se ut slik at vi har noe å jobbe mot for å gjøre arbeidet med å lage nettsiden enklere og spare tid. Eksempel på hvordan vår trådramme ser ut på figur 4.

3.2.4 Flytskjema

Vi har brukt flytskjema for å få en oversikt over hvordan man skal navigere seg på nettsiden. For eksempel, om du er på en side så kan vi bruke flytskjema for å vite alle sider du kan komme deg til fra den siden.

3.3 Utviklingsverktøy

3.3.1 IntelliJ Idea

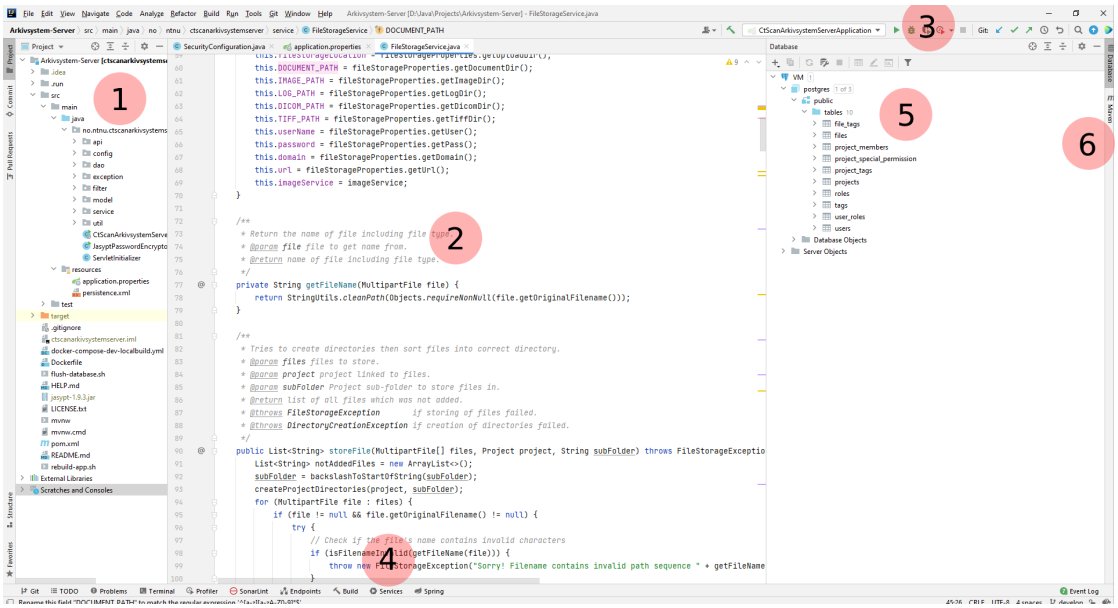
IntelliJ er et integrert utviklingsmiljø for Java som gir oss tilgang til alt fra auto-complete for kode, auto-importering, tilkobling til database og mer. Dette er bare noen av grunnene til vi valgte å bruke IntelliJ. Vi har også en del erfaring med IntelliJ fra før. Siden vi også viste at vi skulle bruke Docker til prosjektet og IntelliJ har god støtte for å kjøre Docker, så gjorde det valget enklere.

1. Dette er mappe strukturen til prosjektet.
2. Dette er editoren.
3. Her kan vi starte et prosjekt eller endre hva vi vil kjøre prosjektet med.

4. Service bruker vi til Docker. Her kan vi starte eller stoppe Docker konteinere med satte yml skript. Vi kan også bruke service til å se outputen til konteineren.

5. Helt til høyre kan vi velge Database. Da vil vi få opp dette vinduet som vi kan koble oss opp mot en database for å kunne se innholdet og kjøre SQL-queries til databasen.

6. Maven bruker vi til å pakke prosjektet til en war-fil som bruker for å kjøre prosjektet.



Figur 12: IntelliJ Idea

3.3.2 WebStorm Ide

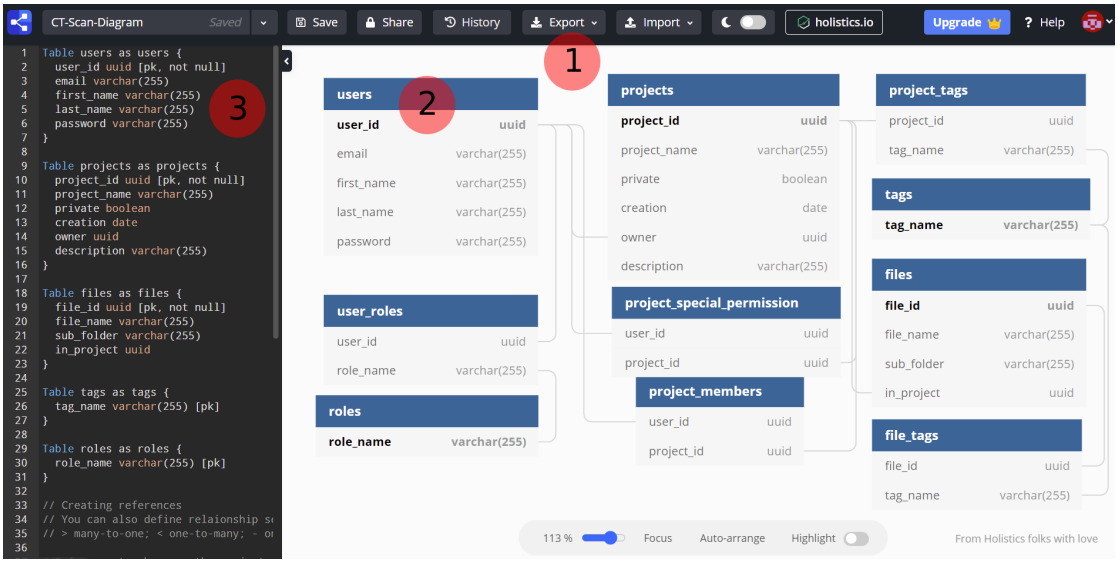
WebStorm er et integrert utviklingsmiljø som er designet for å bli brukt med JavaScript, biblioteker og rammeverk til JavaScript og generelt all ting nettside relatert. WebStorm gir forslag for kode når du skriver, hjelper til med auto-importering, den sjekker for skrivefeil, for mulige feil eller overflødige kode og mer.

I starten av prosjektet når vi først begynte på frontend så ble IntelliJ brukt, men etter en vi ut at samme selskapet som lagde IntelliJ, som heter JetBrains, også hadde lagd en IDE som fokuserer på JavaScript og som støtter React, så byttet vi over til WebStorm for frontend utvikling. Editorene er ganske like og det var derfor veldig enkelt å bytte fra den ene til den andre og vi møtte på ingen problemer.

3.3.3 dbdiagram.io

Dbdiagram.io har vi brukt for utvikling av database tabeller, relasjonene mellom dem å generere SQL koden for PostgreSQL.

1. Export kan bli brukt for å generere SQL koden for både MySQL og PostgreSQL.
2. Dette er en visuell framvisning av koden som tabeller og relasjonene mellom dem.
3. Dette er kode vinduet for å lage diagramene.



Figur 13: dbdiagram.io

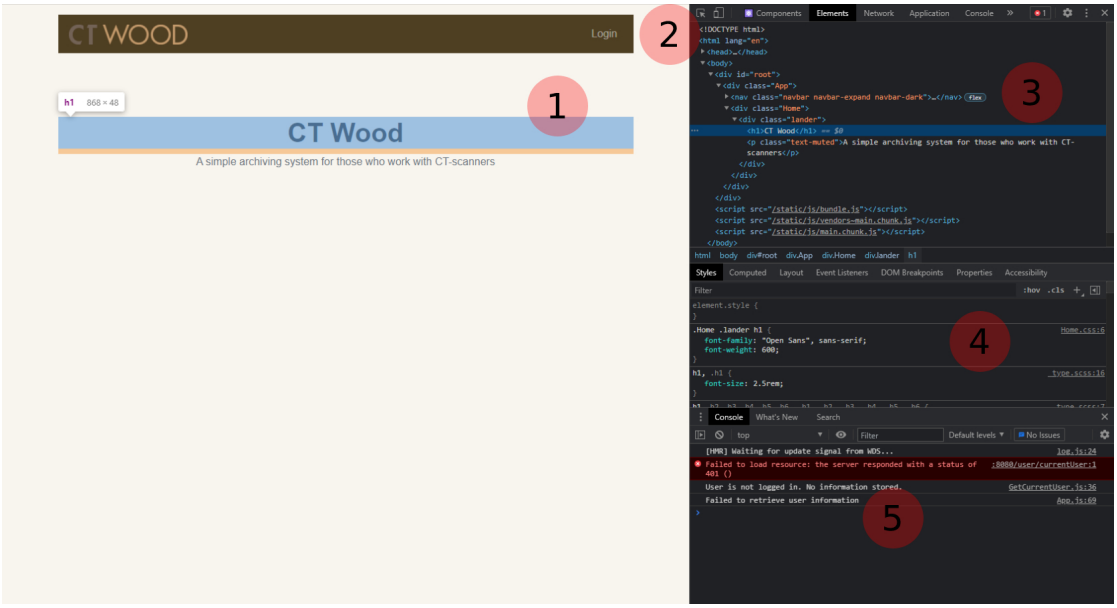
3.3.4 Nettleser utviklingsverktøy

I de fleste nettlesere finnes det ett sett med web utviklingsverktøy. Disse verktøyene brukes vanligvis for å feilsøke problemer og utvikle funksjonaliteter på nettsider. For det meste så brukte vi nettleseren Google Chrome for å teste nettsiden, og brukte dermed Google Chromes innebygde utviklingsverktøy.

1. Nettsiden. Boksen som er markert, viser ett valgt elements plassering.
2. Forskjellige tabber med diverse funksjoner. Her er de tabbene som ble brukt og hva de ble brukt til:
    - Components: Se react komponenter og deres state og props.

- Elements: Tabben som vises.
- Network: Se API forespørsler og deres detaljer.
- Application: Se lagrede cookies til nettstedet.

3. Her er elementene på nettstedet. Det er mulig å endre på elementene eller legge til nye.
4. Her vises stilene til elementene på nettstedet. Det valgte elementet få dens stil vist hær. Det er mulig å endre på stilen til elementet og legge til nye stiler til det.
5. Konsollen. Vanligvis vises forskjellige feilmeldinger i konsollen. Det er også mulig å sende egendefinerte meldinger gjennom kode. Det som er rødt er en feilmelding, det som er hvit er en melding som blir skrevet av kode. Det kan også komme gule varsel meldinger som kan advare mot noe som kan være feil.



Figur 14: Google Chrome sitt utviklingsverktøy

3.4 Distribuering av applikasjon

Fra starten av prosjektet viste vi at prosjektet måtte være i stand å kjøre på en virtuell maskin. Applikasjonen skulle være enkel å sette opp og kjøre om den skal bli videreutviklet og brukt i framtiden. For å oppnå dette trengte vi en virtuell maskin å teste på, Docker for å kunne enkelt starte applikasjonen uansett miljø og maven til å pakke prosjektet.

3.4.1 Virtuell maskin

For virtuell maskin brukte vi *Ubuntu 18.04* fra autodeploy.uials. Her kunne vi enkelt opprette virtuelle maskiner for testing, og om noe gikk galt kunne vi enkelt ta en omstart på dem. På den virtuelle maskinen installerte vi Docker, docker-compose og git for å kunne klone prosjektet fra Github og kjøre det på maskinen ved hjelp av docker-compose.

3.4.1.1 SMB-Filserver

Til filserver brukte vi SMB. SMB også kalt Samba er en gratis programvare som bruker SMB-nettverksprotokoll. Denne programvaren gjør det enkelt å koble seg til en filserver med funksjoner som Windows sin *Koble til nettverksstasjon* for å kunne se og laste opp filer manuelt. Applikasjonen vår forventer denne type filserver siden den bruker et bibliotek som bruker SMB-filserver (Mer om dette i kapittel 3.6.10). Vår løsning krever altså en SMB-filserver på en virtuell eller vanlig maskin for å kunne laste opp og ned filer. I starten av prosjektet fikk vi bruke en slik filserver som tilhørte LTU, men etter problemer som gjorde at vi ikke fikk lov å bruke den filserveren lengre måtte vi finne et alternativ. Vi valgte derfor å sette opp en ny virtuellmaskin på autodeploy.uials som vi lastet ned SMB-programvaren på for å bruke som en filserver.

3.4.2 Maven

Maven brukte vi til å handtere alle avhengighetene vi brukte i prosjektet gjennom en fil som heter: *pom.xml*. Liste over avhengighetene vi brukte i kapittel 3.6. Med Maven kan vi installere rammeverk og biblioteker og bygge en *.war* fil av prosjektet. Denne filen er en samling av Java filer som vi brukte til å kjøre prosjektet. Med Maven kan altså hver gruppe medlem enkelt ta inn et bibliotek som også vil bli lastet ned for de andre gruppe medlemmene.

3.4.3 Docker

Docker er en programvare som bruker virtualisering på operativsystems nivå for å kjøre konteinere. En konteiner er en isolert pakke med egne programvarer, biblioteker og konfigurasjonsfiler. Konteinere kan kommunisere med hverandre ved hjelp av veldefinerte kanaler. Alle konteinere deler operativkjerne som gjør at de bruker mindre ressurser enn virtuelle maskiner. Docker (software) 2021

Docker compose er et verktøy som brukes for å definere og kjøre flere konteinere samtidig. Compose bruker YAML fil for å definere applikasjonens serviser, plattformer og konfigurasjonsalternativer. Ved hjelp av en enkel kommando kan du starte, stoppe eller bygge flere konteinere samtidig. Docker (software) 2021

For å gjøre det enkelt å kjøre applikasjonen uansett miljø brukte vi Docker og docker-compose. For backend fant vi et Docker skript som installerer maven, og deretter bruker maven til å hente alle avhengighetene vi trenger for å kjøre applikasjonen og bygger opp prosjektet som en *.war* fil. Vi bruker også Docker skriptet til å sette opp en Tomcat web server der vi kopierer over *.war* filen som ble bygget. Deretter bruker vi et docker-compose skript også kalt *yml* til å starte tre forskjellige tjenester: en for database, en for REST serveren og en med reverse-proxy (Mer om reverse-proxy i kappitel 5.1.1 om HTTPS). For frontend brukte vi et Docker skript til å hente inn frontend prosjektet og installere et React skript for å kunne kjøre frontend prosjektet. For å starte backend og frontend brukte vi *yml* skripet for å starte eller bygge alt med en kommando.

3.5 Programmeringsspråk

3.5.1 Java

Java er et objektorientert programmeringsspråk. Det er et kompilert høynivåspråk som kan bli kjørt på de aller fleste plattformer. Dette er ved hjelp av Java Virtual Machine som gjør at Java kun trenger å bli kompilert en gang og kan deretter bli kjørt på mange plattformer. Java bruker også Just-in-time-compilation som gjør at koden blir kompilert når programmet blir kjørt. Vi valgte å bruke Java til backend slik at vi kunne lage en REST api-server som kan kjøre med Spring Boot rammeverket.

3.5.2 Hypertext Markup Language (HTML)

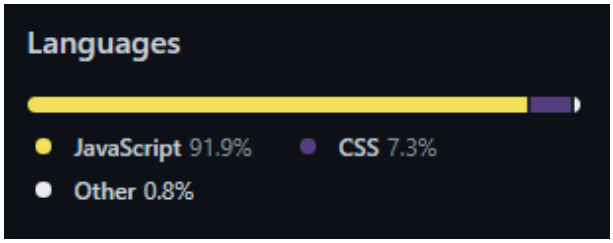
HTML er det standard markeringsspråket som blir brukt når et dokument skal bli vist i en nettleser. Et markeringsspråk er ett system for å annotere et dokument slik at selve annoteringen ikke blir vist til brukeren, men er brukt for å formatere teksten (Nordal 2019). Ett HTML-dokument er bygd opp av *elementer* som definerer hvordan dokumentet er bygd opp. HTML-elementene blir definert gjennom tagger slik som `<h1>` og `</h1>` . Disse taggene kan ha attributter som definerer hvilke funksjonaliteter og utseende elementet vil ha. Ett HTML-dokument blir ofte brukt sammen med CSS og JavaScript for å lage fullstendige nettsider med styling og funksjonalitet. (HTML 2021)

Siden prosjektets frontend kjører på en nettside blir mye HTML brukt. Selv har ikke vi skrevet mye ren HTML, men siden vi bruker biblioteket React 3.6.1, som implementerer JSX 3.6.1.3, så skriver vi for det meste i JSX. JSX elementene blir gjort om til HTML elementer gjennom React og blir vist på nettsiden som ren HTML.

3.5.3 Cascading Style Sheet (CSS)

CSS er et stilspråk, som betyr at det brukes for å beskrive presentasjonen av et dokument som er skrevet i et markeringsspråk (Learn to style HTML using CSS - Learn web development — MDN 2021). CSS brukes som oftest for å gjøre det enkelt å skill mellom presentasjon og innhold. Hvilket innhold som blir vist styres av HTML mens CSS handler bare om å endre på presentasjonen av innhold gjennom endring av oppsettet til dokumentet, fargene som blir brukt og fonter (HTML & CSS - W3C 2021). CSS brukes som oftest sammen med HTML og JavaScript for å lage nettsider.

CSS har blitt brukt veldig mye hos oss. For at en nettside skal se bra ut så er bruk av stilark veldig viktig. I prosjektet så har de fleste komponentene 3.6.1.1 sitt eget stilark som bestemmer hvordan den ene komponenten skal se ut, og hvordan elementene som er i komponenten skal bli plassert og se ut. Ifølge vår git repository så består 7.3% av vårt frontend prosjekt av CSS kode.



Figur 15: Git code percentages

3.5.4 JavaScript

JavaScript er et programmeringsspråk som blir brukt på nettsider for å skape dynamiske nettsider med høy funksjonalitet, fleksibilitet og respons. Javascript jobber best sammen med HTML og CSS. (What is JavaScript? - Learn web development — MDN 2021)

JavaScript er en av de viktigste delene i frontenden vår. Den blir brukt til nesten all funksjonaliteten på nettstedet og er en nøkkelfaktor til at resultatet vårt oppfyller kravene som ble stilt til oss i oppgaven. Uten JavaScript ville det ikke vært mulig å lage en dynamisk nettside som gir en god brukeropplevelse samtidig som den oppfyller alle kravene som ble stilt i oppgaven. Måten vi skriver JavaScript kode på blir også forandret veldig mye av React biblioteket 3.6.1. Vi vurderte å bruke TypeScript istedenfor JavaScript, men valgte å ikke gjøre det siden vi var usikre på tiden og vi hadde erfaring med JavaScript fra før.

3.6 Rammeverk & biblioteker

3.6.1 React

https://reactjs.org/
React er et åpent kilde JavaScript bibliotek som spesialiserer i å skape brukergrensesnitt. Det er hovedsakelig vedlikeholdt av Facebook og et samfunn som bidrar til den åpne kildekoden.

3.6.1.1 Komponenter

I React så er det mange nyttige funksjonaliteter som en utvikler kan ta i bruk. Hva hele React bygger på er komponenter. En komponent kan tenkes lik ett HTML-element med flere funksjonaliteter og muligheter. Det er mulig å sende verdier gjennom propsslik at komponenten kan ta i bruk informasjon som kommer fra forelder komponenten. Denne prosessen er ganske lik hvor en sender en parameter til en funksjon i de fleste programmeringsspråk. En komponent kan bli bygd opp på to forskjellige måter; som en funksjonell komponent eller en klasse-basert komponent. Gjennom prosjektet har vi hovedsakelig brukt funksjonelle komponenter.

3.6.1.2 States

En veldig viktig del av React er dens mulighet til å bruke *states*. En state i en komponent er en form for variabel som holder på verdier som tilhører komponenten. Hver gang en state blir oppdatert så vil komponenten oppdatere seg selv og vise en nyere oppdatert versjon med den nye informasjonen på nettsiden. I en klasse komponent lages en state ved å definere den i konstruktøren, og i en funksjonell komponent så blir en state laget ved å bruke en *hook*. Dette ble brukt i mesteparten av komponentene våre og var en essensiell del til hvordan vi bygde opp våre komponenter.

3.6.1.3 JSX

React tar også i bruk JavaScript XML (JSX). JSX er en syntaks utvidelse for JavaScript og gjør utvikling av brukergrensesnitt enkelt og veldig likt som å skrive vanlig HTML. JSX lager React elementer som gjøres om til HTML elementer på siden. Du kan bruke JavaScript uttrykk i JSX og JSX utrykk i JavaScript med React. JSX syntaksen har blitt brukt i nesten alle komponentene som vi har skrevet, og gjorde koden mye mer forståelig og lettere å skrive. (Introducing JSX – React 2021)

3.6.1.4 Virtuell DOM

En annen funksjonalitet React tilbyr er at den har en virtuell document object model (DOM). DOMen er ansvarlig for å holde på tre strukturen til XML eller HTML elementer. Det er enkelt sagt DOMen som holder på hvordan en nettside er bygd opp i struktur. At React har en virtuell DOM betyr at den klarer å se forskjellen med den forrige versjonen av nettsiden og den nye versjonen etter en oppdatering (Virtual DOM and Internals – React 2021). Dermed så kan React oppdatere nettsiden etter hver endring i koden uten at det er nødvendig å laste inn siden på nytt. Dette ble brukt mye og gjorde utvikling av nettsider en veldig god opplevelse.

3.6.1.5 React Bootstrap

https://react-bootstrap.github.io/
React Bootstrap er et rammeverk bygd i React, og inneholder mange komponenter som er enkel å implementere og enkel å endre stiler på. Noen av de mest brukte komponentene i vårt prosjekt kommer fra React Bootstrap. Eksempel på komponenter vi har brukt er `<Button>`, `<Dropdown>` og `<Form>`.

3.6.2 Spring & Spring Boot

3.6.2.1 Spring

Spring er et rammeverk som tilbyr en omfattende programmerings og konfigurasjonsmodell for Java-baserte applikasjoner. Spring selv sier:

> Spring focuses on the plumbing"of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.
> (Spring Framework 2021)

Altså Spring er en av de mest brukte Java EE (Enterprice Edition) rammeverkene for bygging av applikasjoner. Spring sikter for å forenkle Java EE utvikling og hjelper utviklere å bli mer produktive. De tilbyr en rekke av funksjonaliteter som testing, auto konfigurasjoner, sikkerhet, Bean livssyklus og mer. Altså ulikt andre rammeverk så fokuserer Spring på flere områder i en applikasjon og gir en stor mengde av forskjellige funksjoner.

3.6.2.2 Spring Boot

https://spring.io/
Spring Boot derimot sikter for å forenkle det å lage en web applikasjon med å korte ned på mengde kode som kreves og gir en enkel løsning på å utvikle den. Dette blir gjort ved hjelp av merknad konfigurasjon (@) på toppen av en funksjon eller klasse, eller ved hjelp av standard kode. Ved hjelp av Spring Boot kreves det nesten ingen konfigurering for å få applikasjonen opp og kjørende.

Etter tidligere erfart å lage en web server applikasjon uten noen rammeverk og hørt om hva Spring kan gjøre for oss, bestemte vi oss for å prøve Spring. Dette gjorde utviklingen mye enklere og vi fikk web serveren opp og kjørene etter kort tid uten noen problemer. Spring Framework 2021

3.6.3 Lombok

https://projectlombok.org/
Lombok er et Java-bibliotek som automatisk lager getters, setters, konstruktør uten parameter eller konstruktør som krever alle parameter. Lombok blir brukt ved å sette *@Data*, *@NoArgsConstructor* eller *@AllArgsConstructor* over klassen slik som i figur 16. Der *@Data* lager hentere og settere, *@NoArgsConstructor* lager konstruktør som ikke krever parameter og *@AllArgsConstructor* lager konstruktør som krever alle parameter. Vi valgte å bruke Lombok for å gjøre modell klasser ryddigere slik at vi kan fokusere på koden som betyr noe ved å sleppe å ha mange setters, getters og konstruktører.

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@NamedQuery(name = User.FIND_ALL_USERS, query = "SELECT u FROM users u ORDER BY u.email")
@NamedQuery(name = User.FIND_USER_BY_EMAIL, query = "SELECT u FROM users u WHERE u.email LIKE: email")
@NamedQuery(name = User.FIND_USER_BY_ID, query = "SELECT u FROM users u WHERE u.userId =: userId")
public class User {
```

Figur 16: Hvordan bruke Lombok.

3.6.4 Eclipse Jersey

https://projects.eclipse.org/projects/ee4j.jersey
Jersey er et rammeverk som skal forenkle det å bruke RESTful API i prosjekter ved å ta seg av detaljer som ville gjort kommunikasjon mellom klient og server mer tungvint. Jersey er en implementasjon av JAX-RS (het tidligere Java API for RESTful Web Services) og utvider den også med ekstra funksjonalitet.

Spring-web ble benyttet til mye av denne funksjonaliteten, mens vi tar i bruk noen exceptions fra rammeverket: *BadRequestException* og *ForbiddenException*.

```
/** Removes a tag from a project. ...*/
public Project removeTag(UUID projectId, List<Tag> tagsToBeRemoved, User remover)
        throws ProjectNotFoundException, ForbiddenException, TagNotFoundException {
    Project project = projectDao.getProjectById(projectId);
    if(!isUserPermittedToChangeProject(project, remover)) {
        throw new ForbiddenException("User is forbidden to do changes on this project!");
    }  else {...}
}
```

Figur 17: Eksempel på bruk av ForbiddenException i ProjectService

3.6.5 imgscalr - Java Image-Scaling Library

https://github.com/rkalla/imgscalr
imgscalr er et Java bibliotek som gjør det enkelt å skalere bilder på backend. I standard Java er det mange forskjellige måter å skalere bilder. Derfor er det praktisk å bruke et bibliotek som tar seg av det på en enkel og rask måte så vi slipper å ta stilling til det. (imgscalr – Java Image Scaling Library — The Buzz Media 2021)

imgscalr brukes til å skalere bildene før de hentes for vises i frontend på prosjekt-sidene.

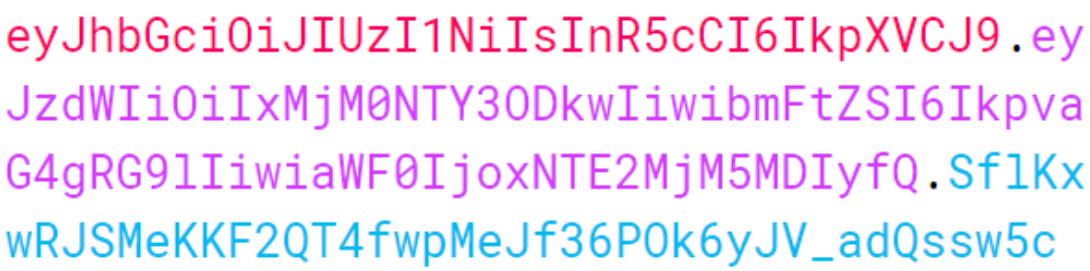
```
/** Scales an image to same as width param. ...*/
public byte[] scaleImage(byte[] imageBytes, String imageFileType, Integer width) throws IOException {
    try {
        ByteArrayOutputStream thumbOutput = new ByteArrayOutputStream();
        BufferedImage thumbImg;
        BufferedImage img = ImageIO.read(new ByteArrayInputStream(imageBytes));
        thumbImg = Scalr.resize(img, Scalr.Method.AUTOMATIC, Scalr.Mode.AUTOMATIC, width, Scalr.OP_ANTIALIAS);
        ImageIO.write(thumbImg, imageFileType, thumbOutput);
        return thumbOutput.toByteArray();
    } catch (ArrayIndexOutOfBoundsException e) {...}
}
```

Figur 18: Hvordan vi bruker *imgscalr* i ImageService

3.6.6 JSON Web Token

https://www.jsonwebtoken.io/
Dette biblioteket bruker vi til å kunne lage *json web token* (jwt). En jwt blir brukt til å huske at en bruker er innlogget ved å sende dem en token. Dette bibliotektet tar tre elementer for å bygge en token. Første elementet er informasjon over hva algoritme som blir brukt, andre er informasjon om brukeren og tredje er en verifisering signatur fra serveren. Se figur 19. Når da innloggingen

er godkjent blir denne sendt inni server responsen som en informasjonskapsel (cookie). Denne informasjonskapselen vil bli lagret i nettleseren til brukeren og vil bli sendt tilbake til serveren ved alle de neste kallene så lenge den er i nettleseren. Slik vet serveren at de er innlogget, fordi de har en gyldig jwt token som blir sendt med hver forespørsel.



Figur 19: Slik ser en jwt ut. Rødt er første, lilla er andre og blå er tredje elementet.

3.6.7 Jasypt Spring Boot

https://github.com/ulisesbocchio/jasypt-spring-boot

Jasypt er er bibliotek som tar en nøkkel verdi og dekrypterer verdien som står inni *ENC()* slik som i figur 20. Vi bruker dette biblioteket for å skjule verdier i filen *application.properties* som holder på verdier som passord, IP-er og andre verdier som vi ikke vil at folk skal kunne se. Om vi skal kryptere en verdi setter vi inn verdien og passordet i en main Java klasse og kjører den som vist i figur 21. Når vi har kjørt den får vi ut den krypterte verdien i konsoll vinduet.

```
file.user=ENC(ZciiKF4jKLN12pIWYKRj+uzFJig+8pUj4ugHsd4aaY02cuSNtZPb00Vru4ts4sRp)
file.pass=ENC(G9bucBSolx/B3JB4cwD/jNIzEOhTw9ueu4P2xxFWg81UP9cS6ImT+NKuPQUtMY2X)
file.domain=ENC(a7X4BFYuJ6z7M80ozjjJjiF5W5LVfOYevEJSZmp4bzlCoEPCBcFWT9Lp4XQfc28O)
```

Figur 20: Krypterte verdier i application.properties

```java
public class JasyptPasswordEncryptor {
    public static void main(String[] args) {

        String textToEncrypt = "appserver";
        String password = "Encryption Password here";

        AES256TextEncryptor encryptor = new AES256TextEncryptor();
        encryptor.setPassword(password);
        String myEncryptedText = encryptor.encrypt(textToEncrypt);
        System.out.println("Encrypted: "+myEncryptedText);

        String plainText = encryptor.decrypt(myEncryptedText);
        System.out.println("Decrypted: "+plainText);
    }
}
```

Figur 21: Kryptering av verdier til application.properties

3.6.8 EclipseLink JPA (Java Persistence API)

https://www.eclipse.org/eclipselink/
JPA kalles for en spesifikasjonen og handler om «vedvarende lagring», det vil si at data blir lagret lengre enn prosessen som opprettet den. Et viktig konsept med JPA er modellen for objekt-relasjons-mapping (ORM). ORM er en oppgave som JPA definerer i det som kalles ORM-laget. Dette laget er ansvarlig for koblingen mellom objekter og radene/kolonnene i relasjonsdatabasen. I Java blir vanligvis klasse-navnet til tabell-navnet og variablene til kolonnene.
(Tyson 2019)

I vårt prosjekt er det EclipseLink som er JPA leverandøren. Det som gjør JPA bra er hvordan det gjør at vi som utviklere kan holde oss til objekt-orientert jobbing og slipper å tenke på detaljene rundt SQL og forholdet mellom modell-klassene og databasen.

```java
@Email
@NotEmpty
private String email;

@NotEmpty
@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
private String password;

@ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
@JoinTable(name = "user_roles",
        joinColumns = @JoinColumn(
                name = "user_id",
                referencedColumnName = "user_id"),
        inverseJoinColumns = @JoinColumn(
                name = "role_name",
                referencedColumnName = "role_name"))
private List<Role> roles = new ArrayList<>();
```

Figur 22: Her bruker vi JPA for å definere kolonner og et mange til mange forhold til en bruker.

3.6.9 Apache Commons IO

https://commons.apache.org/

Apache Commons er et Java bibliotek som inneholder «gjenbrukbare Java komponenter» (Apache Software Foundation 2021). Det vil si at det inneholder mange verktøy som gjør ting enklere siden det er ferdige klasser som vi kan benytte. Det utvikles av personer i Apache-fellesskapet.

En komponent fra dette biblioteket er *Commons IO* som vi bruker for å konvertere fil-strømmer til byte-array.

```java
smbFile = new SmbFile( url: url + "/" + getFileLocation(fileName, project, subFolder) + "/" + fileName, getContextWithCred());
inputStream = new SmbFileInputStream(smbFile);
bytes = IOUtils.toByteArray(inputStream);
```

```java
/** Package all files in param into a zip and return them as a byte array. ...*/
public byte[] getFilesAsZip(List<String> filesToZip, Project project, String subFolder) throws IOException {
    try (FileOutputStream fos = new FileOutputStream( name: "multiCompressed.zip"); ZipOutputStream zipOut = new ZipOutputStream(fos)) {
        for (String srcFile : filesToZip) {...}
    }
    return FileUtils.readFileToByteArray(new File( pathname: "multiCompressed.zip"));
}
```

Figur 23: IOUtils og FileUtils er to klasser fra *Commons IO*. Fra FileStorageService

3.6.10 jcifs-ng

https://github.com/AgNO3/jcifs-ng/

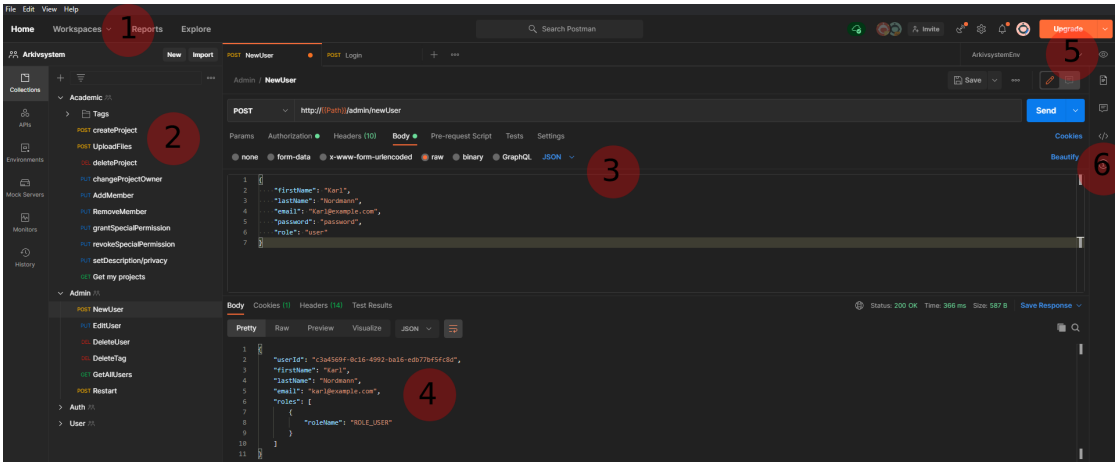
jcifs-ng er en åpenkilde bibliotek som vi bruker til å koble oss til en SMB-filserver for å hente filer, laste opp filer og lage nye mapper. Den fungerer med at vi lager en *SmbFile* som tar inn stien til der filen ligger eller skal ligge med filnavnet og legitimasjon som inneholder passord, bruker navn og domene til fil serveren. Deretter lager vi en inngang strøm eller utgang strøm til å laste ned eller opp informasjonen til filen. Eksempel på å laste ned en fil på figur 23

3.7 Testing

3.7.1 Postman

For testing av REST serveren har vi brukt Postman. Denne applikasjonen lar deg bygge opp en forespørsel og sende den til en server. Med Postman kan du velge mellom å jobbe aleine eller i en gruppe. Som en gruppe kunne vi dele på alle forespørslene vi lagte som gjør at alle på gruppen kunne teste gamle og nye forespørsler. Med Postman kunne vi enkelt bytte mellom alle forespørselen vi har laget fra før for å teste ny kode. Vi brukte også Postman til å se eksempler på hvordan en forespørsel kunne bli laget i forskjellige språk.

1. Med workspace kan du bytte mellom grupper eller prosjekt du vil jobbe med.
2. Her er alle forespørslene som er lagret.
3. Her kan du bygge opp en forespørsel der du kan velge type (POST, GET, PUT..), lenke og innhold som skal bli med i forespørselen.
4. Her kommer tilbakemeldingen fra serveren. Du kan se litt forskjellig informasjon som HTTP status, tid, innhold or mer.
5. På øyet kan du lage forskjellige arbeidsmiljø og ved siden av kan du bytte mellom dem. I et arbeidsmiljø kan vi lage globale variabler som vist på 24 med ”Path”.
6. På </>kan vi sjå eksempler på hvordan denne forespørselen kunne blitt sent med forskjellige språk.



Figur 24: Postman

3.7.2 SonarLint

SonarLint er en utvidelse til IntelliJ for Java som går igjennom koden vi skriver for å sikre kvalitet og kommer på alternativer til å rette problemer. Vi valgte derfor å bruke SonarLint for å sikre god kode stil igjennom prosjektet.

3.7.3 Brukertesting

Brukertesting er når man finner en person som helst er relevant i forhold til hvem som skal bruke applikasjonen, og gi denne personen enkle oppgaver som skal bli utført. Denne personen bør helst ikke ha erfaring med applikasjonen fra før. Med å ta tiden på oppgaven og få tilbakemelding på hvor lang tid oppgaven tok kan vi finne ut om noe trengs å bli fikset eller endret. Det er viktig at personen som tester ikke får noe mer informasjon enn noen andre som tester. Et eksempel på en bruker test oppgave vi hadde er: *Arne Trulsen synes ditt prosjekt virker interessant, men siden ditt prosjekt er privat kan han ikke se filene. Gi han tillatelse til å se og/eller laste ned filer fra ditt prosjekt. Email-en hans er: fridatrulsen@teleworm.us.* Testeren får ikke noe mer informasjon enn dette og informasjonen fra tidligere oppgaver. Brukertesting har hjulpet oss med å finne feil og se at designet vårt har funket slik som det skulle. Tiden testerne brukte på de forskjellige oppgavene har lav som vil si at testerne fort fant fram knappene og funksjonene de trengte for å utføre oppgaven.

3.7.4 Integrasjonstester

En integrasjonstest er definert som en test der programmet er integrert logisk og testet som en gruppe. Altså et program kan bestå av flere klasser eller metoder som er laget av en eller flere personer, så en injeksjonstest vil avsløre feiler mellom klassene eller/og metodene. Vi forsøkte å legge til integrasjonstester for å teste REST forespørslene våre, men hadde problemer med at når vi prøvde å kjøre testene ville ikke applikasjonen starte.

3.8 Kommunikasjon

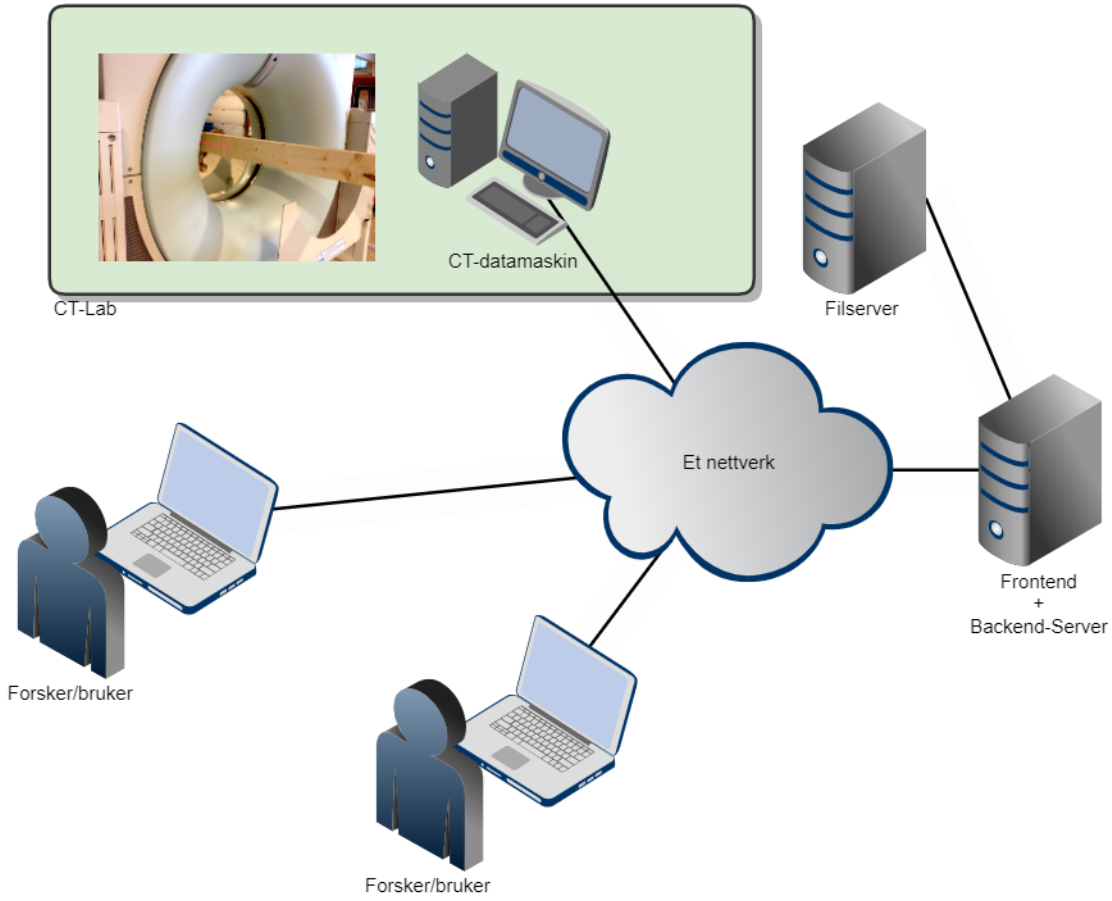
3.8.1 HTTP-statuskoder

HTTP statuskoder er en måte for backend og frontend til å kommunisere. For eksempel om en REST forespørsel har gått godt eller dårlig. Det er mange satte status koder som har forskjellig betydning der det er 5 forskjellige kategorier som blir skilt ut med første tallet i koden. *1xx* er for informasjons koder. *2xx* er for forespørsler som er suksess fra serveren. *2xx* er mye brukt der *200-Ok*, *201-Created* og *204-No Content* er de mest vanlige. *200-Ok* blir ofte brukt til å sende ekstra informasjon til brukeren som for eksempel om en bruker skal hente noen objekter fra en database vil disse objektene bli sent som en body inni *200-Ok*. *3xx* er for omdirigering. Denne statusen blir brukt for å omdirigere en bruker til en annen url. *4xx* er brukt til å si ifra at brukeren har gjort noe galt eller ulovlig. Mye brukte koder er: *400-Bad Request*, *401-Unauthorized*, *403-Forbidden*, *404-Not Found* og *409-Forbidden*. En av de mest vanlige her er *403-Forbidden* som blir brukt om brukeren for eksempel sender noe som er ulovlig i en parameter eller mangler en parameter i en forespørsel. Til slutt *5xx* er for å si ifra om server feil. De blir brukt for å si ifra om noe gikk galt på serveren som ikke er brukeren sin feil. *5xx* kommer for det meste fra programmerings feil der det blir kasten en excepion som ikke blir håndtert riktig. Den mest vanlige her er *500-Internal Server Error*. (Mer om alle statuskodene her eller vår bruk av statuskodene i kapitel 5.1.5)

4 Resultater

4.1 Konsept

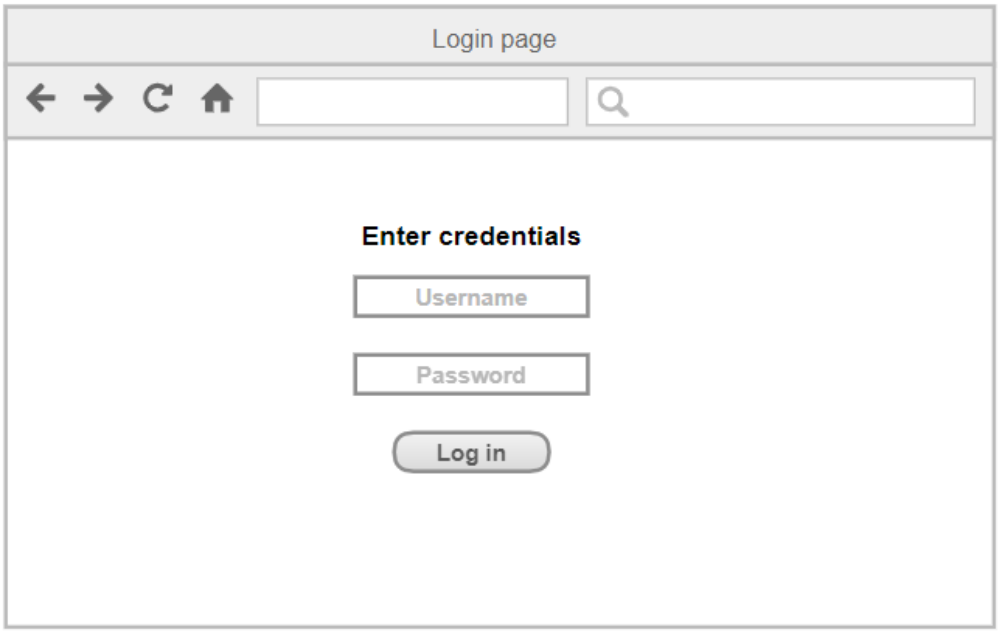
Konseptet går ut på at en bruker skal kunne logge seg inn på nettsiden hvor som helst der han kan lage nye prosjekter, laste opp filer til et prosjekt eller se detaljer, filer og bilder fra eksisterende prosjekter. Alt brukerne henter eller laster opp vil gå til en filserver som skal holde på alle filene. Se figur 25.



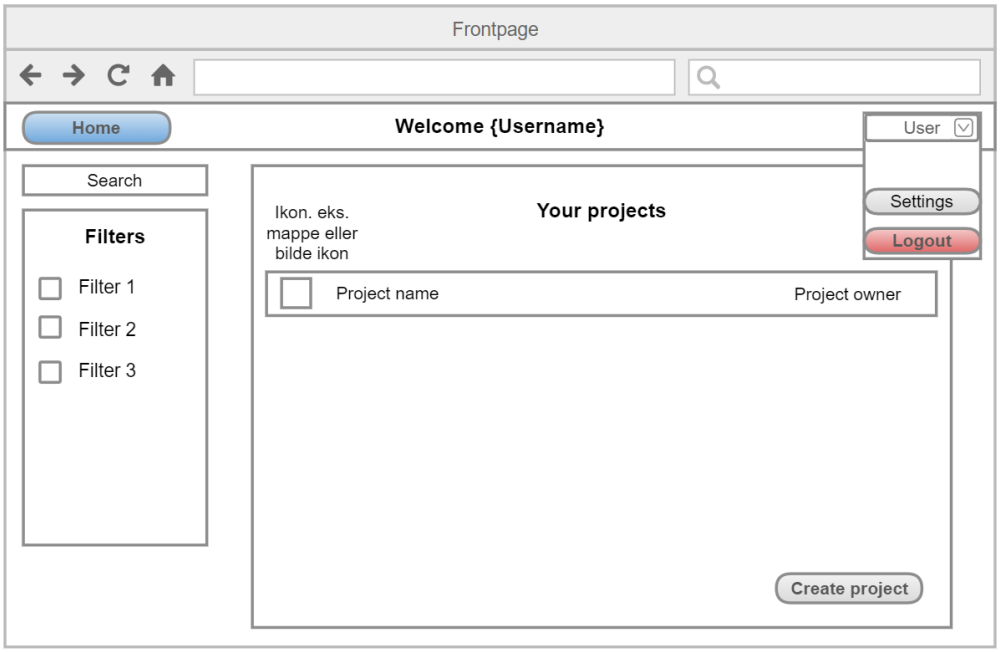
Figur 25: Hvordan vårt resultat fungerer (Dagens løsning kan bli sett på figur 2)

4.2 GUI - Design

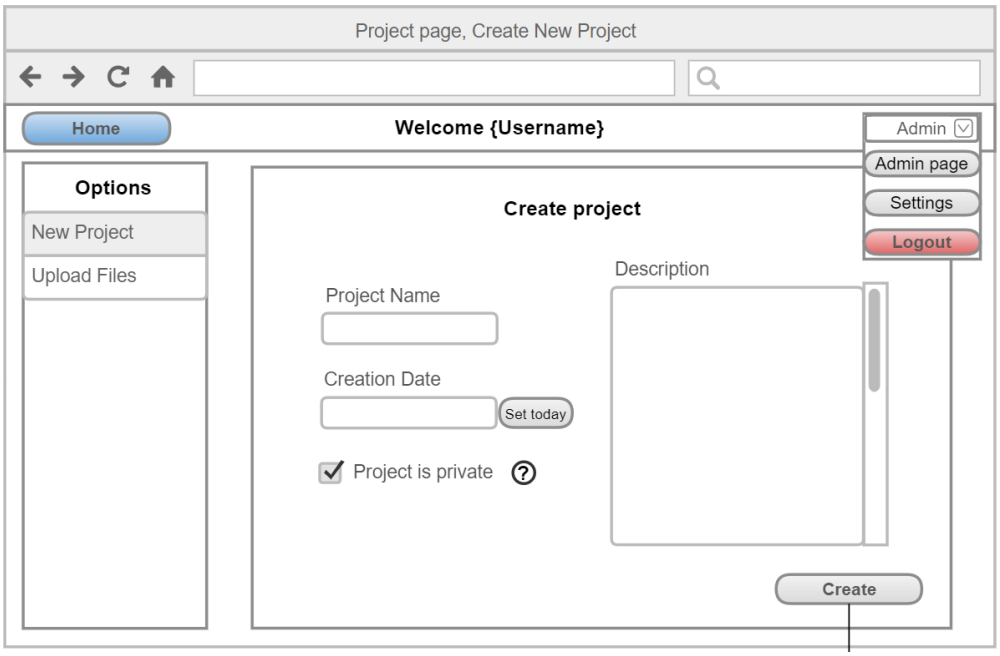
Når vi lagde designet gikk vi for et enkelt design med så få knapper som mulig. Vi ville at nettsiden skulle være så enkel som mulig å bruke. Nesten alle designene ble brukt til resultatet med unntak av endring av prosjekttagger og prosjektbilder. Vi fant ut at ingen av dem trengte en egen side. Prosjektbilder fikk vi også tilbakemelding om at den burde ligge inne i prosjektfiler for å gjøre løsningen mer konsistent. Utenom det var det bare gjort små endringer der vi har endret plassen på noen elementer eller endret et rutenett fra 3 til 1 i bredden.



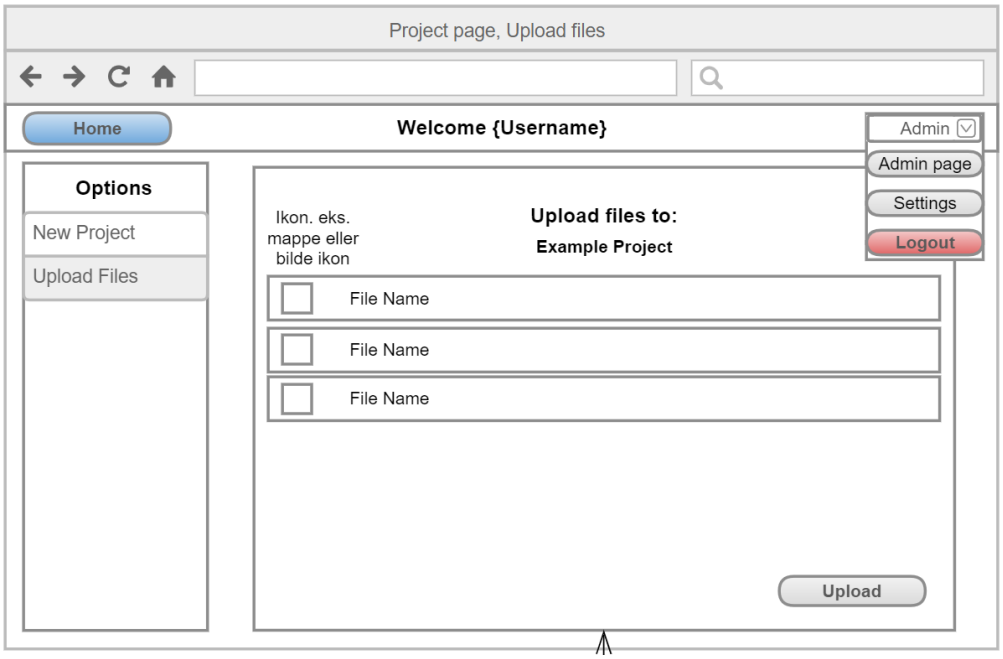
Figur 26: Log inn side konsept. Se figur 45 for resultatet.



Figur 27: Framside side konsept. Se figur 47 for resultatet.

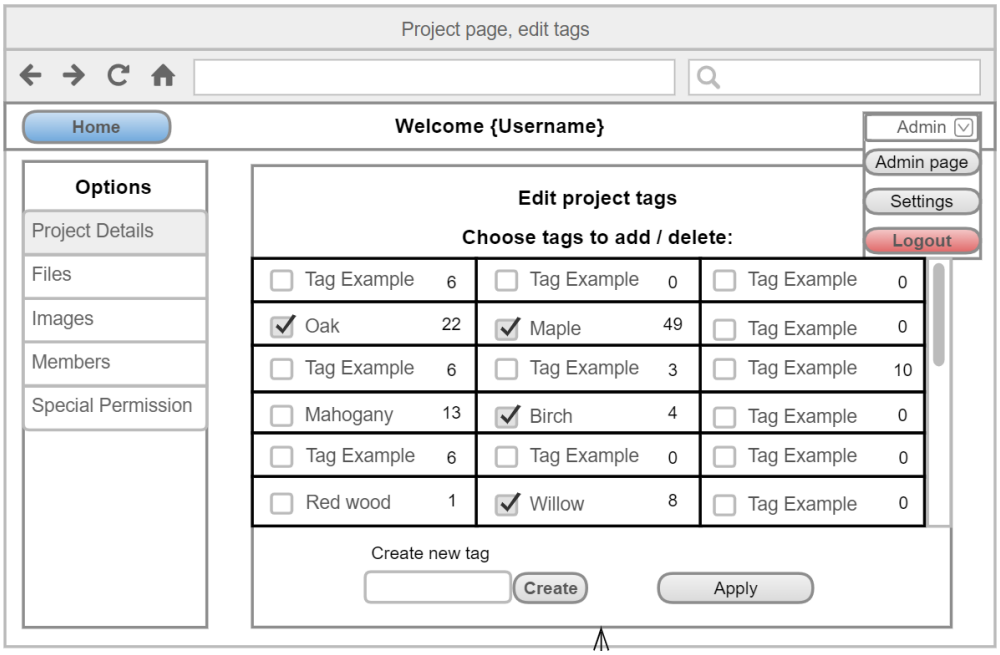


Figur 28: Lag prosjekt side konsept. Se figur 48 for resultatet.

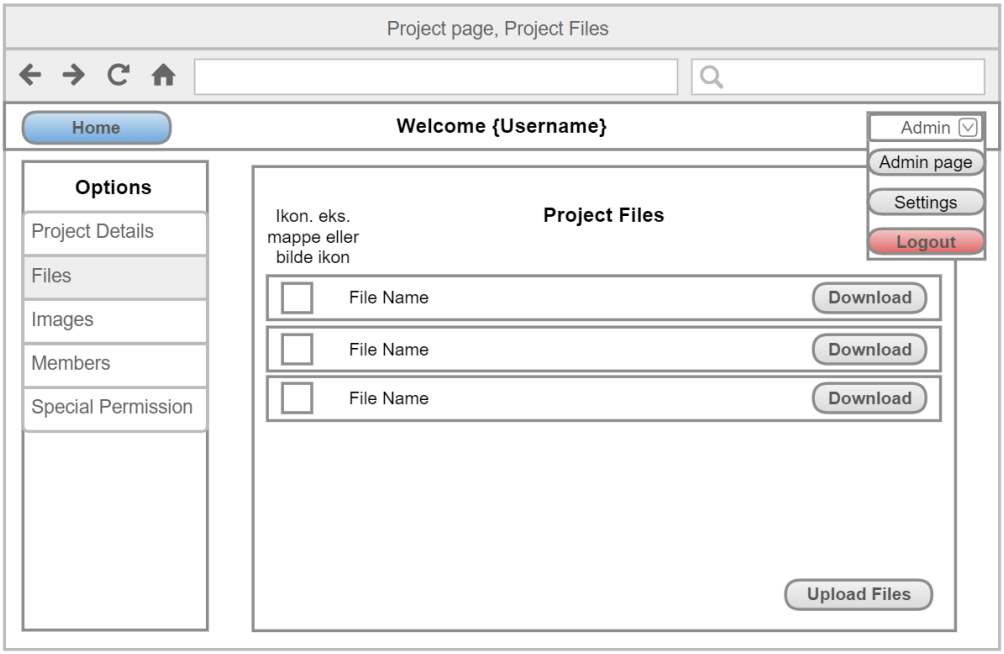


Figur 29: Fil opplasting side konsept. Se figur 53 for resultatet.

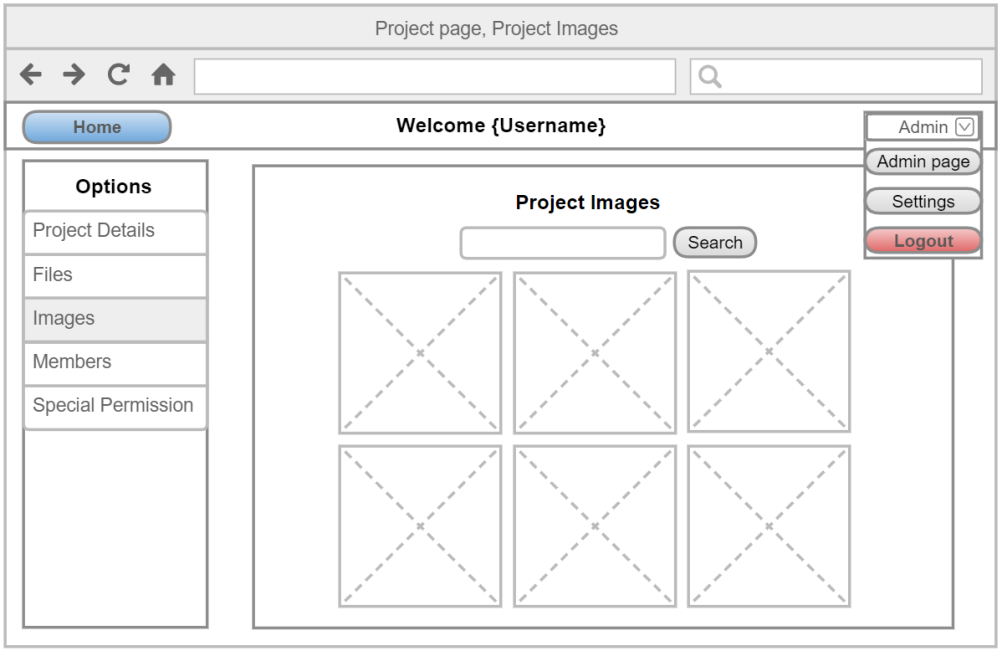
Figur 30: Prosjekt detaljer side konsept. Se figur 50 for resultatet.



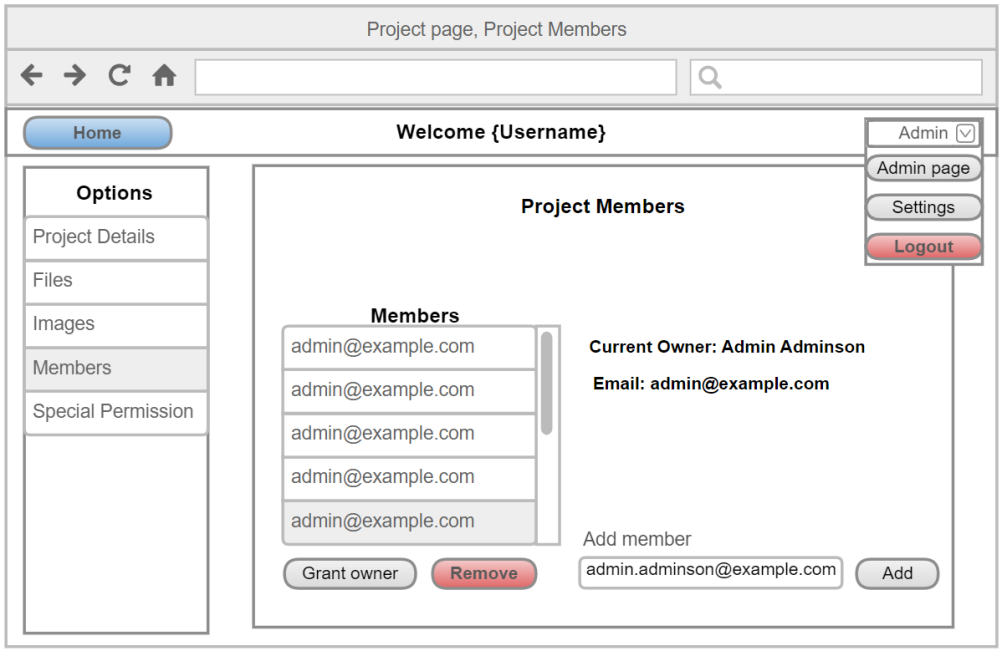
Figur 31: Endre på prosjekt tag side konsept. Se figur 52 for resultatet.



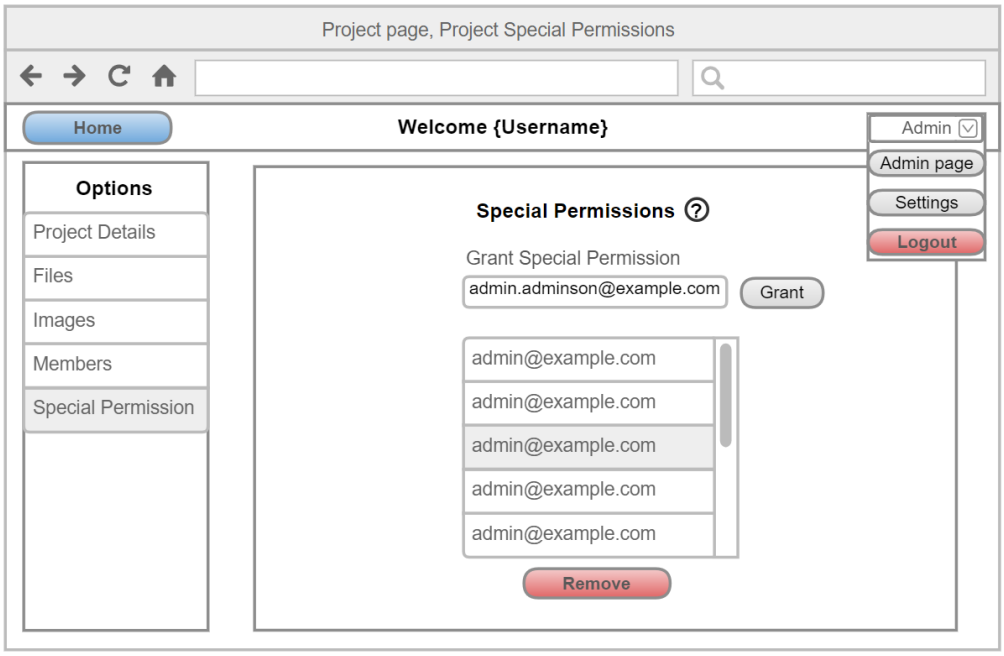
Figur 32: Prosjekt filer side konsept. Se figur 54 for resultatet.



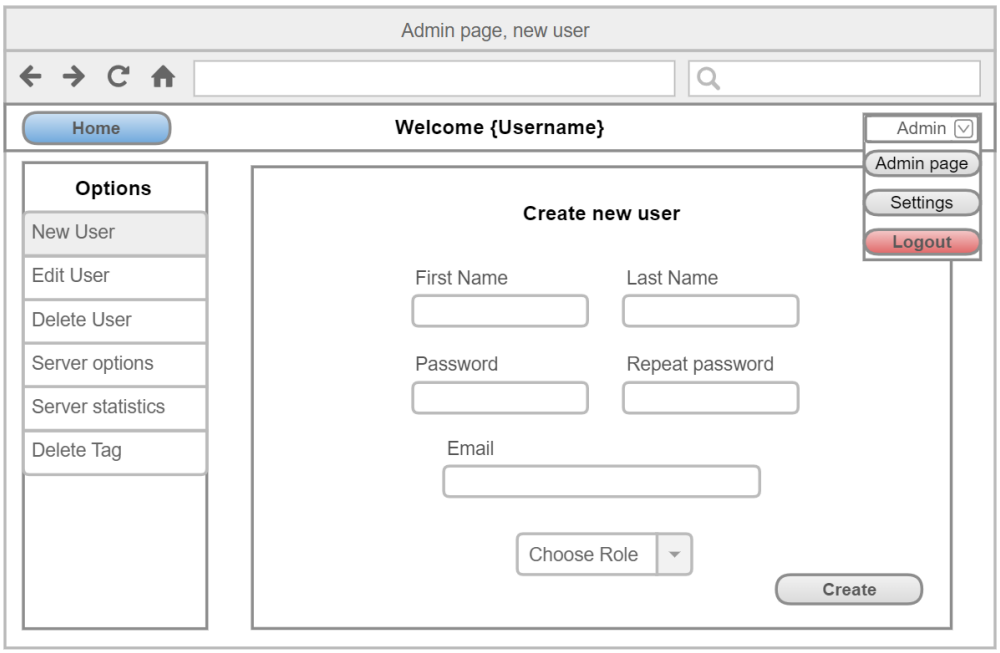
Figur 33: Prosjekt bilder side konsept. Se figur 55 for resultatet.



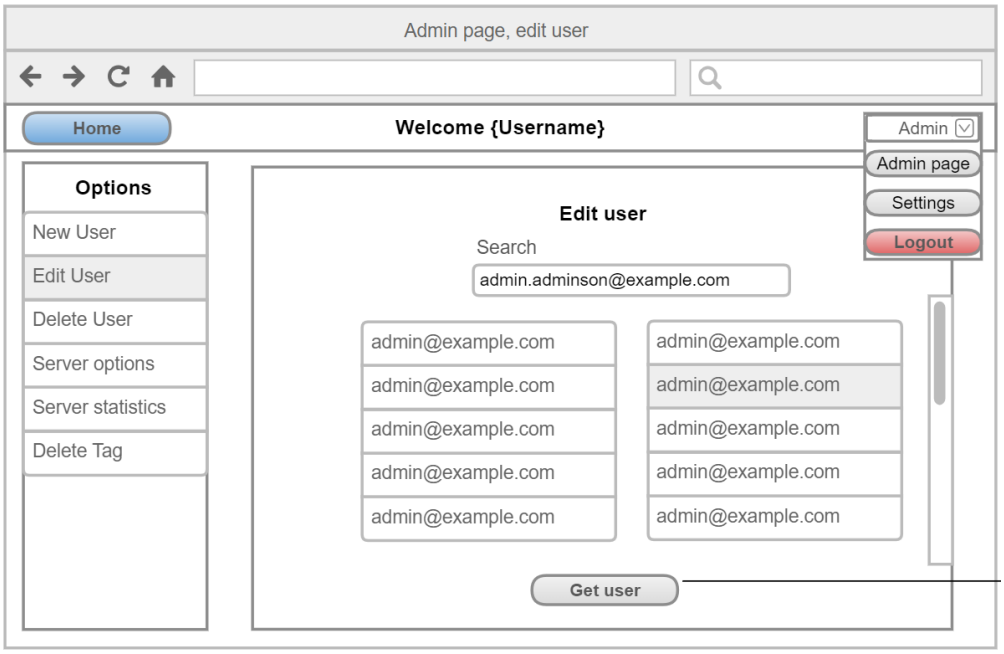
Figur 34: Prosjekt medlemmer side konsept. Se figur 57 for resultatet.



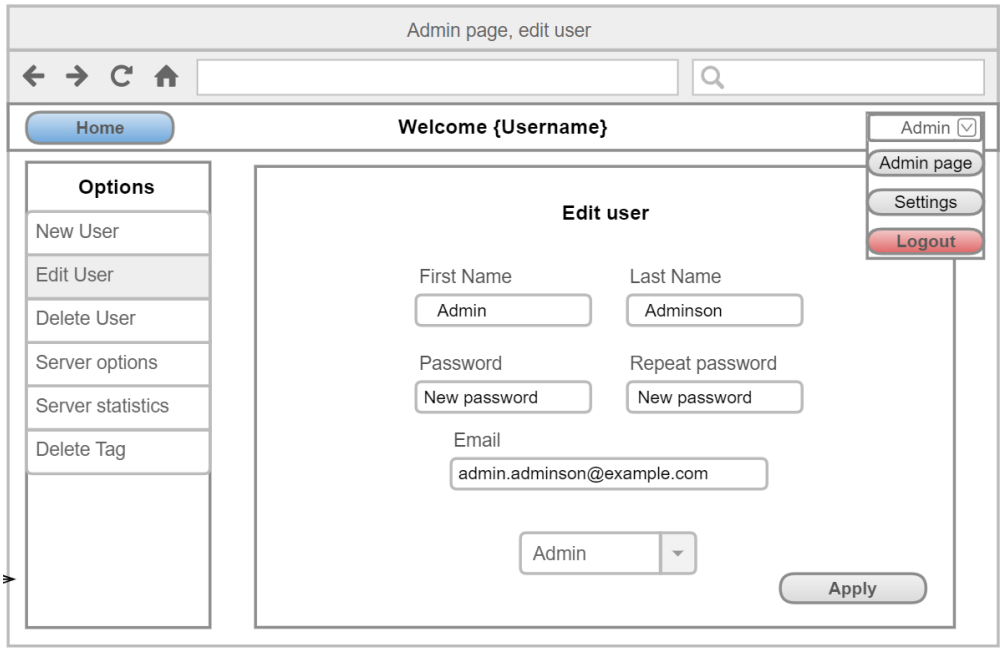
Figur 35: Prosjekt spesial tillatelse side konsept. Se figur 60 for resultatet.



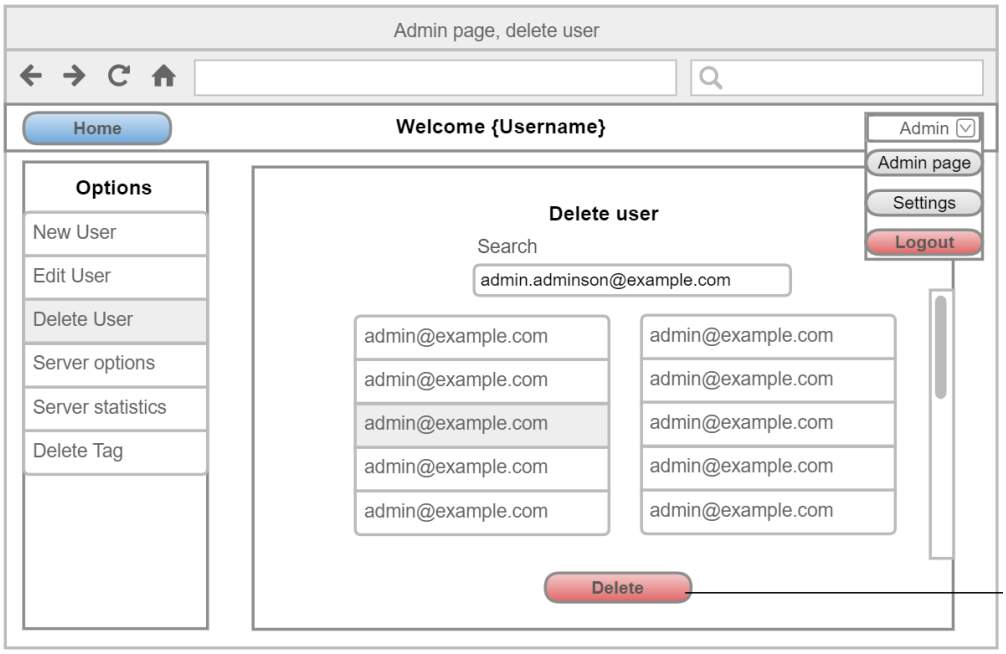
Figur 36: Ny bruker side konsept. Se figur 63 for resultatet.



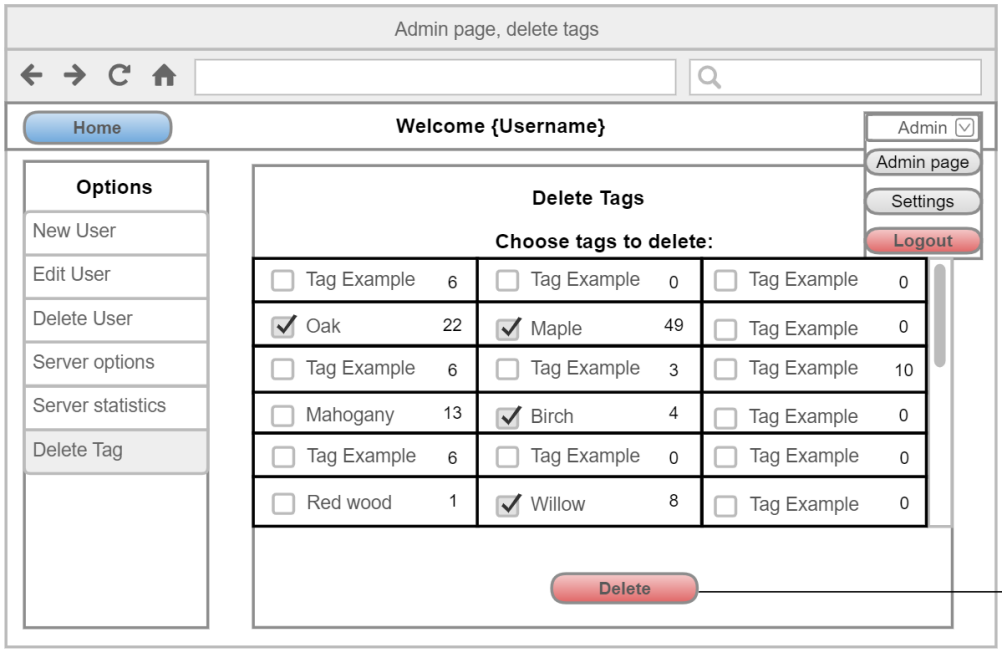
Figur 37: Finn en bruker til å endre side konsept. Se figur 64 for resultatet.



Figur 38: Endre på en bruker side konsept. Se figur 65 for resultatet.



Figur 39: Slett en bruker side konsept. Se figur 67 for resultatet.



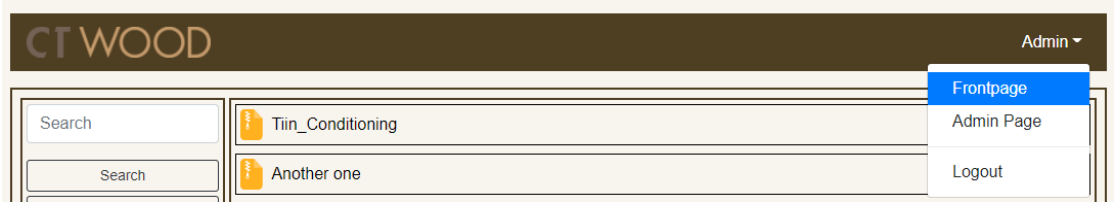
Figur 40: Slett en tag side konsept. Se figur 69 for resultatet.

Figur 41: Eksempel på dialog konsept. Se figur 70 for resultatet.

4.3 GUI-Brukergrensesnitt

På nettstedet så er det en navigasjons bar som er synlig på alle sider innenfor nettstedet. Denne navigasjonsbaren ble satt i toppen av nettstedet og inneholder logoen til nettstedet og en knapp. Logoen fungerer som en link og vil lede brukeren til hjemmesiden hvis de ikke er logget inn. Hvis de er logget inn vil den lede dem til framsiden til brukeren. Knappen vil lede brukeren til login siden om de ikke er logget inn. Hvis de er logget inn så vil knappen bli til en nedtrekksmeny som vil vise en knapp for å dra til framsiden, og en knapp for å logge ut. En bruker med rollen admin vil også se en knapp som leder til administrator siden i nedtrekksmenyen. Se figur 42 for når brukeren ikke er logget inn, og figur 43 for når brukeren er logget inn som en bruker med rollen admin.

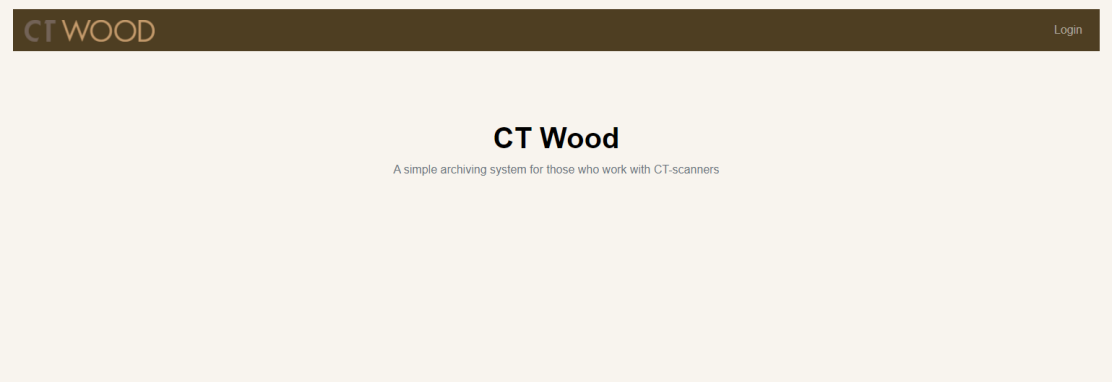
Figur 42: Navigasjonsbar når brukeren ikke er logget inn



Figur 43: Navigasjonsbar som en admin bruker

4.3.1 Hjemmesiden

Hjemmesiden til nettstedet har veldig lite innhold, fordi en bruker skal ikke kunne gjøre noe når de ikke er logget inn. Siden vises bare når en bruker ikke er logget inn. Se figur 44 for hvordan hjemmesiden ser ut.

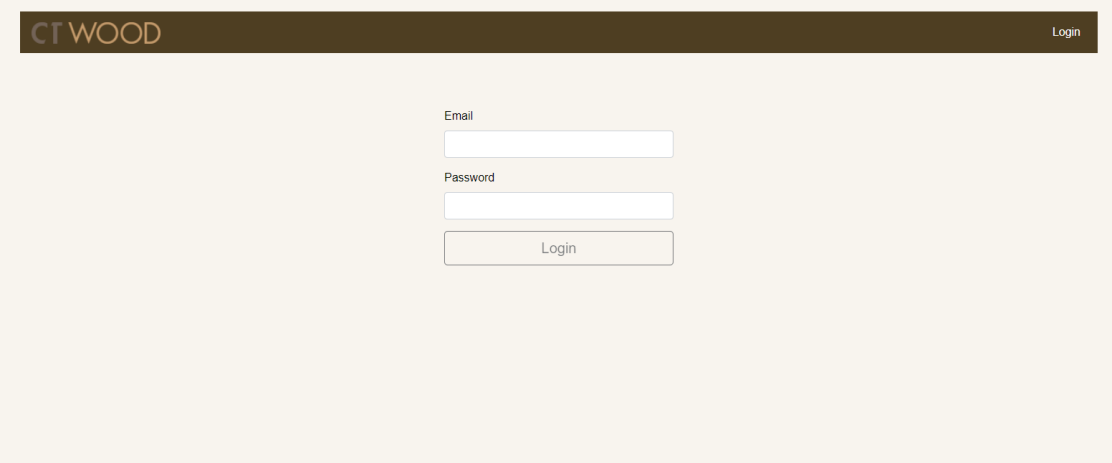


Figur 44: Hjemmeside

4.3.2 Login siden

På login-siden vil brukeren ha mulighet til å fylle inn to felt. I det øverste feltet så fyller brukeren inn deres epost adresse og i det nederste så fyller de inn med passordet sitt, se figur 45 for hvordan siden ser ut. Når begge feltene er fylt inn med gyldig informasjon så får brukeren mulighet til å trykke på Loginknappen som vil sende en API-forespørsel til backenden om brukeren sin informasjon stemmer med en bruker.

Forspørselen som ble sendt blir behandlet i backend og sender tilbake sitt svar. Hvis brukeren sendte feil epost eller passord vil brukeren få en tilbakemelding under login knappen hvor det står Email and/or password was incorrect.". Hvis brukeren ble logget inn vil en informajonskapsel bli lagret i nettleseren til brukeren, og brukeren vil bli sendt til framsiden deres.

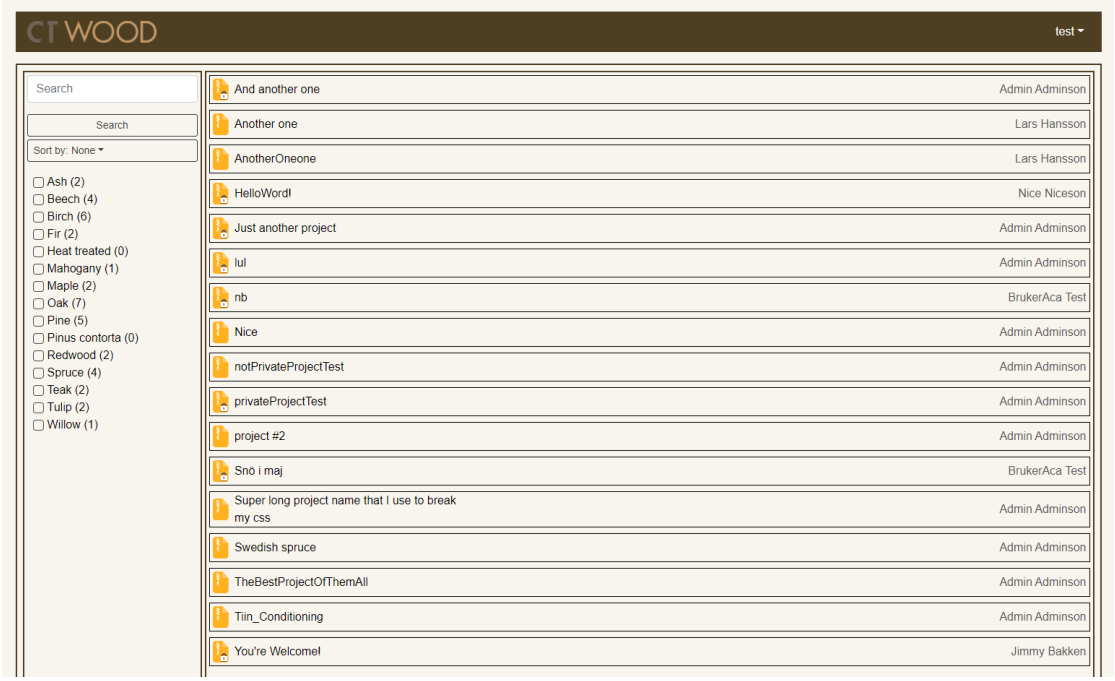


Figur 45: Login side resultat

4.3.3 Framsiden til en bruker

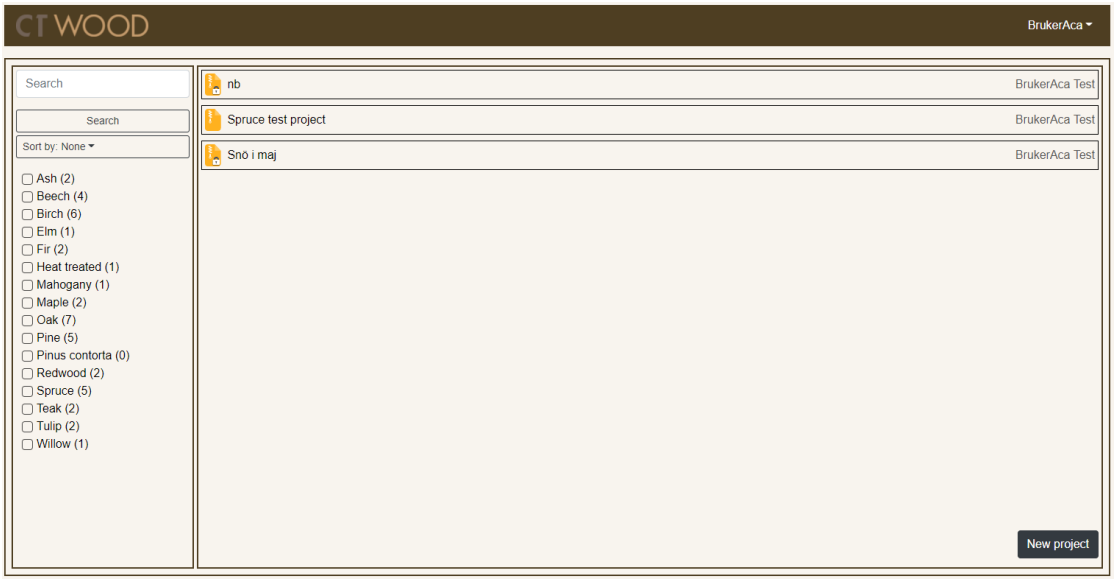
På framsiden vil brukere kunne se hvilke prosjekter som har blitt lagt til i systemet. Nettsiden vil være delt inn i to deler. Delen til venstre kan brukeren søke og filtrere søkeresultater for å finne ett spesifikt prosjekt eller se hvilke prosjekt som eksisterer. Delen til høyre inneholder alle prosjekter som skal bli vist. Om en bruker har rollen *academic* eller *admin* så vil prosjekt de eier eller er medlem av bli vist. En bruker med rollen *user* vil se alle prosjektene som eksisterer. Søkeresultater vil også bli vist hær. Et prosjekt som blir vist fram kan alle brukere trykke på og bli sendt til prosjektsiden til det prosjektet. En bruker med *academic* eller *admin* rollen kan også gå til siden hvor de kan opprette ett nytt prosjekt. Se figur 46 og 47 for resultat.

En bruker med rollen user ser alle prosjekter uten å måtte søke, og det eneste de kan gjøre er å gå inn på ett prosjekt.



Figur 46: Framsiden som en user bruker

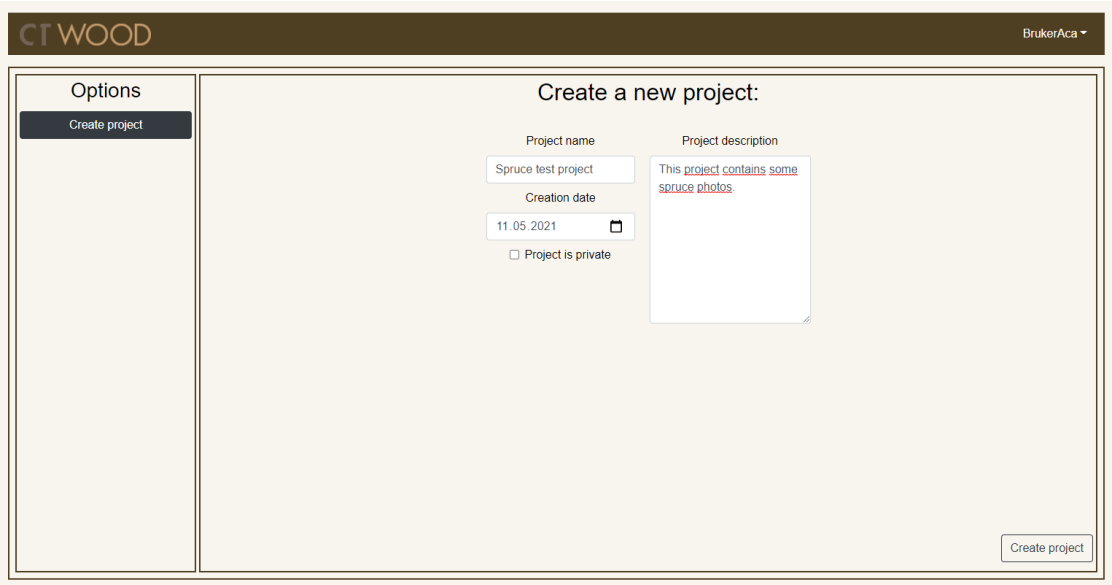
En bruker med rollen academic eller admin vil se sine egne prosjekter før de søker, og har muligheten til å gå inn på opprett nytt prosjekt siden.



Figur 47: Framsiden som en academic bruker

4.3.4 Opprett nytt prosjekt

En bruker med rollen academic eller admin har mulighet til å opprette nye prosjekt på denne siden. Det er tre felt som brukeren må fylle inn for å få opprettet ett prosjekt. De må fylle inn navnet til prosjektet, beskrivelsen av prosjektet og hvilken dato prosjektet ble startet. De har også muligheten til å sette prosjektet som privat eller ikke. Når feltene er fylt inn, vil brukeren kunne opprette prosjektet. Brukeren blir da sendt til prosjektets side. Se figur 48 for resultat.

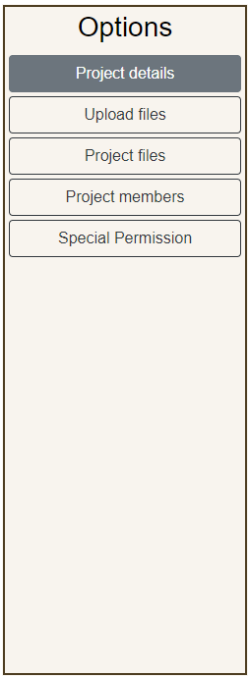


Figur 48: Opprett nytt prosjekt siden

4.3.5 Prosjekt siden

På prosjekt siden kan brukere se og endre på informasjon om prosjektet, se og laste opp filer til prosjektet, se og endre på medlemmer i prosjektet og se og endre på brukere med spesial tillatelse til prosjektet. Hva en bruker har lov til å gjøre i ett prosjekt kommer an på deres rolle og hvilke rettigheter de har i prosjektet. For å se hva en bruker kan gjøre i ett prosjekt se kapittel 1.5.1.

På prosjekt siden er det en ekstra navigasjons bar som står til venstre på siden. Hær vil en bruker kunne navigere seg mellom forskjellige *tabber* med innhold. Se figur 49 for resultat.



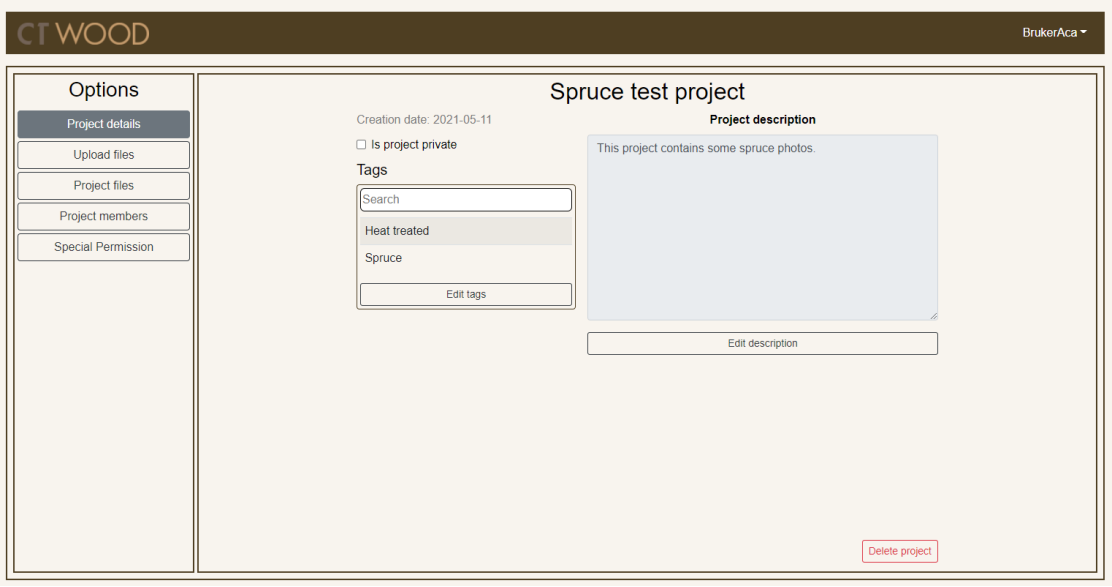
Figur 49: Navigasjonsbaren i et prosjekt

4.3.5.1 Prosjekt detaljer

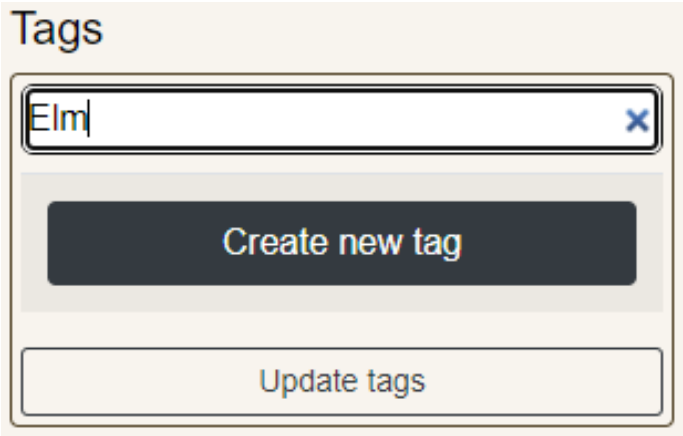
I denne delen av prosjekt siden vil en bruker kunne se og eventuelt endre på grunnleggende informasjon om prosjektet. En bruker kan se og endre på prosjektets private status, prosjektets beskrivelse og hvilke *tagger* som er satt på prosjektet. Se figur 50.

Når en bruker trykker på `Edit tags` knappen vil siden liste opp alle de eksisterende taggene med avhukingsbokser på seg. Brukeren kan da velge hvilke tagger de vil ha på prosjektet, se figur 52. Når brukeren har valgt hvilke tagger som skal være på prosjektet så kan de trykke på `Update tags` knappen og prosjektet vil oppdatere seg. De nye taggene vil nå bli vist i Tagslisten. Brukeren kan også skrive inn en tag som ikke eksisterer og de vil da få muligheten til å opprette den. Se figur 51.

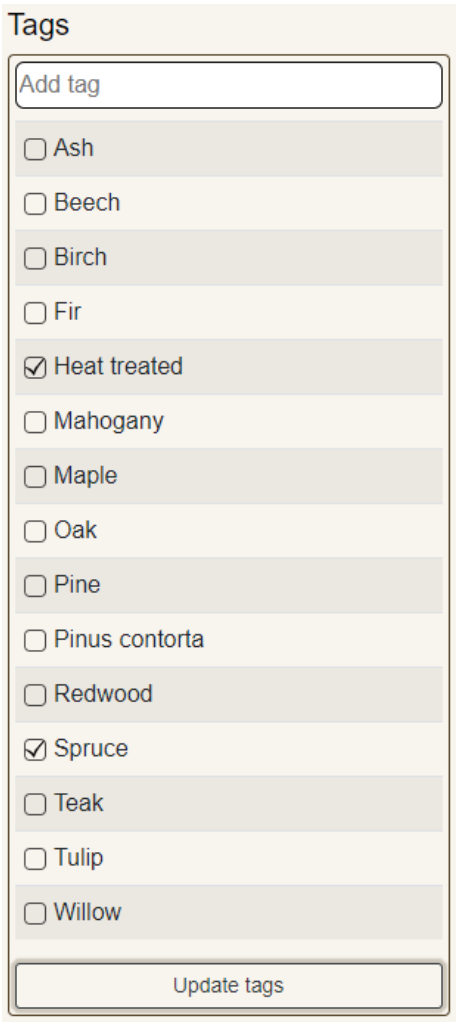
Dette er en av de to tingene som ikke ble som planlagt i design fasen vår.
I designet vårt ble det tenkt at når en bruker skulle legge til tags til et prosjekt så ville de først bli sendt til en ny side. På denne siden skulle de få en oversikt over alle tags og kunne huke av de taggene de villa ha på prosjektet, se figur 31 for original designet til tagging av et prosjekt. Når det kom til implementering av dette virket det ikke særlig brukervennlig å flytte brukeren fra hele siden for å utføre denne funksjonen, og det virket som det ville bli mye mer arbeid å utvikle enn det som var nødvendig.



Figur 50: Prosjekt detaljer siden



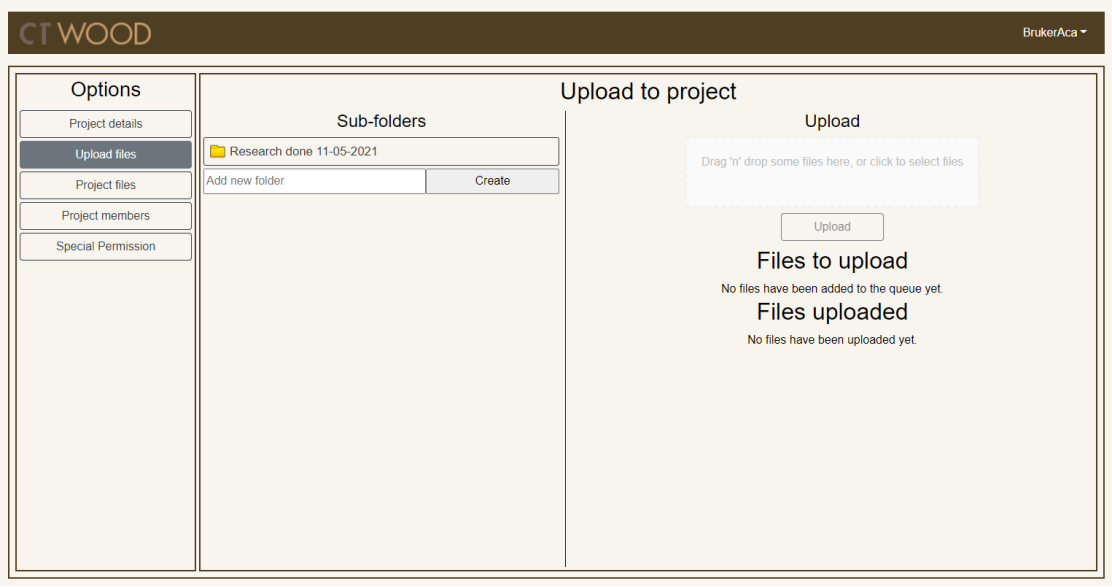
Figur 51: Hvordan en bruker kan opprette en ny tag



Figur 52: Hvordan en bruker kan endre på taggene i ett prosjekt

4.3.5.2 Opplasting av filer

I prosjektet har brukeren mulighet til å laste opp filer. For at brukeren skal kunne laste opp må de først lage en *undermappe* i prosjektet. Deretter kan de velge en undermappe, og laste opp filer ved å enten dra og slippe filer i boksen på nettsiden eller ved å trykke på boksen og velge filer gjennom et pop-opp vindu i utforskeren deres. Deretter blir filene vist på nettsiden og brukeren kan trykke på `Upload` knappen for å laste de opp til den valgte undermappen i prosjektet. Se figur 53.



Figur 53: Siden hvor brukeren laster opp filer

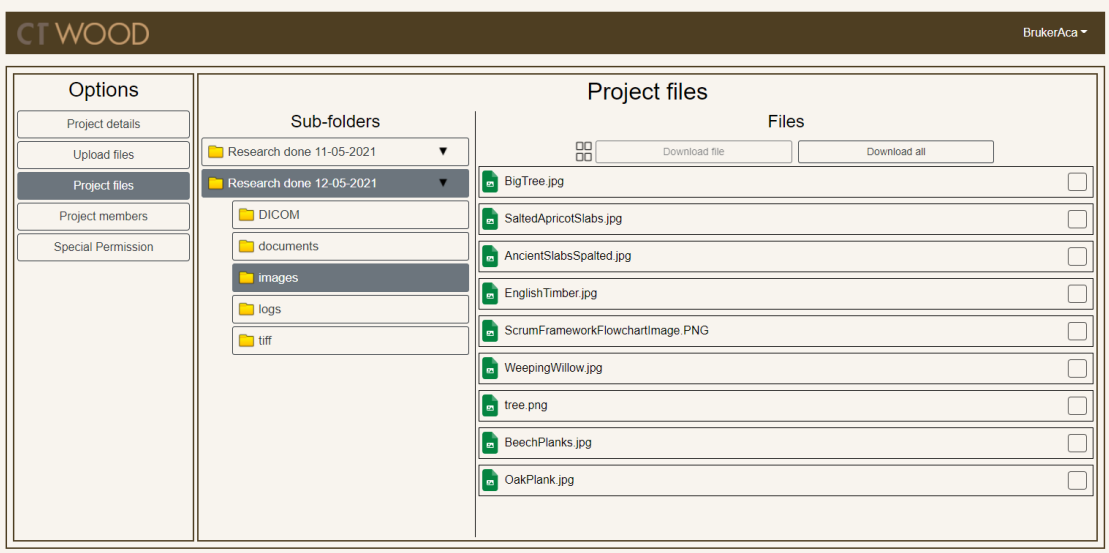
4.3.5.3 Visning av filer

Hvis et prosjekt har undermapper med filer i seg så kan en bruker se filene fra denne siden. Siden er delt inn i to deler, mappestrukturen og filviseren. I mappestrukturen kan en bruker velge en undermappe og deretter hvilken mappe de vil hente filer fra. Filene er sortert ut i fra deres filtype slik at brukeren alltid vet hvilken mappe de ligger i så lenge de vet hvilken type fil de leter etter.

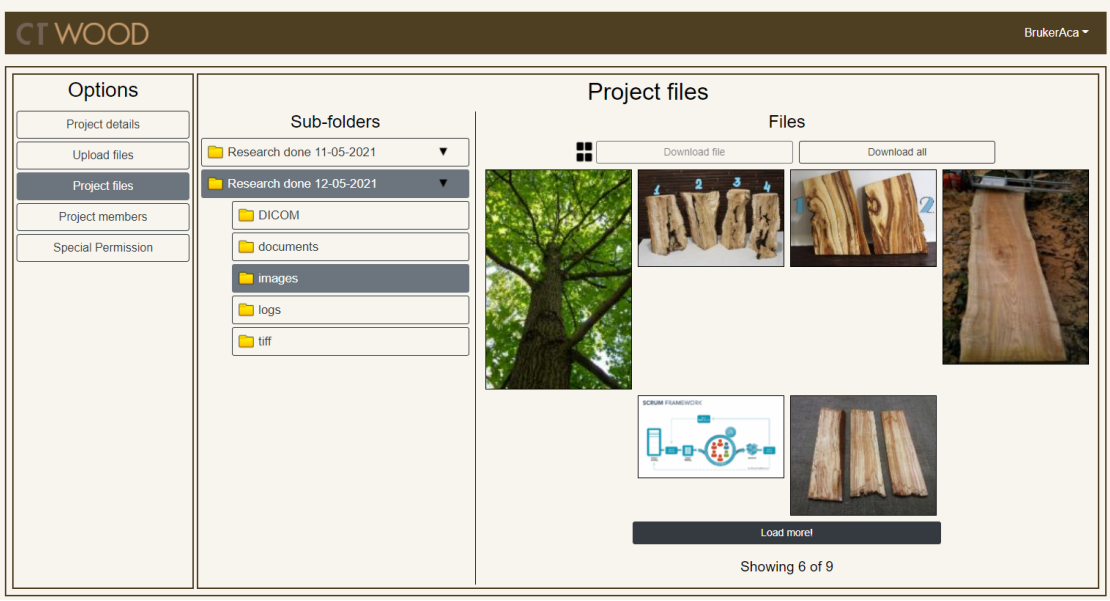
Brukeren kan laste ned filene på nettsiden. Brukeren kan enten huke av enkelte filer og så laste dem ned ved hjelp av `Download file(s)` knappen, eller laste ned alle som blir vist fram ved hjelp av `Download all` knappen. Se figur 54

Hvis brukeren er i `images` mappen i noen undermappe så vil en ekstra funksjon bli tilgjengelig. Et lite ikon vises til venstre for nedlastningsknappene og hvis den er trykket på så kan brukeren se bildene på nettsiden. Brukeren kan også trykke på et av bildene og laste det ned, eller laste ned alle. Se figur 55 for å se denne funksjonen i bruk.

Denne siden inneholder også avvik fra det som ble originalt designet. Funksjonen av å kunne se bildene i ett prosjekt skulle egentlig være på en egen side. Dette vises på figur 33. Vi fikk tilbakemelding om at denne funksjonaliteten burde heller bli implementert i prosjekt filer siden, og etter en god del refaktorering så ble funksjonen flyttet dit og prosjekt bilde siden ble slettet.



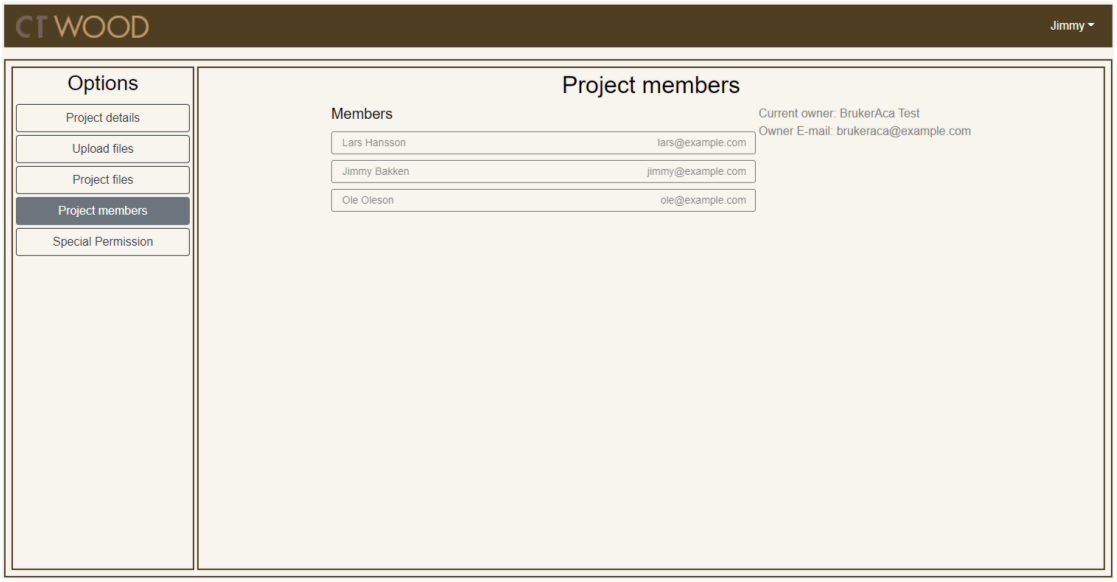
Figur 54: Side hvor brukeren kan se filene i prosjektet



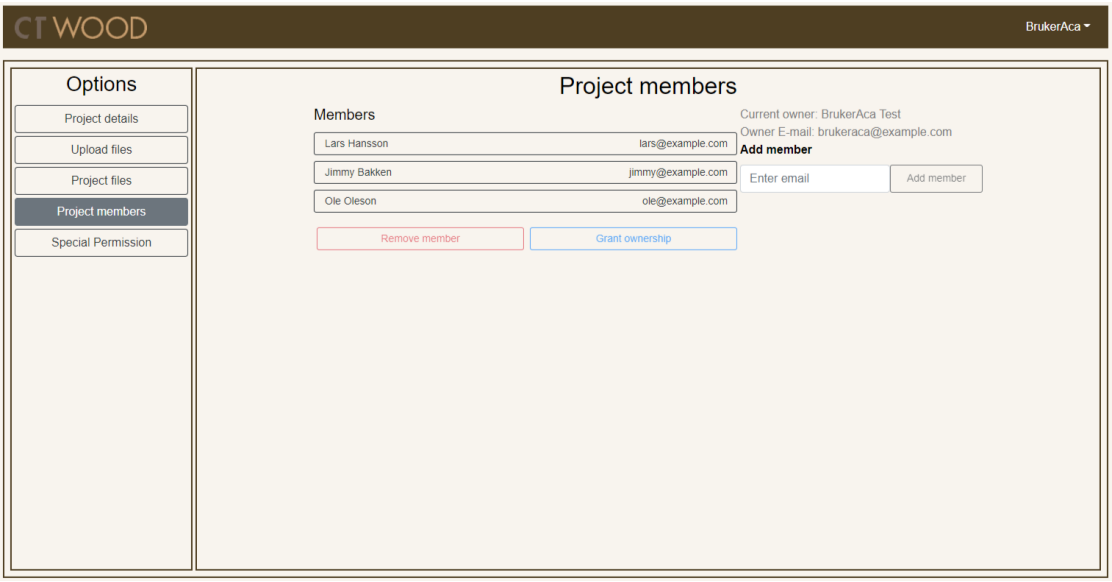
Figur 55: Bilder i en undermappe som blir vist

4.3.5.4 Medlem oversikt

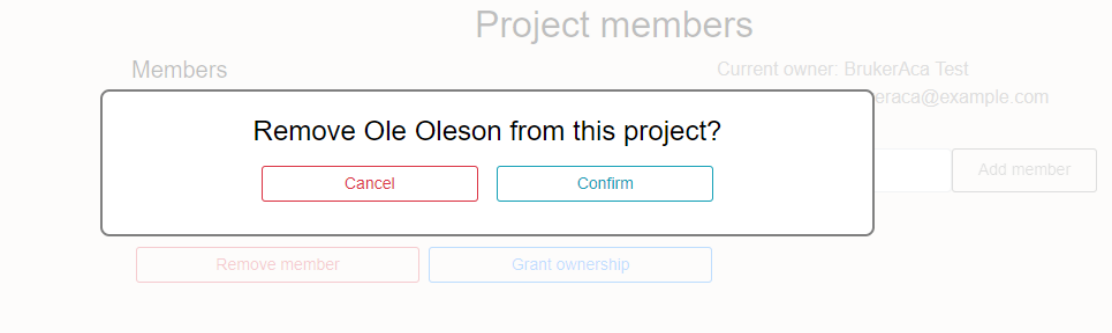
I siden med medlem oversikten kan en bruker finne informasjon om eieren til prosjektet og alle medlemmene i prosjektet, se figur 56 for resultat. Hvis brukeren er eieren av prosjektet, vil de se flere funksjonaliteter. De vil da ha muligheten til å legge til nye medlem i prosjektet, fjerne en bruker fra prosjektet, eller gi en ny bruker eierskapet over prosjektet. Se figur 57 for resultat. Hvis eieren prøver å slette eller gi bort eierskapet så vil det komme en pop-opp bekreftelse boks som spør eieren om de virkelig vil gjøre dette, se figur 58 for resultatet av denne bekreftelse boksen.



Figur 56: Oversikt over medlemmer og eier



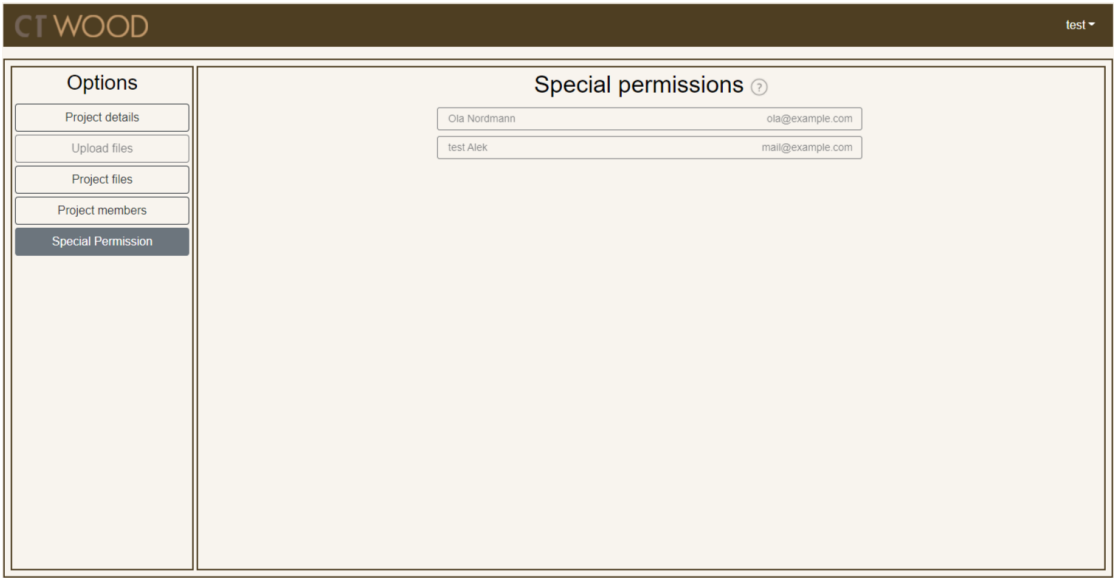
Figur 57: Eieren av prosjektet sin medlem side



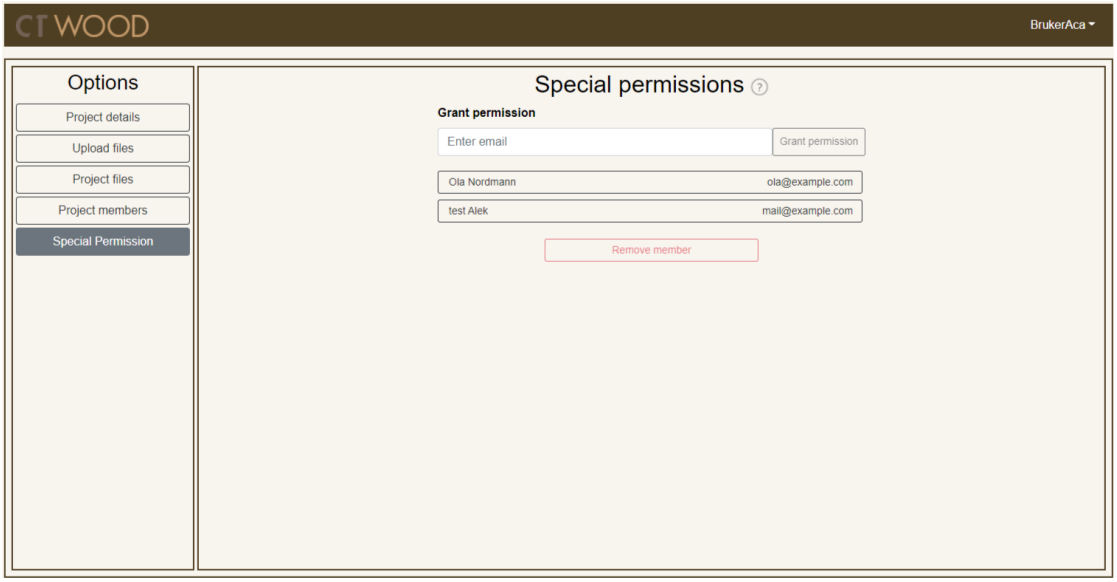
Figur 58: Bekreftelse boksen som kommer opp når eieren prøver å fjerne en bruker fra prosjektet

4.3.5.5 Spesial tillatelse

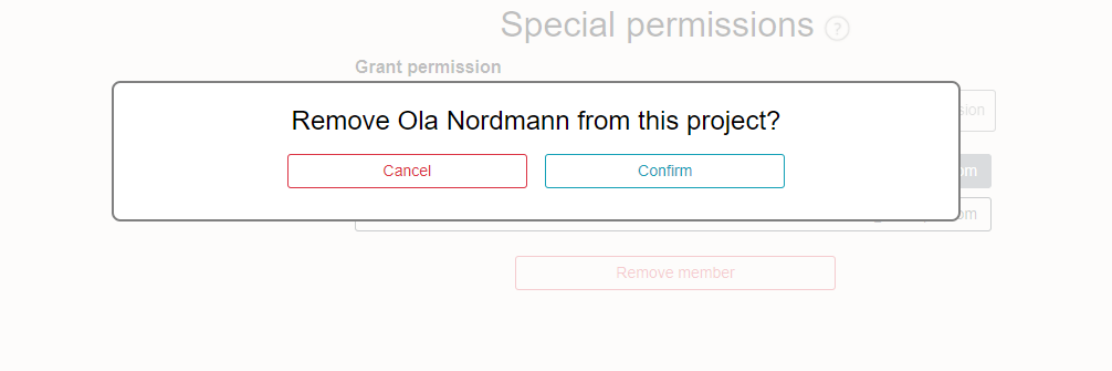
I denne siden vil det være en oversikt over alle brukerne som har blitt gitt spesial tillatelse til å kunne se og laste ned filer fra prosjektet hvis det er privat, se figur 59. Hvis brukeren er et medlem eller er eieren av prosjektet vil de ha muligheten til å gi eller ta vekk rettigheter fra brukere. Se figur 60 for et medlem sitt synspunkt på denne siden. Når en bruker prøver å fjerne rettighetene til en annen bruker så vil det komme en bekreftelse boks. Se figur 61 for å se dets resultat.



Figur 59: Prosjekt spesial tillatelse som en user



Figur 60: Spesial tillatelse oversikt siden i ett prosjekt



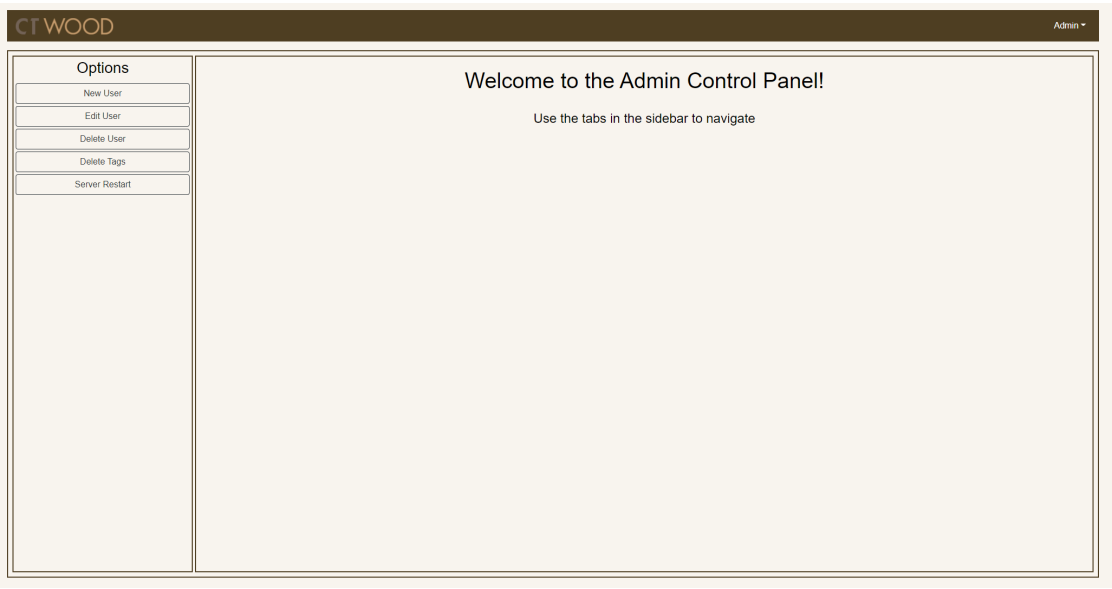
Figur 61: Bekreftelse boksen på spesial tillatelse siden

4.3.6 Admin kontrollpanel

Admin siden er satt opp slik at alt er på samme adresse. Når brukeren benytter knappene i sidebaren endres innholdet på ruten til høyre til det aktuelle innholdet.

4.3.6.1 Admin forside

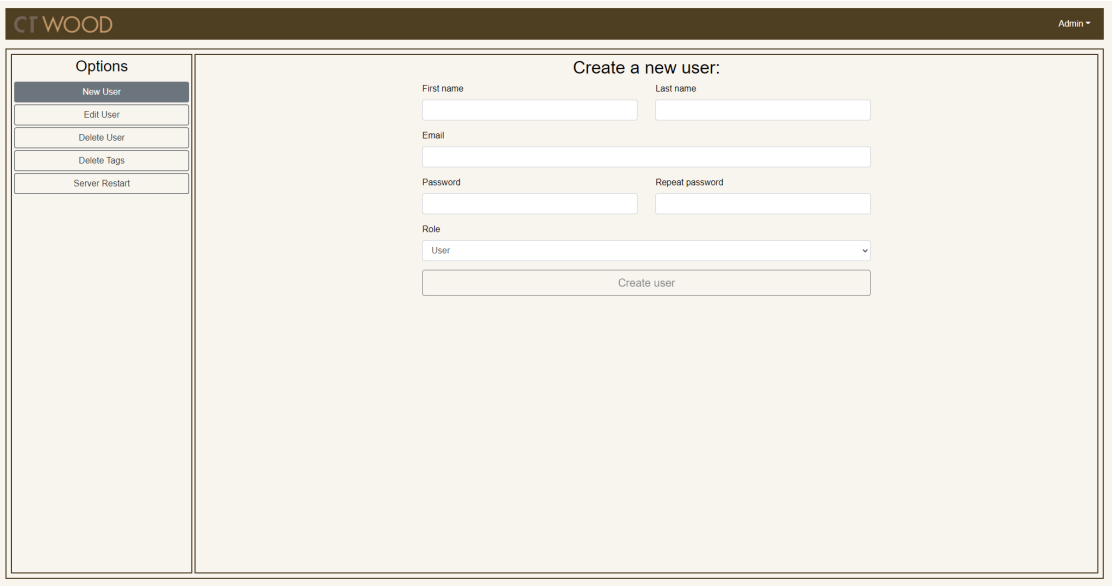
Dette er siden brukeren kommer til etter å ha valgt «Admin Page» i dropdown menyen. Det er bare en velkomstside og er ikke inkludert i sidebar. Se Figur 62.



Figur 62: Forsiden til admin-kontrollpanelet for en admin bruker

4.3.6.2 Opprett ny bruker

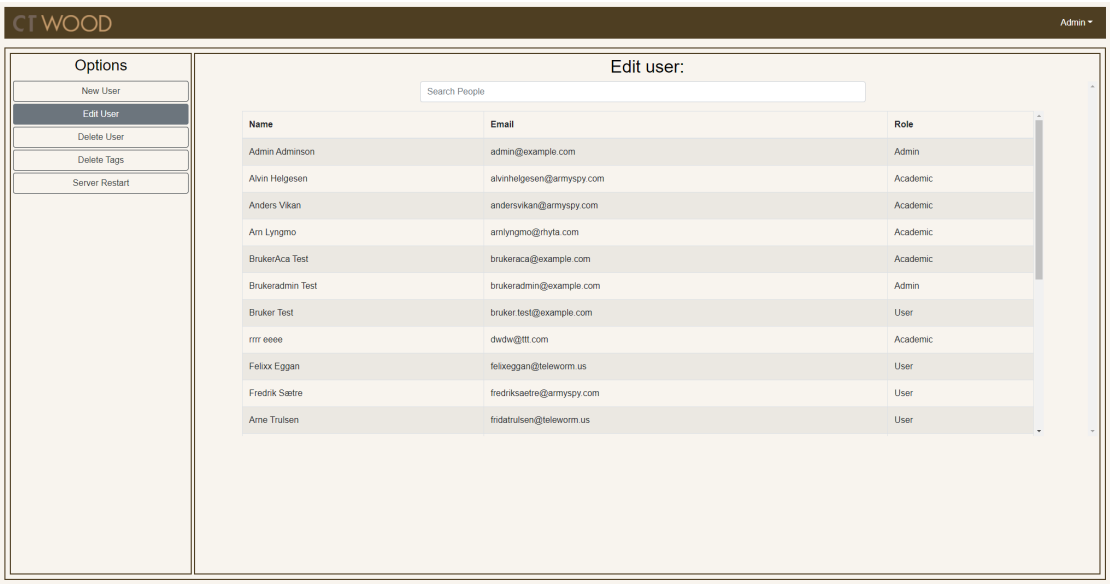
En administrator har den rollen som har tillatelse til å opprette brukere i systemet. Ved å trykke på «New User» får brukeren opp denne skjermen. Den inneholder alle input-felt for den informasjonen som treng for å opprette en ny bruker: fornavn, etternavn, epost, passord og rolle. Hvis brukeren har begynt å fylle ut et felt og innholdet er ugyldig vil det komme opp en feilmelding under knappen som varsler om hva som er problemet. Det kan være begrensninger på lengde, at eposten ikke er korrekt formatert eller at passordene ikke er like. Når siden først lastes inn, vil knappen være avslått så brukeren ikke kan sende en ukomplett forespørsel. Med en gang alt som er skrevet inn er gyldig vil den aktiveres.



Figur 63: Siden på admin-kontrollpanelet for å opprette ny bruker

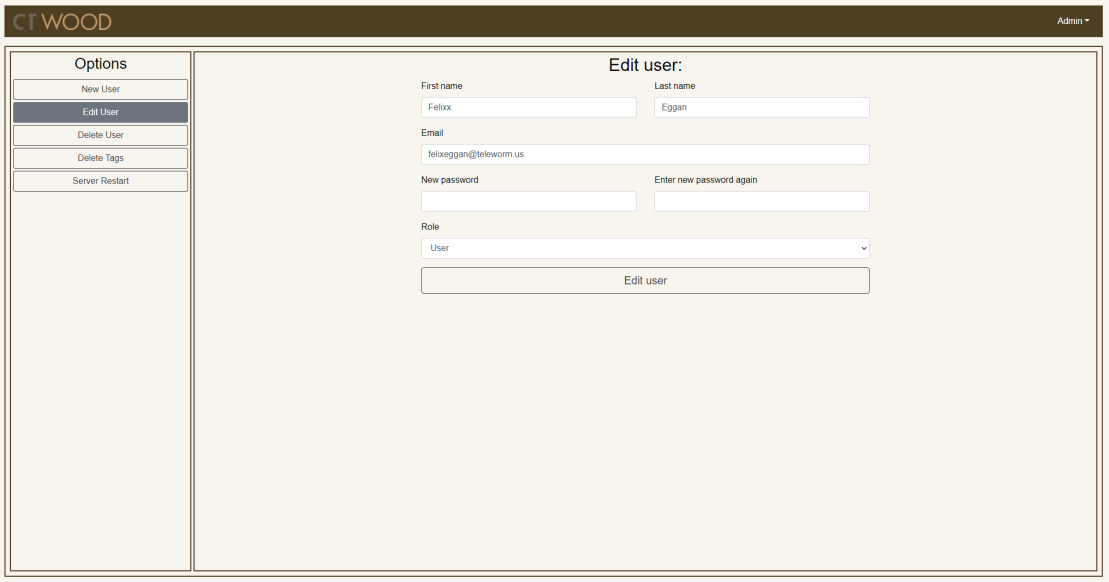
4.3.6.3 Redigere brukerdetaljer

Ved å klikke på «Edit User» i sidebaren komme man til denne siden. Se Figur 64. Der kan en administrator endre alle detaljene til en bruker. Denne siden består av to forskjellige undersider: søkesiden og redigerings-siden. Søkesiden består av et søkefelt og en skrollbar tabell som inneholder navn, epost og rollen til alle brukerne. I søkefeltet øverst på siden kan man skrive inn tekst for å begrense brukerne som vises i resultat-tabellen. Det er mulig å søke etter samtlige kolonner og resultatet vil oppdatere seg samtidig som det skrives inn.

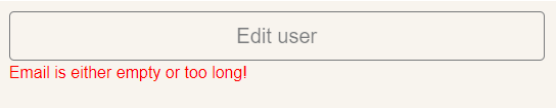


Figur 64: Skjermen for å velge en bruker

Tabellradene er klikkbare og fører til en ny side der brukerdetaljene kan endres. Se Figur 65. På denne siden kan administratoren endre fornavn, etternavn, epost og passord og rolle til brukeren. Hvis passordet ikke skal endres er det ikke nødvendig å fylle inn noe i passordfeltene. Knappen nederst på siden er deaktivert hvis noen av feltene mangler eller passordene er ulike. Når det blir fikset blir den klikkbar igjen. Under knappen vil det også komme en feilmelding som forteller hva som er problemet. Se Figur 66. Når alt er i orden, kan brukeren klikke på knappen og bli sent tilbake til siden for å velge brukere igjen.



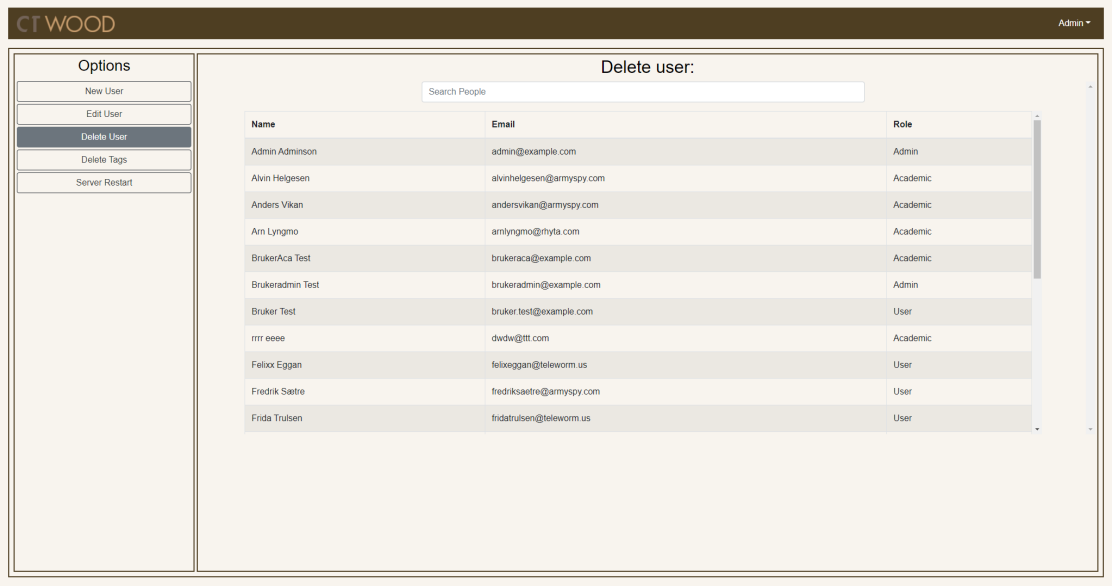
Figur 65: Skjermen for å redigere brukerdetaljer



Figur 66: Errormelding hvis epost mangler

4.3.6.4 Slett bruker

Ved å trykke på «Delete User» i navbaren kommer brukeren til denne siden. Se Figur 67. Siden ser identisk ut som den delen av Edit User der man søker opp en bruker, men er her litt annerledes. Å søke etter brukere fungerer på akkurat samme måte; skriv inne navn, epost eller rolle for å filtrere brukerne. Siden fører ikke til en ny side, men åpner i stedet en dialogboks når en bruker-rad klikkes på. Se Figur 68. Dialogboksen spør brukeren om å bekrefte valget eller avbryte. Denne boksen er der for at det skal være vanskeligere å slette brukere, siden det ikke er noen måte å angre operasjonen.

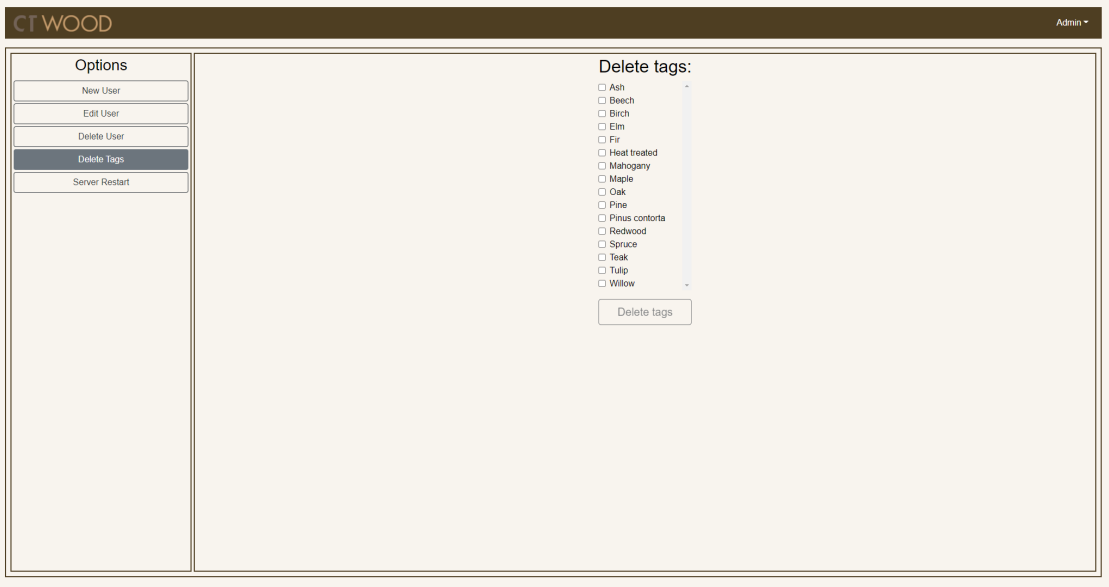


Figur 67: Delete User side

Figur 68: Dialogboks for å bekrefte sletting av bruker

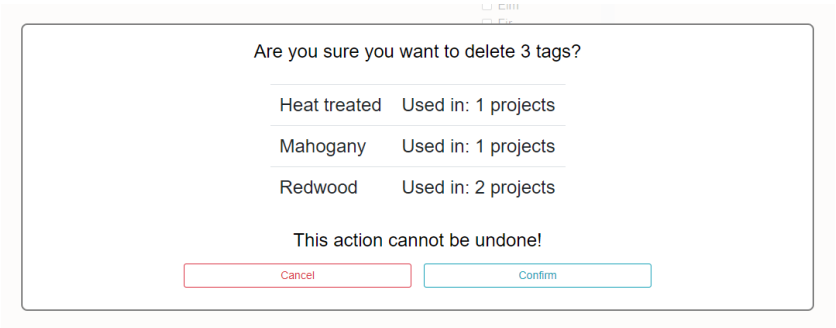
4.3.6.5 Slett tagger

Dette er siden brukeren kommer til etter å trykke på «Delete Tags» knappen. På denne siden er det en liste av alle eksisterende tagger og avkryssingsbokser. Se Figur 69 Dette tillater brukeren å velge flere tagger om gangen til sletting. Knappen her blir også deaktivert så lenge ingen tagger er valgt.



Figur 69: Siden for å slette tagger

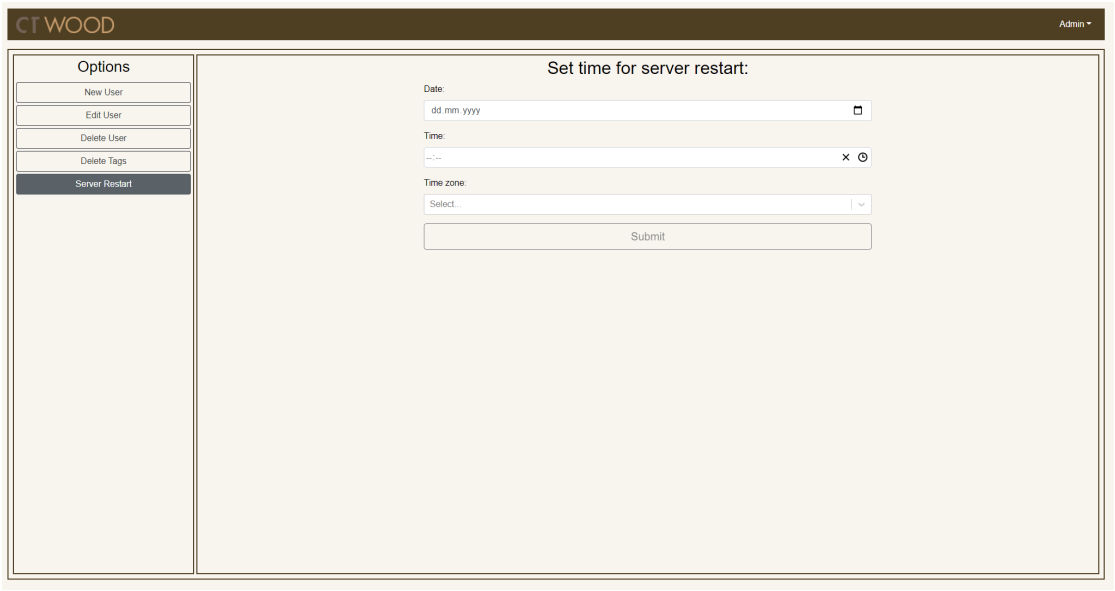
Når brukeren velger en eller flere tagger og trykker på knappen nederst, kommer det opp en dialogboks. Se Figur 70. Der spør den brukeren om bekreftelse før sletting og lister opp taggene og i hvor mange prosjekter de blir brukt i. Dette er gjort for å gi brukeren et ekstra varsel før han sletter tagger som for er mye brukt i prosjekter. Denne dialogboksen har også en knapp for å fortsette og en for å avbryte.



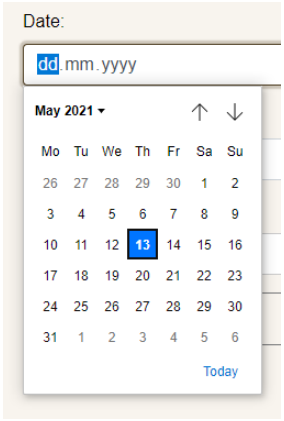
Figur 70: Dialogboksen for å bekrefte sletting av tagger

4.3.6.6 Omstart av server

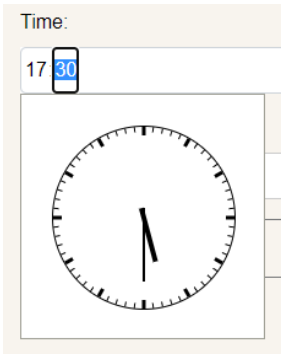
Knappen ≪Server Restart≫ sender brukeren til denne siden. Se Figur 71. Her kan brukeren sette et tidspunkt som serveren skal starte på nytt. Siden består av tre felt; dato, tid, og tidssone. Dato-feltet åpnet en pop-up boks som ser ut som en kalender der brukeren kan klikke på datoen for å velge den. Se Figur 72. Under datoen kan brukeren skrive inn tidspunktet for omstart. Mens brukeren skriver inn, blir klokkeslettet illustrert i en boks under feltet. Se Figur 73. Nederst er en nedtrekksmeny hvor brukeren kan velge tidssone. Se Figur 74.



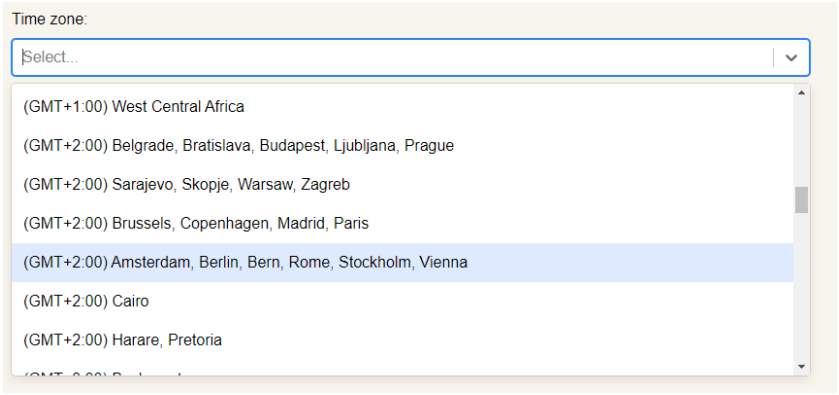
Figur 71: Siden for å sette tid til server omstart



Figur 72: Kalender pop-up for å velge dato



Figur 73: Felt for å skrive inn klokkeslett



Figur 74: Nedtrekksmeny for å velge tidssone

4.4 Prosjektgjennomføring

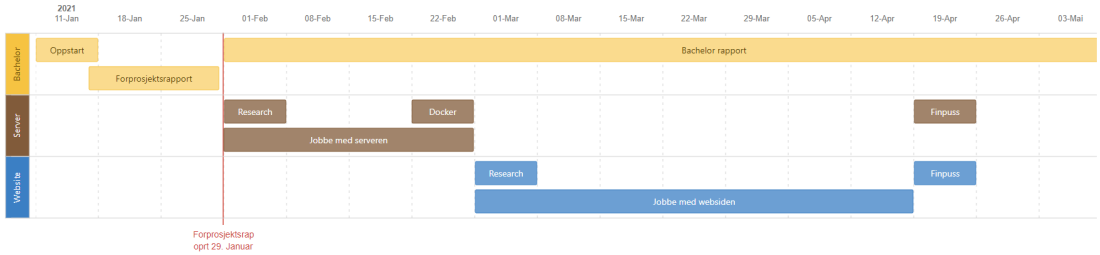
4.4.1 Scrum

I kapittel 3 skrev vi om hvordan vi planlagte gjennomføringen av Scrum for dette bachelorprosjektet. Planen var å bruke sprinter på en uke og dette har vi fulgt gjennom hele utviklingsprosessen. Vi hadde også planlagt sprint gjennomgang og sprint tilbakeblikk på fredager. Etter det første møtet med veileder og oppdragsgiver, innså vi at vi måtte flytte sprint tilbakeblikk til torsdag på ukene med møte. Dette var så vi kunne presentere det på møtet. Møtet annenhver uke gjennomførte vi som planlagt. På grunn av kort tid mellom hvert møte fikk vi også rask tilbakemelding og innspill på prosjektet av begge. Om det var noe som vi trenge å gå igjennom før neste møte var oppdragsgiver også alltid tilgjengelig for møte utføre de planlagte møtene. Ellers ble e-post brukt for mindre spørsmål til oppdragsgiver og veileder.

Fra starten at prosjektet var vi ikke klar over at Jira issues skulle formuleres på en spesiell måte. Vi lagde issues mer fra vårt perspektiv enn fra brukerens perspektiv. Etter hvert fikk vi avklaring fra veileder om hvordan dette burde gjøres. Det skulle være en konkret funksjonalitet fra brukerens ståsted. Etter det prøvde vi å gjøre navnet på issues mer som brukerhistorier.

4.4.2 Generelt

Planen vår var å først jobbe med for forprosjektrapporten, deretter gjør backend ferdig og til slutt frontend ferdig. Vi tenkte også å jobbe med bachelorrapporten i løpet av hele perioden som vist på figur 75. I stedet for ble resultatet at vi jobbet med forprosjektrapporten etter planen, men resten ble ikke etter planen. Vi startet med Docker så fort som vi fikk backend til å kjøre. Vi startet også tidligere med frontend som startet 15.februar istedenfor 1.mars som planlagt. Deretter jobbet vi med både frontend og backend fram til 1.mai. Vi jobbet litt med rapporten, men startet ikke fult på den før 1.mai.



Figur 75: Gantt diagram

4.5 Brukertesting

Vi har hatt brukertester på fire forskjellige personer. Ei tredje års student og tre i en alder rundt 50. Ingen av deltakerne hadde noe kunnskap om nettsiden fra før. De fikk informasjon om bakgrunnen til nettsiden om hva den er for og relevant informasjon angående oppgaven de var på. Det var seks oppgaven relatert til rollen *bruker*, fire til *akademiker* og tre til *administrator*. Oppgavene gjekk ut på å uføre enkle oppgaver som er laget for å se hvordan en bruker faktisk bruker nettsiden. Om deltakeren slet med å komme seg videre fikk de et hint. To av testene ble gjort igjennom nettet ved å dele skjermen og to ble gjort fysisk. Vi fortalte deltakerne at de skulle tenke høyt og gi tilbake melding om det var noe som kunne vært forbedret. Tiden ble tatt på hver test for å se om det var en del på nettsiden som skilte seg ut.

Oppdragsgiver har også hatt tre brukertester. Dette var de tre første som ble tatt og det var problemer med alle tre. Det er usikkert hva informasjon de fikk i forkant av testen og mens testen foregikk.

4.5.1 Testresultat

Resultatet som kan bli sett på vedlegg C der testene for Jose, Margot og Niclas ble gjort av oppdragsgiver, og Stein, Bjørn, Ida og Linda ble gjort av oss. På resultatet kan vi se på tidene at de fleste testene gjekk fort med unntak av oppgave 4 for rollen bruker og oppgave 3 for rollen akademiker. Med unntak av de to oppgavene kan vi si at siden er enkel å forstå og bruke.

4.6 Kjente feil og mangler

Siden starten av prosjektet var vi usikre på om vi kom til å ha tid til å komme i mål med alt oppdragsgiveren og vi ville ha. Vi har kommet i mål med mer enn vi selv hadde trodd, men har noen mangler som enten kom av at vi ikke fant utav det eller bare ikke hadde tid.

4.6.1 Kjente mangler

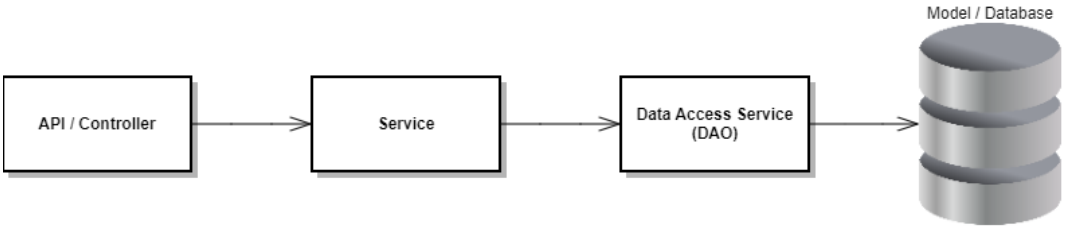
Kjente mangler vi har i prosjektet er tagging av filer, kan ikke slette eller endre på filer på filserveren, en bruker kan ikke endre dato eller navn på et prosjekt, vi kan ikke vise DICOM og Tiff filer som et bilde, vi viser ingen timer til alle brukere når serveren skal til å starte på nytt og vi mangler tester til applikasjonen.

4.6.2 Kjente feil

Når en bruker starter systemet på nytt så setter brukeren tiden når den skal omstarte. Denne timeren fungerer, men kan være noen sekunder feil. Om serveren faktisk restarter er noe vi er usikre på. Serveren kjører koden som skal starte den på nytt, men vi ser ingen tegn til at den har startet på nytt. Nettsiden fungerer heller ikke på safari nettleseren. Når nettsiden bli åpnet på safari vises det bare en blank side. Når *JSON web token* går ut på dato og man navigerer igjennom en sidemeny blir man ikke sendt tilbake til log inn.

4.7 Backend

Backend ble nesten ferdig med små mangler og en kjent feil. Til backend bruker vi et (API → Service → DAO → Model) system som vist på figur 76. På denne måten har vi et ryddig system til backend. I dette systemet vil API ta seg av REST forespørsel funksjonaliteten, service vil ta seg av generell logikk, DAO vil ta seg av database logikk og Model er objekt klasser. Denne strukturen er i utgangspunktet basert på MVC designmønsteret nevnt i kapittel 2.4.3.



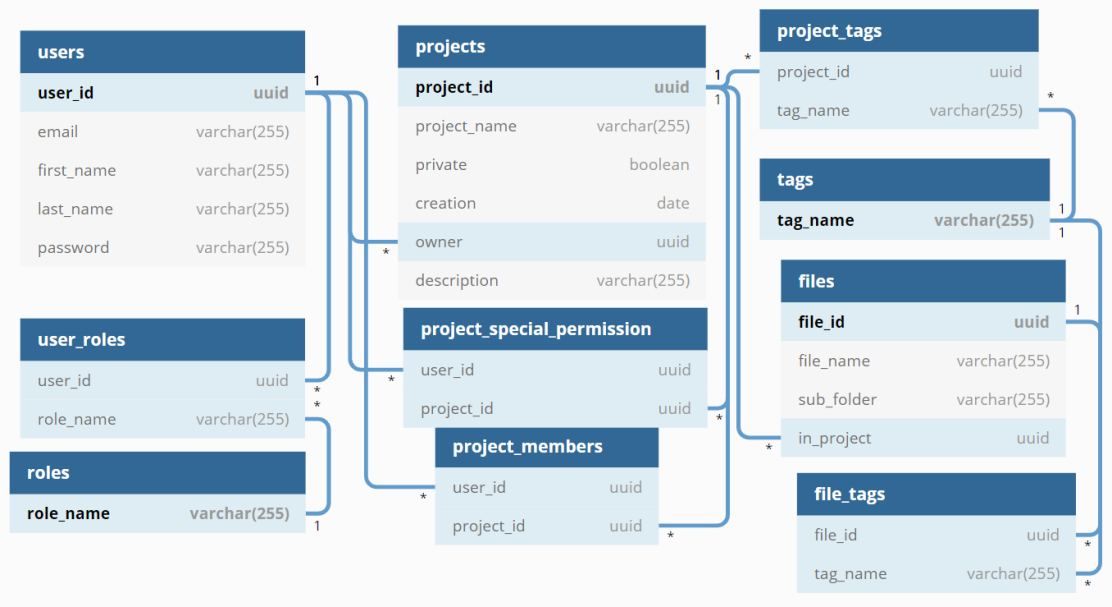
Figur 76: Resultatet av backend modellen.

4.7.1 Database

Databasen vår ble laget ved hjelp av nettsiden dbdiagram.io som forklart i kapittel 3.3.3. Resultatet av databasen som vist i figur 77 ble skrevet som kode og eksportert som postgreSQL kode som vi kjørte inni IntelliJ for å generere databasen. Databasen består av tabellene *users*, *roles*, *projects*, *tags* og *files*. Der *user roles*, *project special permission*, *project members*, *project tags* og *file tags* er mange til mange forhold, og users har en til mange forhold til projects og projects har en til mange forhold med files.

4.7.1.1 Datatilgangstjeneste

Datatilgangstjeneste (DAO) er det vi kaller klassene vi bruker for å få tilgang til informasjon i databasen. Jobben til denne klassen er bare å hente, endre eller fjerne objekt fra databasen. Det meste av sikkerhet skjer i klassene som bruker DAO klassene våre som heter service. Ellers bruker DAO klassene våres en *EntityManager* fra bibliotektet EclipseLink JPA (Mer om dette i kapittel 3.6.8). Ved hjelp av EntityManger bruker vi *NamedQuery* til å hente informasjon fra databasen (NamedQuery er ferdig definerte SQL queries som ligger i en database modell klasse, se eksempel på figur 16).



Figur 77: Database resultat

4.7.2 REST kontroller

Rest kontrolleren ble inndelt i 4 deler: en for autorisasjon som alle kan bruke uansett om du er innlogget eller ikke, en for rollen bruker som alle med rollen bruker og oppover kan bruke, en for rollen akademiker som administrator også kan bruke og en for administrator. Ellers har vi noen ekstra restriksjoner på noen REST forespørsler som krever enten å være eieren, medlem eller ha spesial tillatelse til et prosjekt. Det er bare en eier eller medlem av et prosjekt som kan gjøre endringer, men bare eieren som kan bytte eier i et prosjekt. Om et prosjekt er privat vil ikke personer med akademiker eller bruker kunne se filene til et prosjekt. Da kan en bruker få spesial tillatelse som gjør at brukeren får tillatelse til å se og laste ned filer fra det prosjektet. En bruker med rollen administrator derimot har ingen restriksjoner og kan gjøre alt. Dette inkluderer også noen administrator sider.

REST kontroller metode oppbygging

For å gi tilbakemeldinger til frontend bruker vi en klasse som heter *ResponseEntity* av Spring rammeverket. Denne klassen lar oss sende HTTP-responser (Mer om HTTP responser og vår bruk med dem i kapitel 3.8.1) til frontend med en status kode til å forklare om REST forespørselen gjekk bra eller dårlig. Et problem vi hadde var å kunne respondere til flere problem fra en metode

siden en metode bare kan responder med en klasse. Vi fikset dette med å bruke exceptions. Så om noe galt skjer kaster vi en exception og fanger den i REST kontrolleren og svarer frontend med en HTTP-status etter hva problemet er. På denne måten kan vi enkelt gi svar uansett hva som skjer på backend.

4.8 Installasjon og konfigurering av applikasjonen

Et viktig krav fra kunden var at applikasjonen skulle være enkel å sette opp og konfigurere. Om dette var noe kunden ville bruke så skulle noen andre ta over og videreutvikle applikasjonen.

For å gjøre det enkelt å endre på forskjellige konfigurasjoner til backend er alt samlet inni filen *application.properties* som vist på figur 78. Her kan man endre på alt fra domene til serveren, databasen, filserveren, maks størrelse på filer backend vil godta, hvor lenge en bruker vil være innlogget og mer. For sikkerhets skyld er noen av feltene kryptert. For å kryptere tekst må de bli kjørt igjennom en algoritme. Deretter må resultatet bli satt inni: *ENC()*. Passordet serveren bruker for å dekryptere dem blir satt i: *jasypt.encryptor.password*. For frontend er det bare IPen som backend kjører på som kan bli endret på. Den blir endret inni *App.js*.

For å gjøre det så enkelt som mulig å kjøre applikasjonen bruke vi Docker til å kjøre både backend og frontend slik at applikasjonen kan bli kjørt på alle maskiner uansett operativsystem og innstillinger. For å kjøre applikasjonen må bare to forskjellige docker-compose skript bli kjørt som vil pakke og bygge prosjektet. Prosjektet blir bygget med kommandoen: *docker-compose -f [yml fil navn] build*. Deretter er det enkelt å starte den med: *docker-compose -f [yml fil navn] up*.

```
#localhost / database
spring.datasource.url=jdbc:postgresql://database:5432/postgres
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.username=ENC(68X0ExTViUe8r8q2acQa8oModvvpf/oaqwC2h4t0F19aSyx+VCAOnGtoAXoxSGRx)
spring.datasource.password=ENC(MdcLScamd8bOhyqg0Sz1c5+1soS30U8n44InkQ9god2Xp/Qp7DfXRR57BrqIgtt3)
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQL9Dialect
spring.jpa.hibernate.ddl-auto=none
## MULTIPART (MultipartProperties)
# Enable multipart uploads
spring.servlet.multipart.enabled=true
# Threshold after which files are written to disk.
spring.servlet.multipart.file-size-threshold=2KB
# Max file size.
spring.servlet.multipart.max-file-size=200MB
# Max Request Size
spring.servlet.multipart.max-request-size=215MB

## File Storage Properties
# All files uploaded through the REST API will be stored in this directory
#To add new directories it has to be added here, in FileStorageProperties.Class and in FileStorageService.Class constructor and FileStorageService method.
#storeFileInDirectory method has control over where files are saved.
#Example of project directory: Archives/YY-MM-DD_ProjectName/images/DICOM
file.upload-dir=/Archives
file.document-dir=/Documents
file.image-dir=/Images
file.log-dir=/Log Files
file.dicom-dir=/CT-Dicom
file.tiff-dir=/CT-Tiff

file.user=ENC(ZciiKF4jKLN12pIWYKRj+uzFJig+8pUj4ugHsd4aaYO2cuSNtZPb00Vru4ts4sRp)
file.pass=ENC(G9bucBSolx/B3JB4cwD/jNIzEOhTw9ueu4P2xxFWg81UP9cS6ImT+NKuPQUtMY2X)
file.domain=ENC(a7X4BFYuJ6z7M80ozjjJjiF5W5LVfOYevEJSZmp4bzlCoEPCBcFWT9Lp4XQfc280)
file.url=ENC(syfSE/N7Hy4Wb4hVz/rhC3UqP7DSHpGhkihQwC0bX7KJWY1+2UYiPNS2x7AS3cUFbYdyQev/hl2s2y1PTPkbOqn/5DuZFhhTivzThBfqEdM=)
prop.jwtKey=ENC(Jj75feZfhtH/Lh4LYAcmXjHLVa6L6IVFRWhp5g6PthxAMV7t2DqcdyvoZ+pEfhKXqBR2almc8G5PQxnVdimhDQ==)
#12 hours: 60 * 60 * 12
#After this time has expired since a user has logged in the user will be logged out.
prop.jwtLifetimeInMin=43200
prop.domain=localhost
prop.port=3000
jasypt.encryptor.password=${SECRET_ENCRYPTION_KEY}
```

Figur 78: application.properties for å endre konfigurasjoner til backend.

5 Drøfting

5.1 Resultat evaluering

Resultatet til nettsiden ble nesten fullført med mindre mangler. Med nettsiden kan en bruker mye enklere lage nye prosjekter og laste opp filer til dem. Filene vil også bli automatisk sortert til rett mappe på filserveren. Med vårt resultat kan også folk som studenter få tilgang til å studere prosjekter som ikke er private.

5.1.1 Sikkerhet

Når det kommer til sikkerhet, var det ikke satt mange krav av arbeidsgiver utenom krav til hva visse roller i systemet skal kunne gjøre eller ikke. Derfor har vi i gruppen selv valgt sikkerhetstiltak for å sikre sensitiv informasjon og data.

5.1.1.1 Rolle-system

For å sikre at bare noen roller skal få gjøre visse ting i systemet har vi laget et rollesystem. Dette består av 3 roller: *administrator*, *akademiker* og *bruker*. Vi har ut ifra dette brukt Spring Boot sine funksjoner til å lage fire forskjellige Java klasser, en til hver rolle og en til innlogging. Dermed ved hjelp av Spring Boot konfigurasjon og merknader kan vi sette opp en klasse med alle REST forespørslene til for eksempel en administrator. Se eksempel på figur 79. Nå kan vi dermed trykt lage alle forespørslene til en administrator inni denne klassen og ingen uten denne rollen kan bruke REST forespørslene inni klassen.

5.1.1.2 Passord-sikkerhet

For å sikre passord bruker vi også Spring sitt innlogging system og passord krypterer. Det vi trenger å gjøre for å lage et innlogging system er å lage en instans av en *AuthenticationManager*. Deretter når en bruker prøver å sende en innlogging REST forespørsel tar vi passordet og brukernavnet og sender dem til *AuthenticationManager*. Den vil da automatisk gjøre jobben med å kjekke om brukeren er i systemet og om passordet er riktig. For å sikre passordene bruker vi *BCryptPasswordEncoder*. Den bruker en BCrypt algoritme også med tilfeldig generert salt for å kryptere passordet til brukeren. Dette kan bare bli gjort en vei, som vil si at databasen holder på uleselige passord. Saltet blir brukt slik at uansett om to brukere bruker det samme passordet vil de ende opp å bli forskjellige i databasen. Måten Spring sjekker om passordet brukeren prøver å logge inn med er riktig er å prøve å bruke den samme algoritmen på passordet brukeren sendte inn og se om det blir likt som det i databasen. Om de er like så blir innloggingen godkjent.

5.1.1.3 Json web token

Når en bruker er innlogget, er det viktig for brukeren sin nettleser å huske det. For dette bruker vi en *json web token* (jwt). Dette er nøkken brukeren får tilbake fra serveren etter brukeren har logget inn. Den vil automatisk bli lagret hos brukeren og automatisk sent til serveren for hver REST forespørsel som blir gjort. På denne måten kan vi trykt la en bruker bruke nettsiden uten å måtte logge inn igjen for hver handling. Jwt blir laget med å ta visse bruker detaljer i lag med et hemmelig passord og en dato/tid. Dermed kan serveren alltid vite hvilken bruker som sender en REST forespørsel. Datoen/tiden er bestemt i systemet sine innstillinger og bestemmer hvor lenge jwt er gyldig.

5.1.1.4 HTTPS/SSL

HTTPS sertifikat er noe vi ønsket å ha på nettsiden, men på grunn av mangel av domene har vi ingenting å ha sertifikatet på. Dermed er ikke brukeren sine detaljer trygge når forespørslene blir sendt. HTTPS bruker *Secure sockets layer* (SSL) til skjuler alt sensitiv informasjon. Vi har derimot en *Reverse proxy* Docker konteiner. En slik konteiner er for å fange opp alle forespørsler som kommer til serveren og sende de videre til riktig plass på serveren. Den skal bli satt opp med SSL sertifikatet, men siden vi ikke har det bruker vi ikke denne konteineren.

```
@RequestMapping("/admin")
@RestController
public class AdminController {

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable().authorizeRequests()
            .antMatchers( ...antPatterns: "/admin/**").hasRole(Role.ADMIN)
            .antMatchers( ...antPatterns: "/academic/**").hasAnyRole(Role.ADMIN, Role.ACADEMIC)
            .antMatchers( ...antPatterns: "/user/**").hasAnyRole(Role.ADMIN, Role.ACADEMIC, Role.USER)
            .antMatchers( ...antPatterns: "/auth/**").permitAll()
            .anyRequest().authenticated()
            .and().addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class)
            .exceptionHandling().authenticationEntryPoint(new HttpStatusEntryPoint(HttpStatus.UNAUTHORIZED));
    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    http.cors();
}
```

Figur 79: Her setter vi opp rolle sikkerheten inni administrator REST kontrolleren og konfigureringen inni klassen *SecurityConfiguration*.

5.1.2 Brukervennlighet

Det er viktig å teste brukervennlighet og for dette har vi brukt 10 kvalitetstester for universell utforming slik at nettsiden er vennlig uansett alder, funksjonsevne og utdanningsnivå.

1. **Tastaturnavigering**
   Det skal være mulig å navigere seg igjennom hele nettsiden med tastaturet og kunne bruke all funksjonalitet. Dette blir gjort med tab, enter og space. Det er mulig å gjøre alt dette på nettsiden vår, men det er noen navigasjonsmenyer som ikke blir fremhevet når knappene er markert som gjør det litt vanskelig til tider. Siden man kan navigere overalt på nettsiden med tastaturet er tastaturnavigering godkjent.

2. **Forstørring og responsivt design**
   Alt på nettsiden fungerer med både 200 og 400 prosent zoom. Alt av tekst og funksjonalitet ble forstørret slik som det skulle. Med responsivt design fungerer også nettsiden slik som den skulle der alle elementer holdt seg der de skulle når nettleseren ble smal. Nettsiden fungerer også bra på mobil. Denne testen er godkjent.

3. **Farger og kontrast**
   For at folk med dårlig syn skal kunne bruke nettsiden bra må farger og kontraster mellom bakgrunn og tekst være minimum 3:1 for stor tekst og 5:1 for vanlig tekst. Vår nettside har 19,16:1 for stor tekst og 6,89:1 for vanlig tekst. Logoen vår er derimot bare 1,79:1 som ikke er innføre for stor tekst. Siden dette er et mindre viktig element på nettsiden, velger vi å godkjenne testen.

4. **Overskrifter**
   Overskriftene på nettsiden er for det meste bra, men er litt hopping mellom overskrift størrelser. Vi bruker for det meste HTML overskrifter fra <h2> til <h4> og de står riktig i forhold til hverandre. Vi velger derfor å godkjenne denne.

5. **Lenker**
   Lenker på nettsiden er for det meste godt markert når du peker med musen over dem. Det er en knapp som ikke er markert bra nok når musen er over den som kan gjøre det vanskelig å se at elementet er klikkbart. Når du bruker tab for å navigere er det også noen elementer i navigasjonsmenyer som ikke blir markert når de er fokusert. Jeg velger derfor å ikke godkjenne denne testen.

6. **Bilder**
   Bilder på nettsiden har relevant informasjon om hva de er. Derfor er denne testen godkjent.

7. **Skjema**
   Alt av skjemafelt blir markert ved musklikk og avkrysningsbokser blir markert ved klikk, så derfor er denne testen godkjent.

8. **Søkefunksjonen**
   Søkefunksjonen gir akkurat de treffene vi forventer. Det er også mulighet å filtrere resultatet og bruke tagger til søket. Derfor velger vi å godkjenne søkefunksjonen.

9. **Sidetitler**
   Fanen til nettsiden har navnet på nettsiden, men det blir aldri markert noe mer om man navigerer rundt på nettsiden. For eksempel om man går innpå et prosjekt burde det stå *CT WOOD - Navn på prosjekt*. Siden dette ikke er tilfellet, blir denne ikke godkjent.

10. **Kodevalidering**
    Vi har testet alle sidene med kodevalidator og kan ikke finne noen feil. I tillegg er:

    - Alle elementer er nøstet korrekt.
    - Alle elementer startes og avsluttes korrekt.
    - Ingen elementer har duplikat attributt.
    - Ingen elementer har identisk verdi på id-attributtet.

    Derfor velger vi å godkjenne denne.

Vi gidde oss selv 8/10 på kravene for universell utforming. Kravet for lenker burde vi hatt i orden, men ellers er vi fornøyde med resultatet. (Sjekk nettstedet ditt selv — Tilsynet for universell utforming av ikt 2021)

5.1.3 Brukergrensesnitt

Vi er generelt ganske fornøyde med brukergrensesnittet, men det er mange små ting som kan være til hindring for en god brukeropplevelse. Nettstedet har et veldig enkelt design og er relativt intuitivt. Noen funksjonaliteter føles litt usynlig, og noen funksjonaliteter kan være ganske vanskelige å finne. Noen knapper og tekstfelt er ikke koblet i lag slik at det ikke er mulig å bruke `Enter` knappen for å automatisk sende informasjonen. Nettsiden er litt inkonsekvent noen ganger med utseendet og hvordan knapper fungerer.

5.1.3.1 Forbedringer av brukergrensesnitt

For å forbedre brukeropplevelsen så må noen funksjoner gjøres mer åpenbare. Dette kan oppnås ved å endre på noen elementer eller vise dem i forskjellige situasjoner for å fortelle brukeren at det er noe de kan gjøre med en funksjon. Knapper og utseendet burde bli så likt som mulig på flest mulige sider, slik at brukeren ikke skal trenge å gjette om hva noe betyr.

5.1.4 Backend

Backend er vi generelt veldig førnøyde med. Fra starten av prosjektet valgte å bruke en vanlig modell for oppbygging av prosjektet (Se figur 76). Dette gjorde at vi hadde en ryddig struktur i backend der vi alltid viste hva en klasse sin oppgave er. Vi tok også tidlig i brukt Postman (Som ble nevnt i kapitel 3.7.1) som vi brukte for å teste REST endepunktene. Vi tok også i brukt en funksjonalitet hos Postman som gjorde slik at vi kunne dele på en samling av REST forespørsler som gjorde at alle på gruppen kunne bruke de samme forespørslene på Postman for testing.

5.1.5 Vår bruk av HTTP-statuskoder

De statusene vi har brukt i prosjektet er: 200-Ok, 204-No Content, 400-Bad Request, 401-Unauthorized, 403-Forbidden, 404-Not Found, 409-Conflict, 410-Gone og 500-Internal Server Error. I løpet av prosjektet har vi diskutert mye om hvilke statuser vi skal bruke hvor og til hva. Hadde vi gjort prosjektet på ny hadde vi endret enn del på bruken.

200-Ok
200-Ok bruker vi til nesten alt som er suksess fra serveren. Om en REST forespørsel blir godkjent og alt gjekk fint sender vi 200-Ok med en body eller uten ettersom hvilken forespørsel som ble sent. For eksempel om man sletter en bruker og alt gjekk bra får man 200-Ok uten en body, men om man skal hente en bruker får man 200-Ok med brukeren som en body med alt informasjonen om brukeren.

204-No Content
Vi brukte originalt 200-Ok til alt som var suksess fra serveren, men etter hvert startet også å bruke 204-No Content. Vi har for det meste brukt denne til for eksempel om en bruker skal hente noe. Alt gikk bra, men ingenting ble funnet. Vi har gått litt fram og tilbake om vi skal bruke 204-No Content eller 404-Not Found for dette.

400-Bad Request
400-Bad Request bruker vi om en bruker sender noe ulovlig inni en parameter. Dette kan være alt fra ulovlige tegn eller om parametere er tomt. Spring Boot sender også 400-Bad Request automatisk om det er mangel på en parameter.

401-Unauthorized
401-Unauthorized brukte vi i starten av prosjektet til alt som en bruker ikke hadde lov til å gjøre, men endret det tidlig til 403-Forbidden istedenfor etter å lest at den er mer normal å bruke til det vi brukte 401-Unauthorized til. Ellers er dette en status kode som Spring Security bruker om en bruker som ikke er autorisert eller ikke har riktig rolle til en REST forespørsel.

403-Forbidden
Som sakt bruker vi 403-Forbidden om en bruker prøver å gjøre noe som brukeren ikke har lov til. For eksempel om brukeren prøver å endre på et prosjekt som brukeren ikke er medlem av eller eier vil brukeren få tilbake 403-Forbidden.

404-Not Found
404-Not Found bruker vi om brukeren prøver å hente noe fra serveren, men det ble ikke funnet. For eksempel om noen prøver å hente en bruker i systemet med en e-post adresse, men ingen brukere med denne e-posten ble funnet gir vi 404-Not Found. Vi har gått litt frem og tilbake om vi skal bruke 404-Not Found eller 204-No Content til dette. Grunne til dette er at 404-Not Found blir mest vanlig brukt om en person prøver å gå innpå en url som ikke er i bruk.

409-Conflict
409-Conflict bruker vi for eksempel om en bruker prøver å legge til noe nytt i systemet, men det allerede eksisterer. Vi bruker altså denne til objekter som skal være unikt som tagger eller e-post til en bruker i vårt system.

410-Gone
410-Gone er også en status vi har brukt istedenfor 404-Not Found. Vi har ikke brukt denne mye, men har brukt den til for eksempel om en bruker prøver å hente en fil, men filen ikke eksistere.

500-Internal Server Error
500-Internal Server Error blir for det meste brukt automatisk om en exception blir kastet på serveren, men ikke håndtert riktig. Da vil en 500-Internal Server Error komme som respons status. Vi har programmert at denne statusen skal bli brukt noen plasser selv også. Eksempel på dette er om serveren ikke klarer å koble seg til databasen. Da vil vi sende 500-Internal Server Error.

5.1.6 Feil og mangler

Funksjonen tagging av filer er ferdig på backend, men vi hadde ikke tid til å legge det til på frontend. Vi fikk også beskjed om at tagging av filer ikke var viktig siden vi allerede hadde tagging av prosjekt.

Mangelen på å kunne slette og endre på filer på filserveren kom av for lite tid. Funksjonaliteten som krever for å få dette til tok mye tid siden biblioteket vi brukte ikke hadde mange gode eksempler. For å få til noe som helst måtte vi bare prøve oss fram med noen få eksempler som gjorde noe lignende det vi ville få til. Siden den viktigste delen av filserveren er å kunne laste opp og hente filer valgte vi å vente med sletting og endring for å se om vi hadde tid til det senere i prosjektet. (Mer om dette biblioteket i kapitel 3.6.10)

Grunnen til mangelen på å kunne endre på dato og navn til et prosjekt kommer av at når en bruker laster opp filer til et prosjekt vil det bli laget en mappe på filserveren som bruker dato og prosjektnavnet som mappe navn. Om en bruker endrer på prosjektnavn eller dato må vi også endre mappe navnet. Det betyr at vi måtte bli ferdig med funksjonaliteten med å kunne endre på filer på filserveren før vi kunne legge til denne funksjonaliteten.

Når det kommer til visning av spesielle bildefiler fant en løsning på DICOM bilder, men denne løsningen krevde lisens som kostet penger for å bruke, så derfor ble denne løsningen ikke lagt til. Vi fant ingen andre løsninger som fungerte. Å hente ut bilder fra en tiff stabel og vise et bilde fra den fant vi derimot ingen løsning som fungerte. Vi klarte å få ut et bilde fra stabelen, men det var problemer med denne løsningen. Det største problemet var at den ikke fungerte på stablene vi fikk fra oppdragsgiver for testing.

Funksjonen med at det skulle vises en timer hos alle brukere før serveren restartet droppet vi siden det viste seg å ikke vær en enkel funksjon å få til. Om timeren hadde blitt lagt til måtte hver bruker sende en forespørsler til serveren ofte for å se om det er planlagt en restart, og få tiden om det er planlagt en snart. Det var også en løsning med å bruke nettleservarsel, men denne løsningen var for kompleks. Derfor valgte vi å prioritere viktigere funksjonalitet istedenfor.

Tester til applikasjonen er også en viktig mangel i prosjektet. Det ble gjort forsøk på å implementere integrasjonstester, men når vi prøvde å kjøre testen ville ikke applikasjonen starte. Vi fant ikke utav hva som var galt og dermed fikk ikke tid til å legge det til. Likt som mye av den andre funksjonaliteten vi ikke la til hadde vi ikke tid til å fortsette med å finne en løsning på det.

5.1.7 Brukertest resultatet

Uansett om brukertestingen for det meste gidde et bra resultat fikk vi avdekket noen feil på nettsiden. Siden vi fikset noen problemet i løpet av testingen ble tidene bedre etter hvert. Dette så vi på som en suksess siden det viser at det vi fisket forenklet nettsiden. Feil som vi fikset, bestod for det meste av feil som gjorde at en oppgave ikke kunne bli gjort. Det var et problem vi så igjennom testene derimot som vi bestemte oss for å fikse. Vi hadde en søkbar til å søke på tagger i et prosjekt. Denne baren skapte mye forvirring når testerne skulle legge til en ny tagg. På grunn av forvirringen og at prosjekter sannsynlig ikke kommer til å ha så mange tagger av vi trenger å søke valgte vi å fjerne denne søkebaren. Dette resulterte i mye raskere tider på denne oppgaven.

5.1.8 Database

I prosjektet var det ingen krav om å bruke en database, men siden applikasjonen skulle være en nettside og krevde innlogging for sikkerhet var det ingen tvil om at vi trengte en database. Vi valgte da også å bruke databasen for å lagre informasjon om forskjellige prosjekter istedenfor å lagre slik informasjon i et XML-dokument. Noe vi diskuterte i starten av prosjektet.

For lagring og henting av informasjon valgte vi PostgreSQL som er et *Relational Database Management System*. Ingen på gruppen hadde noe erfaring med noSQL databaser, så derfor var det utav bildet. En på gruppen hadde derimot jobbet litt med PostgreSQL fra før som gjorde valget enkelt. PostgreSQL er også godt dokumentert og en kjent database som gjør det enklere å finne informasjon om den.

5.1.9 Samarbeid

Samarbeidet til gruppen anser vi som veldig bra. Vi har prøvd å holde oss til faste tider å jobbe i lag igjennom Discord fra klokken 09:15 til 17:00 hver dag. Det har vært få unntak til dette og gjennomsnittlig har vi jobbet normale arbeidsdager. Vi jobbet da i lag hjemme via Discord siden Covid-19 gjorde at skolen var stengt mye av tiden. På grunn av at vi valgte å alltid jobbe i lag med Discord var det enkelt å spør om hjelp om noen trengte det. Vi hadde også daglige standup møter hver dag før lunsj til å kunne spør om hjelp om det var noen problemer. Relatert kompetanse nivået til gruppen var relativt likt med unntak av en som hadde mye mer erfaring med backend programmering. Dette førte til vi hadde to til backend og en til frontend der han ene gikk over til frontend etter hvert. For det meste endte alle opp med å jobbe med det de selv hadde lyst til.

5.1.10 Scrum

Som nevnt i delen om Scrum i kapittel 4.4, hadde vi ikke forstått den korrekte måten å lage issues på. Dette kom av at vi ikke hadde tenkt over koblingen mellom User Stories og issues. Vi var kjent med begge konseptene fra før av, blant annet fra tidligere prosjekter, men også kravspek til bacheloroppgaven selv. Issues ble i starten gjort på en måte som var for å hjelpe oss holde oversikt over hva som var gjort/hva som måtte gjøres. Etter tilbakemelding fra veileder fikk vi det forklart at de skulle formuleres fra oppdragsgivers perspektiv. En mulig grunn til denne misforståelsen kan ha vært at vi ikke har fått en veldig grundig innføring i alle Scrums detaljer før vi begynte å ta metodene i bruk. Både nå og tidligere da vi har jobbet med Scrum, har vi bare startet og lært mens vi arbeider. Dette kan ha ført til at vi bare har antatt at måten vi har gjort det på er korrekt.

I startfasen av prosjektet lagde vi veikart som Gantt diagram. Se Kapittel 4.4.2. Vi endte opp med å begynne med frontend tidligere og fortsatte å jobbe mer både frontend og backend samtidig gjennom hele prosjektet. Vi ser på det som en god ting derimot, siden å arbeide med bare en ting om gangen ville gitt oss saktere framgang. Ved å ha fulgt planen hele veien, i stedet for å revurdere, ville vi nok ikke ha kommet like langt i mål som i dag. I ettertid så er det tydelig at planen ikke var helt gjennomtenkt.

5.2 Problemer gjennom prosjektet

5.2.1 Docker

I starten av prosjektet hadde vi mye problemer med at backend serveren ikke ville fungere på Docker. Først hadde vi et problem med at vi ikke klarte å få kontakt med REST kontrolleren. Dette problemet ble løst etter nesten en uke og vi fant aldri ut hva som løste dette problemet. Etterpå fikk vi et nytt problem der backend ikke klarte å få kontakt med databasen. Vi fant til slutt ut at dette var på grunn av adressen vi satt i *application.properties* til databasen. Det viste seg at når to Docker konteinere skal snakke i lag bruker du navnet til konteineren som IP-adressen. I vårt tilfelle prøvde vi å bruke adressen *localhost*, men som skulle være *database* som var navnet vi gidde til database konteineren.

5.2.2 CORS feilmeldinger

CORS er en mekanisme som brukes for å øke sikkerheten til nettsteder med servere, se kapittel 2.9. Siden vi kjører frontend og backend på to forskjellige porter så var det nødvendig å konfigurere CORS konfigurasjonen i vår backend slik at den ville tillate kommunikasjonen med frontenden vår.

Helt siden vi starten å jobbe med frontend hadde vi problemer med *Cross-Origin Resource Sharing (CORS)* feilmeldinger. Vi hadde ingen tidligere erfaring med CORS og visste dermed ikke hvorfor vi fikk feilmeldinger eller hvordan vi kunne fikse dem. Det var flest feilmeldinger i starten av prosjektet, men det oppsto noen problemer når flere begynte å jobbe på frontenden og når vi byttet over til å bruke en virtuell maskin for serveren. Vi fikk løst problemene våre ved å lære om hva CORS var, og hvordan vi kunne implementere innstillinger for CORS i Spring.

5.3 Arbeidsfordeling

Alle på gruppen jobbet med det de selv hadde lyst til så fordelingen ble slik:

Trym:

- Backend
- Docker
- Database
- Virtuell maskin

Aleksander:

- Frontend

Brage:

- Backend
- Frontend

6 Konklusjon

I dette prosjektet var målet å lage et brukervennlig, web-basert arkivsystem som kan bli brukt av forskere og studenter for å lagre filer på en filserver, for å redusere sløsing av tid og dobbeltarbeid. Vi fikk oppfylt de fleste og viktigste kravene som ble stilt til oss.

Vi er generelt ganske fornøyde med brukergrensesnittet vårt. Det er relativt enkelt å forstå og bruke, men det har noen feil og mangler som har en negativ effekt på brukeropplevelsen. I forhold til brukervennlighet så er det noen mangler som burde ha blitt fikset for å sørge for at brukerne har et bedre opphold på nettstedet vårt.

Arkivsystemet består av en React-basert nettside, en Spring-basert server og en PostgreSQL-basert database. Gjennom prosjektet har vi lært mye om de forskjellige rammeverkene og bibliotekene som ble tatt i bruk, og mye av det vi har lært er veldig relevant informasjon som vil være nyttig å ha med seg videre i livet.

Vi anser prosjektet som en suksess, fordi vi fikk levert en løsning som fungerer bra og som oppfyller de viktigste kravene oppdragsgiveren hadde. Løsningen, selv om den mangler noen funksjonaliteter og har noen feil, fungerer til det den var bygd for. Oppdragsgiver har nå en plass de kan opprette prosjekter, ett system som kan holde på filene til prosjektet, og muligheten til å kunne enkelt nå andre folk sine prosjekter. Siden alt er på en plass og det er enkelt å finne ting selv, vil det bli mindre sløsing med tid og mindre dobbeltarbeid.

Dette prosjektet har vært en veldig nyttig opplevelse for hele gruppen. Vi har fått prøvd oss på mange nye relevante teknologier, vi har fått prøvd å jobbe som et team i et ordentlig prosjekt. Vi har også fått jobbet i ett prosjekt som har tatt i bruk smidig arbeidsmetodikk og lært om hvordan holde et prosjekt gående.

I prosjektet tok vi i bruk smidig arbeidsmetodikk, og vi bestemte oss for å bruke Scrum som vårt smidige prosessrammeverk. Vi brukte Jira og Confluence til å holde styr på all ting Scrum, og vi lærte mye om hvordan et Scrum prosjekt blir gjennomført.

Bibliografi

Agile software development (2. apr. 2021). I: Wikipedia. Page Version ID: 1015587879. URL: https://en.wikipedia.org/w/index.php?title=Agile_software_development&oldid=1015587879 (sjekket 21. apr. 2021).

Apache Software Foundation (2021). Apache Commons – Apache Commons. URL: https://commons.apache.org/ (sjekket 4. mai 2021).

Atlassian (2021). What is Jira used for? Atlassian. URL: https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for (sjekket 21. apr. 2021).

Bratbergsengen, Kjell (30. apr. 2019). relasjonsdatabase. I: Store norske leksikon. URL: http://snl.no/relasjonsdatabase (sjekket 23. apr. 2021).

Brekke, Magne, Alf Kolbenstvedt og Arne Borthne (17. jan. 2018). CT. I: Store medisinske leksikon. URL: http://sml.snl.no/CT (sjekket 5. mai 2021).

Cross-Origin Resource Sharing (CORS) - HTTP — MDN (2021). URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (sjekket 13. mai 2021).

DICOM (13. sep. 2020). I: Wikipedia. Page Version ID: 20761389. URL: https://no.wikipedia.org/w/index.php?title=DICOM&oldid=20761389 (sjekket 7. mai 2021).

Docker (software) (21. apr. 2021). I: Wikipedia. Page Version ID: 1019024652. URL: https://en.wikipedia.org/w/index.php?title=Docker_(software)&oldid=1019024652 (sjekket 21. apr. 2021).

Driessen, Vincent (5. jan. 2010). A successful Git branching model. nvie.com. URL: http://nvie.com/posts/a-successful-git-branching-model/ (sjekket 10. mai 2021).

Echion2 (24. feb. 2010). Visualization of the "history tree" of a revision controlled project, [...] URL: https://commons.wikimedia.org/wiki/File:Revision_controlled_project_visualization-2010-24-02.svg (sjekket 10. mai 2021).

GitLab (2021). What is version control? GitLab. URL: https://about.gitlab.com/topics/version-control/ (sjekket 10. mai 2021).

Hannah, Jaye (8. apr. 2021). What Exactly Is Wireframing? A Comprehensive Guide. URL: https://careerfoundry.com/en/blog/ux-design/what-is-a-wireframe-guide/ (sjekket 6. mai 2021).

Hick's law (22. mar. 2021). I: Wikipedia. Page Version ID: 1013675149. URL: https://en.wikipedia.org/w/index.php?title=Hick%27s_law&oldid=1013675149 (sjekket 5. mai 2021).

HTML (9. apr. 2021). I: Wikipedia. Page Version ID: 1016919858. URL: https://en.wikipedia.org/w/index.php?title=HTML&oldid=1016919858 (sjekket 23. apr. 2021).

HTML & CSS - W3C (2021). URL: https://www.w3.org/standards/webdesign/htmlcss#whatcss (sjekket 23. apr. 2021).

imgscalr – Java Image Scaling Library — The Buzz Media (2021). URL: https://web.archive.org/web/20151209102423/http://www.thebuzzmedia.com/software/imgscalr-java-image-scaling-library/ (sjekket 3. mai 2021).

Introducing JSX – React (2021). URL: https://reactjs.org/docs/introducing-jsx.html (sjekket 21. apr. 2021).

Kumar, Saket (18. aug. 2017). MVC Design Pattern. GeeksforGeeks. Section: Design Pattern. URL: https://www.geeksforgeeks.org/mvc-design-pattern/ (sjekket 6. mai 2021).

Learn to style HTML using CSS - Learn web development — MDN (2021). URL: https://developer.mozilla.org/en-US/docs/Learn/CSS (sjekket 23. apr. 2021).

Manifesto for Agile Software Development (2021). URL: https://agilemanifesto.org/ (sjekket 21. apr. 2021).

Nordal, Ola (12. nov. 2019). markeringsspråk. I: Store norske leksikon. URL: https://bit.ly/3nlsfns (sjekket 23. apr. 2021).

RegisFrey (mai 2010). English: The model, view, and controller (MVC) pattern relative to the user. URL: https://commons.wikimedia.org/wiki/File:MVC-Process.svg (sjekket 6. mai 2021).

Rekhi, Sachin (25. feb. 2018). Don Norman's Principles of Interaction Design. Medium. URL: https://medium.com/@sachinrekhi/don-normans-principles-of-interaction-design-51025a2c0f33 (sjekket 5. mai 2021).

Rossen, Eirik (11. aug. 2020). brukergrensesnitt. I: Store norske leksikon. URL: http://snl.no/brukergrensesnitt (sjekket 23. apr. 2021).

Schwaber, Ken og Jeff Sutherland (2021). Scrum Guide — Scrum Guides. URL: https://scrumguides.org/scrum-guide.html (sjekket 3. mai 2021).

Scrum Ceremonies and Artifacts (31. jul. 2020). Scrum Ceremonies and Artifacts: What Wasn't [...] Geekbot blog. Section: Agile. URL: https://geekbot.com/blog/scrum-ceremonies-and-artifacts-what-wasnt-in-the-scrum-guide/ (sjekket 4. mai 2021).

Sjekk nettstedet ditt selv — Tilsynet for universell utforming av ikt (2021). URL: https://www.uutilsynet.no/regelverk/sjekk-nettstedet-ditt-selv/708 (sjekket 11. mai 2021).

Software requirements specification (20. jan. 2021). I: Wikipedia. Page Version ID: 1001592932. URL: https://en.wikipedia.org/w/index.php?title=Software_requirements_specification&oldid=1001592932 (sjekket 23. apr. 2021).

Spring Framework (2021). URL: https://spring.io/projects/spring-framework (sjekket 23. apr. 2021).

Tyson, Matthew (2. apr. 2019). What is JPA? Introduction to the Java Persistence API. InfoWorld. URL: https://www.infoworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html (sjekket 4. mai 2021).

Virtual DOM and Internals – React (2021). URL: https://reactjs.org/docs/faq-internals.html (sjekket 18. mai 2021).

virtualisering – IT (22. aug. 2020). I: Store norske leksikon. URL: http://snl.no/virtualisering_-_IT (sjekket 13. mai 2021).

What is JavaScript? - Learn web development — MDN (2021). URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (sjekket 23. apr. 2021).

Vedlegg

A Prosjekt-avhengigheter

A.1 Frontend

Navn	Versjon	URL
mobx-react	7.1.0	https://github.com/mobxjs/mobx-react
react	7.1.0	https://reactjs.org/
react-dom	17.0.2	-
react-bootstrap	1.5.0	https://react-bootstrap.github.io/
react-router-bootstrap	0.25.0	-
react-dropzone	11.3.1	https://react-dropzone.js.org/
react-icons	4.2.0	https://react-icons.github.io/react-icons/
react-scripts	4.0.2	https://github.com/facebook/create-react-app
react-time-picker	4.2.1	https://github.com/wojtekmaj/react-time-picker
react-timezone-select	0.10.10	https://github.com/ndom91/react-timezone-select
reactstrap	8.9.0	https://reactstrap.github.io/
sanitize-filename	1.6.3	https://github.com/parshap/node-sanitize-filename
styled-components	5.2.1	https://styled-components.com/
web-vitals	1.1.0	https://web.dev/vitals/
bootstrap	4.6.0	https://getbootstrap.com/

A.2 Backend

Gruppe	Artefakt	Versjon	URL
org.springframework.boot	spring-boot-starter-web	2.4.2	https://spring.io/
-	spring-boot-starter-data-jpa	-	-
-	spring-boot-starter-validation	-	-
-	spring-boot-starter-tomcat	-	-
-	spring-boot-starter-test	-	-
-	spring-boot-starter-security	-	-
-	spring-boot-configuration-processor	-	-
org.projectlombok	lombok	1.18.16	https://projectlombok.org/
org.postgresql	postgresql	42.2.18	https://www.postgresql.org/
org.glassfish.jersey.media	jersey-media-multipart	2.31	https://projects.eclipse.org/projects/ee4j.jersey
org.glassfish.jersey.core	jersey-server	3.0.1	-
org.eclipse.persistence	org.eclipse.persistence.jpa	2.7.8	https://www.eclipse.org/eclipselink/
com.github.ulisesbocchio	jasypt-spring-boot	3.0.2	https://github.com/ulisesbocchio/jasypt-spring-boot
io.jsonwebtoken	jjwt	0.9.1	https://www.jsonwebtoken.io/
eu.agno3.jcifs	jcifs-ng	2.1.5	https://github.com/AgNO3/jcifs-ng/
commons-io	commons-io	2.8.0	https://commons.apache.org/
org.imgscalr	imgscalr-lib	4.2	https://github.com/rkalla/imgscalr

B SQL

B.1 Generering av tabeller (PostgreSQL)

```
CREATE TABLE "users" (
  "user_id" uuid PRIMARY KEY NOT NULL,
  "email" varchar(255),
  "first_name" varchar(255),
  "last_name" varchar(255),
  "password" varchar(255)
);

CREATE TABLE "projects" (
  "project_id" uuid PRIMARY KEY NOT NULL,
  "project_name" varchar(255),
  "private" boolean,
  "creation" date,
  "owner" uuid,
  "description" varchar(255)
);

CREATE TABLE "files" (
  "file_id" uuid PRIMARY KEY NOT NULL,
  "file_name" varchar(255),
  "sub_folder" varchar(255),
  "in_project" uuid
);

CREATE TABLE "tags" (
  "tag_name" varchar(255) PRIMARY KEY
);

CREATE TABLE "roles" (
  "role_name" varchar(255) PRIMARY KEY
);

CREATE TABLE "project_tags" (
  "project_id" uuid,
  "tag_name" varchar(255)
```

```
);

CREATE TABLE "file_tags" (
  "file_id" uuid,
  "tag_name" varchar(255)
);

CREATE TABLE "user_roles" (
  "user_id" uuid,
  "role_name" varchar(255)
);

CREATE TABLE "project_members" (
  "user_id" uuid,
  "project_id" uuid
);

CREATE TABLE "project_special_permission" (
  "user_id" uuid,
  "project_id" uuid
);

ALTER TABLE "projects" ADD FOREIGN KEY ("owner") REFERENCES "users"
↪  ("user_id");

ALTER TABLE "files" ADD FOREIGN KEY ("in_project") REFERENCES "projects"
↪  ("project_id");

ALTER TABLE "project_tags" ADD FOREIGN KEY ("project_id") REFERENCES "projects"
↪  ("project_id");

ALTER TABLE "project_tags" ADD FOREIGN KEY ("tag_name") REFERENCES "tags"
↪  ("tag_name");

ALTER TABLE "file_tags" ADD FOREIGN KEY ("file_id") REFERENCES "files"
↪  ("file_id");

ALTER TABLE "file_tags" ADD FOREIGN KEY ("tag_name") REFERENCES "tags"
↪  ("tag_name");
```

```
ALTER TABLE "user_roles" ADD FOREIGN KEY ("user_id") REFERENCES "users"
↪  ("user_id");

ALTER TABLE "user_roles" ADD FOREIGN KEY ("role_name") REFERENCES "roles"
↪  ("role_name");

ALTER TABLE "project_members" ADD FOREIGN KEY ("user_id") REFERENCES "users"
↪  ("user_id");

ALTER TABLE "project_members" ADD FOREIGN KEY ("project_id") REFERENCES
↪  "projects" ("project_id");

ALTER TABLE "project_special_permission" ADD FOREIGN KEY ("user_id") REFERENCES
↪  "users" ("user_id");

ALTER TABLE "project_special_permission" ADD FOREIGN KEY ("project_id")
↪  REFERENCES "projects" ("project_id");
```

B.2 Generering av roller og en bruker (PostgreSQL)

```
INSERT INTO "roles" (role_name)
VALUES ('ROLE_ADMIN');
INSERT INTO "roles" (role_name)
VALUES ('ROLE_ACADEMIC');
INSERT INTO "roles" (role_name)
VALUES ('ROLE_USER');

INSERT INTO "users" (user_id, email, first_name, last_name, password)
VALUES ('e94fcf40-7147-4b03-a23e-276aee8257b9', 'admin@example.com', 'Admin',
↪  'Adminson',
        '$2a$10$OrdCqh73IsnxM8/pvjwCDuuyQxnL9Jm5jv2YONOGB/pIL4vSaRa/O');

INSERT INTO user_roles (user_id, role_name)
SELECT 'e94fcf40-7147-4b03-a23e-276aee8257b9', r.role_name
FROM roles r
WHERE r.role_name = 'ROLE_ADMIN';
```

C Brukertest

Kilde: Confluence-siden
⋮

(Vedlegg starter på neste side...)

BrukerTest

For rollen User:

Oppgave1:

Logg inn på nettside: http://158.38.101.96:3001/login

Brukernavn: bruker.test@example.com

Passord: password

Oppgave2:

Gå innpå et privat prosjekt.

Oppgave3:

Gå tilbake til heime siden.

Oppgave4:

Du skal finne et prosjekt om "Spruce". Du vet at Lars er en medlem av dette prosjektet.

Oppgave5:

Last ned en DICOM fil fra sub-prosjektet "Norway Spruce 201711" fra dette prosjektet.

Oppgave6:

Logg ut fra nettsiden.

Deltaker	Oppgave 1 tid	Oppgave 2 tid	Oppgave 3 tid	Oppgave 4 tid	Oppgave 5 tid	Oppgave 6 tid	Tilbake melding
Jose	33 sek	10 sek	5 sek	Går ej att finna			Det finns inget prosjekt Spruce
Margot	5	10	2	5	10	20	Ändra Bruker User, dvs titta över språket. Engelska är det som gäller.
Niclas	5	15	2	5	15	1	
Stein	35	23	5	Trodde tag filter oppdatere listen til høyre. Fikk hint. Gikk inn på feil prosjekt.	1:59	5	Søk kan bli komplekst uten noen instruksjoner.
Bjørn	27	7	9 - Brukte tilbake knapp	3:18	25	3	Så ikke tags var en ting når han søkte. Trodde Sort by var for å sortere tags nedføre.
Ida	14	11	3	2:41	19	2	La ikke merke til Sort by listen. Forsto heller ikke helt hva tags var. Trenge hint på søking oppgaven.
Linda	12	8	4	ca. 1:10. Fikk hint, gikk inne på feil prosjekt Usikker på hvilken kombinasjon av søking som skulle benyttes.	20	4	Ville ha info om hva feltene betydde på framsiden. Usikker på hvilken kombinasjon av søking som skulle benyttes. Også usikker på hva tags var

For rollen Academic:

Brukernavn: brukeraca@example.com

Passord: password

Oppgave1:

Opprett et nytt privat prosjekt.

Oppgave2:

Ditt prosjekt skal inneholde informasjon om "Birch" og "Oak". Gjør dette synlig for andre brukere.

Oppgave3:

Last opp gitte filer til prosjektet i en ny sub-folder som heter: Test.

Oppgave4:

Arne Trulsen synes ditt prosjekt virker interessant, men siden ditt prosjekt er privat kan han ikke se filene. Gi han tillatelse til å se og/eller laste ned filer fra ditt prosjekt.

Email-en hans er: fridatrulsen@teleworm.us

Deltaker	Oppgave 1 tid	Oppgave 2 tid	Oppgave 3 tid	Oppgave 4 tid	Tilbake melding
					Får felmeddelande när upprättar ett privat projekt, men det bildas ändock
Margot	5	5	8	10	Foldarna: Dicom: byt namn till Siemens Dicom dokuments:-> Documents images: Images logsLog files tiff Microtec Tiff
Niclas					Datum create new projekt fungerar inte i safari, plus det blir någon krach
Stein	47	Error i tag box	ToggleFIle krasj	Frida Trulsen er User	Gikk ganske bra egentlig, men ble stoppet av noen krasj
Bjørn	27	1:17	2:09	56	Prøvde å skrive inn Oak, birch i søk for å så trykke på edit tags. Trenge til slutt å få hint om at han bare måtte bruke edit tags. Forstod ikke hvordan han valgte sum-folder. Etter å fått hint så han ikke at upload knappen ble trykkbar Prøvde først å gjøre slik at prosjektet ikke var privat, men forstod raskt at han skulle end til special permission.
Ida	32	4	1:05	20	Upload knapp bør bli plassert under der filer havnet. Trenge hint om at sub-mappe ikke var markert.
Linda	25	10	39	1:12	Forventet at sub-folderen skulle bli auto valgt. Fikk lastet opp etter lite hint. Kunne sagt i fra til brukeren hvis de legger til fil og ikke har valgt sub-folder. Usikker på om brukeren skulle bli lagt til som medlem eller spesial tillatelse. Måtte gi hint om hvilken side.

For rollen Administrator:

Brukernavn: Brukeradmin@example.com

Passord: password

Oppgave1:

Du skal legge til en ny bruker i systemet:
Navn: Jarle Håland
Email: Jarle.Haaland@example.com
Rolle: Academic
Passord: password

Oppgave2:

Brukeren Maja Larsen har glømt sitt passord. Endre passordet hennes.
Email: majalarsen@rhyta.com
Passord: password

Oppgave3:

Slett brukeren til Jarle Håland.
Email: Jarle.Haaland@example.com

Deltaker	Oppgave 1 tid	Oppgave 2 tid	Oppgave 3 tid	Tilbake melding
Jose	30 sek	30 sek	20 sek	Man skulle behöva en varning innan man tar bort användare, är du säker på detta!
Margot	15	15	15	
Stein	1:24. Epost 'å' error	41	18	Gikk bra. Enklere sider med færre valgmuligheter gikk raskere
Bjørn	1:10	15	13	
Ida	50	24	11	
Linda	43	23	08	

D API - forespørseler

Kilde: Confluence-siden
⋮

(Vedlegg starter på neste side...)

Api-Requests

Admin:

/admin

Navn	Adresse	Mapping	Parameter type	Parameter	Respons	Respons Årsak	Beskrivelse
addUser	/newUser	Post	Body raw JSON	firstName lastName email password role	200-Ok (User) 400-Bad Request 400-Bad Request 400-Bad Request 409-Conflict	Bruker ble laget User er null Email er null eller password i kke er gyldig. Role er ikke gyldig Email finnes allerede	Et gyldig passord må minst ha 5 teign.
editUser Details	/editUser	Put	Body raw JSON	userId firstName lastName email password role	400-Bad Request 200-Ok (User) 500-Internal Server Error 404-Not Found 409-Conflict 400-Bad Request 400-Bad Request	User, role eller userId og em ail er null Bruker ble endret Brukeren ble ikke lagret inn i databasen Fant ingen bruker med userId Email finnes allerede Email er null password ikke er gyldig. Role må være gyldig eller tom	Om firstName, lastName, email, password eller role er tom vil det ikke skje noen endringer på bruker detaljen. Et gyldig passord må minst ha 5 teign.
deleteUs er	/deleteUser	Delete	Body raw JSON	userId	400-Bad Request 500-Internal Server Error 404-Not Found 200-Ok	User, userIder null eller user Ider tom Klarte ikke å fjerne brukeren fra databasen Fant ingen bruker med userId Brukeren ble fjernet fra databasen	Sletter en bruker fra databasen.
deleteTa gs	/deleteTags	Delete	Body x-www-urlencoded	tagNames (List)	400-Bad Request 500-Internal Server Error 200-Ok (List<String>)	tagNames er null Klarte ikke å fjerne tagen fra databasen Tag ble fjernet fra databasen	Sletter en tag fra databasen. Kan ta inn flere tag navn på en gang. Returnerer alle tags som ikke ble slettet som en liste.
getUser	/getUser	Get	Query Param	email	200-Ok (User) 400-Bad Request 404-Not Found	Brukeren med email ble funnet email er null eller tom Ingen bruker med email ble funnet	Må bruke Query Param!
getAllUse rs	/allUsers	Get			404-Not Found 200-Ok (List<User>)	Fant ingen brukere i databasen Fant minst en bruker i databasen	
restartSe rver	/restartServ er	Post	Body raw JSON	date time zone	200-Ok 400-Bad Request	Om planlegging ble vellykket. Om parameter ikke kunne bli omgjort til en dato eller datoen er før dagens dato/tid.	date format: yyyy-MM-dd time format: HH:mm zone format: +0200

Academic:

/academic

Navn	Adresse	Mapping	Parameter type	Parameter	Respons	Respons Årsak	Beskrivelse

createTag	/createTag	Post	Body x-www-urlencoded	tagName	400-Bad Request 409-Conflict 200-Ok (Tag)	tagName er null eller tom Tag finnes allerede Tag ble laget	
addTag	/addTag	Put	Body x-www-urlencoded	tagNames (List) projectId	200-Ok (Project) 400-Bad Request 404-Not Found 403-Forbidden 500-Internal Server Error 409-Conflict 400-Bad Request	Tags ble lagt til i prosjekt tagNames eller projectId er null Tags eller prosjekt ble ikke funnet Brukeren har ikke lov å gjøre endringer på prosjektet Tags ble ikke lagt til i databasen En eller flere Tags finnes allerede i prosjektet tagNames kan ikke ha mindre enn 2 tegn	Legger til 1 eller flere tags til et prosjekt
removeTag	/removeTag	Put	Body x-www-urlencoded	tagNames (List) projectId	200-Ok (Project) 400-Bad Request 404-Not Found 403-Forbidden 500-Internal Server Error 400-Bad Request	Tags ble fjernet fra prosjektet tagNames eller projectId er null En eller flere tags eller prosjektet ble ikke funnet Brukeren har ikke lov å gjøre endringer på prosjektet Klarte ikke å fjerne tags fra prosjektet tagNames kan ikke ha mindre enn 2 tegn	Fjerner 1 eller flere tags fra et prosjekt.
createProject	/createProject	Post	Body raw JSON	projectName isPrivate creation description	200-Ok (Project) 400-Bad request 409-Conflict	Prosjekt ble opprettet ProjectDto eller user er null Et prosjekt med dette navnet eksisterer allerede	
deleteProject	/deleteProject	Delete	Body raw JSON	projectId	200-Ok 400-Bad request	Prosjekt ble slettet ProjectDto eller user er null	
changeProjectOwner	/changeProjectOwner	Put	Body raw JSON	projectId userId	200-Ok 400-Bad request 404-Not found	Eieren ble endret ProjectDto er null projectId eller userId eksisterer ikke i databasen	
addMemberToProject	/addMemberToProject	Put	Body raw JSON	projectId userEmail	200-Ok 400-Bad request 403-Forbidden 404-Not found 500-Internal Server Error	Nytt medlem ble lagt til ProjectDto er null eller om brukeren med userEmail er eieren av prosjektet eller medlem. Bruker har ikke lov til å endreprosjekter Prosjekt eller bruker finnes ikke En annen Exeption ble kastet eller metoden klarte ikke legge til nytt medlem	
removeMemberFromProject	/removeMemberFromProject	Put	Body raw JSON	projectId userEmail	200-Ok 400-Bad request 403-Forbidden 404-Not found 500-Internal Server Error	Et medlem ble fjernet fra prosjektet ProjectDto er null Bruker har ikke lov til å endreprosjekter Prosjekt eller bruker finnes ikke En annen Exeption ble kastet eller metoden klarte ikke fjerne medlemmet	

grantSpecialPermission	/grantSpecialPermission	Put	Body x-www-urlencoded	projectId userEmail	200-Ok 400-Bad request 403-Forbidden 404-Not found 409-Conflict 500-Internal Server Error	En bruker ble gitt special permission projectId eller userEmail er null eller tomme Bruker har ikke lov til å endere prosjekter Ingen prosjekter eller brukere har disse projectId eller userEmail Bruker har allerede special permission Noe gikk galt med å gi brukeren special permission	
revokeSpecialPermission	/revokeSpecialPermission	Put	Body x-www-urlencoded	projectId userEmail	200-Ok 400-Bad request 403-Forbidden 404-Not found 409-Conflict 500-Internal Server Error	En bruker fjernet fra special permission projectId eller userEmail er null eller tomme Bruker har ikke lov til å endere prosjekter Ingen prosjekter eller brukere har disse projectId eller userEmail Bruker har ikke special permission i utgangspunktet Noe gikk galt med å fjerne special permission	
uploadFiles	/uploadFiles	Post	form-data	files (List) projectId subFolder	200-Ok 400-Bad Request 404-Not Found 403-Forbidden	Filer ble lastet opp. Det blir også sendt med en liste med alle filer som ikke ble lastet opp. Om subFolder er null eller tom. Om prosjekt med projectId ikke finnes. Brukeren har ikke tillatelse til å laste opp filer til dette prosjektet.	Om subFolder ikke eksisterer vil en mappe med navnet bli opprettet.
addTagsToFile	/addFileTag	Put	Body x-www-urlencoded	tagNames (List) projectId subFolder fileName	200-Ok 204-No Content 400-Bad Request 400-Bad Request 403-Forbidden 409-Conflict	Tags med tagNames ble lagt til på filen med navn fileName i databasen. Alt gjekk bra, men det finnes ingen fil med navn fileName i undermappe subFolder i prosjekt med projectId. Liste med tagNames, projectId, subFolder eller subFolder er tom. tagNames har 2 eller mindre bokstaver i navnet. Brukeren har ikke lov å gjøre endringer på prosjektet En eller flere Tags finnes allerede på filen med fileName.	If file with fileName does not already exist in the database in subFolder associated with projectId a new instance of that object will be created and added.
removeTagFromFile	/removeTagFromFile	Put	Body x-www-urlencoded	tagNames (List) projectId subFolder fileName	200-Ok 204-No Content 400-Bad Request 400-Bad Request 403-Forbidden	Tags med tagNames ble fjernet fra databasen. Alt gjekk bra, men en eller flere filer finnes ikke i databasen. Liste med tagNames, projectId, subFolder eller subFolder er tom eller null. tagNames har 2 eller mindre bokstaver i navnet. Brukeren har ikke lov å gjøre endringer på prosjektet	
setProjectPrivacy	/setProjectPrivacy	Put	Body x-www-urlencoded	projectId privacy (boolean)	200-Ok 204-No Content 400-Bad Request 403-Forbidden	Om privacy ble satt og alt gjekk bra. Alt gjekk bra, men det finnes ingen prosjekt med id projectId. projectId eller privacy er null. Brukeren har ikke lov å gjøre endringer på prosjektet	

setProjectDescription	/setProjectDescription	Put	Body x-www-urlencoded	projectId description	200-Ok 204-No Content 400-Bad Request 403-Forbidden	Om description ble satt og alt gjekk bra. Alt gjekk bra, men det finnes ingen prosjekt med id projectId. projectId eller description er null. Brukeren har ikke lov å gjøre endringer på prosjektet	
getMyProjects	/getMyProjects	Get			200-Ok 204-No Content	Om serveren fant noen prosjekt som brukeren eier eller er medlem av. Om serveren ikke fant noen prosjekt som brukeren eier eller er medlem av.	

User:

/user

Navn	Adresse	Mapping	Parameter type	Parameter	Respons	Respons Årsak	Beskrivelse
getAllProjectsTagIsUsedIn	/getAllProjectsTagIsUsedIn	Get	Query Param	tagName	200-Ok (List<Project>) 400-Bad Request 404-Not Found 400-Bad Request	Om tag med tagName ble funnet Om tagName null eller tom Om ingen tag med tagName ble funnet Om tagName har mindre enn 2 tegn	
currentUser	/currentUser	Get			200-Ok (User) 401-Unauthorized	Brukeren er innlogget Fant ingen bruker	Brukes for å hente den innlogget brukeren.
getAllProjects	/getAllProjects	Get			200-Ok (List<Project>) 404-Not Found	Har funnet alle prosjekter Det finnes ingen prosjekter	
getProject	/getProject	Get		projectId	200-Ok (Project) 404-Not Found 500-Internal Server Error	Prosjektet eksisterer og ble returnert Finne ikek et prosjekt med denne projectId Fikk ikke til å returnere prosjektet eller annen Exception	
getAllTags	/getAllTags	Get			200-Ok (List<Tag>) 404-Not Found	Om minst en tag ble funnet Om det ikke finnes noen tagger i databasen	
downloadFile	/downloadFIle	Post	Body x-www-urlencoded	fileName (List<String>) projectId subFolder	200-Ok (File content) 400-Forbidden 404-Not Found 403-Forbidden 410-Gone	Om filene ble funnet. Om fileName ikke inneholder filtype eller subFolder string er tom. Om prosjekt med projectId ikke finnes. Om brukeren ikke har tillatelse til å hente filer fra dette prosjektet. Om fil med fileName ikke ble funnet.	Om 2 eller flere filer blir sent i requesten vil filene bli lastet ned som ".zip". Om bare 1 fil blir sent i requesten blir filen lastet ned som samme filtype som filen er.
getAllFileNames	/getAllFileNames	Get	Query Param Query Param Query Param	directory projectId subFolder	200-Ok (FileOTD object with file name and tags (List<Tag>)) 400-Bad Request 400-Bad Request 404-Not Found 403-Forbidden 410-Gone	Om mappen ble funnet. Om directory ikke er en gyldig mappe. Om subFolder er null eller tom. Om prosjekt med projectId ikke finnes. Om brukeren ikke har tillatelse til å se filer fra dette prosjektet. Om mappen ikke finnes.	Gyldige directory parameter er: documents, images, logs, dicom, tiff og all. Denne er ikke case sensetive.

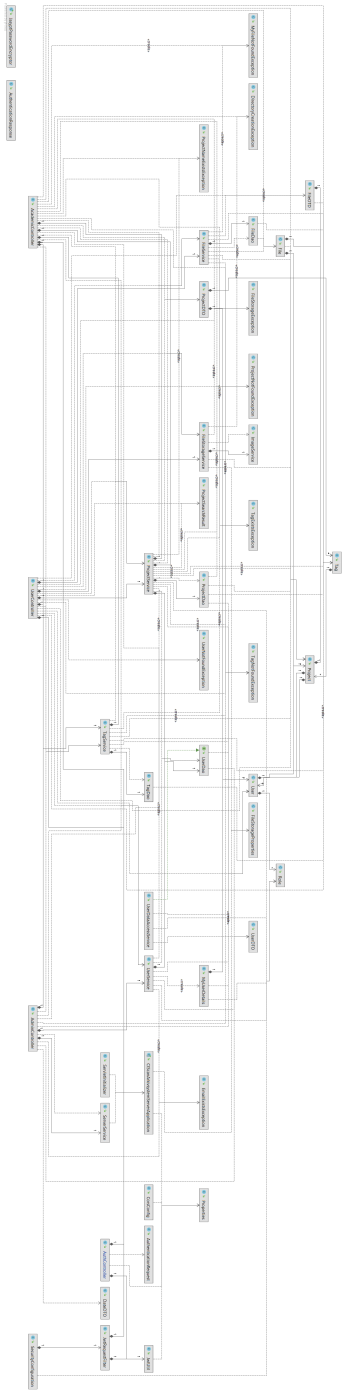
getAllProject SubFolders	/getAllProject SubFolders	Get	Query Param	projectId	200-Ok (List<String) 204-No Content 403-Forbidden 404-Not Found	Om prosjektet ble funnet med sub-prosjekt. Om prosjekt mappen ikke ble funnet. Om brukeren ikke har lov til å se prosjekt filene. Om prosjektet ikke ble funnet.	
searchForPr oject	/search	Get	Query Param	search tagFilter (List)	200-Ok (List<ProjectSearchResult>) 204-No Centent 400-Bad Request	Om søket ble fullført. Om ingenting ble funnet vil det bli returnert en tom liste. Om det ikke ble funnet noen prosjekt i databasen. Om tagFilter ikke er med.	tagFilter kan være tom. Om ingen tagFilter er med vil systemet søke igjennom alle prosjekt. Om tagFilter ikke er tom vil bare prosjekt som har alle tags som tagFilter har bli returnert.
getImage	/getImage	Post	Body x-www-urlencoded	imageName projectId subFolder size	200-Ok (Image) 400-Bad Request 403-Forbidden 404-Not Found 410-Gone	Om bildet ble funnet og bilde formatet støttes. imageName inneholder ikke file type. Om den brukeren ikke har tillatelse til å se på filene. Fant ingen prosjekt med projectId. Om bildet ikke ble funnet.	Om size er 0 vil bildet bli returnert i original størrelse. size fungerer ikke på bilder av type: gif

Other:

/auth

Navn	Adresse	Mapping	Parameter type	Parameter	Respons	Respons Årsak	Beskrivelse
login	/login	Post	Body raw JSON	userName password	200-Ok (Header<jwt token>) 401-Unauthorized	Innlogging godkjent Feil userName eller password	
logout	/logout	Get			200-Ok (Header)	Logout godkjent	

E Backend - klassediagram



Figur 80: Backend klassediagram.

F Forprosjektrapport

⋮

(Vedlegg starter på neste side...)

FORPROSJEKT - RAPPORT

FOR BACHELOROPPGAVE

NTNU
Kunnskap for en bedre verden

TITTEL:
Arkivsystem for CT-scan

KANDIDATNUMMER(E):
Trym Vaaland, Aleksander Bakken, Brage Tranvik

DATO:	EMNEKODE: *	EMNE:	DOKUMENT TILGANG:
29.01.2021	**IE303612**	**Bacheloroppgave (Data)**	- Åpen

STUDIUM:	ANT SIDER/VEDLEGG:	BIBL. NR:
BACHELOR I INGENIØRFAG -DATA	1/10	- Ikke i bruk -

OPPDRAGSGIVER(E)/VEILEDER(E):
Oppdragsgiver: Lars Hansson Veileder: Arne Styve

OPPGAVE/SAMMENDRAG:
Dette prosjektet er bacheloroppgaven til Trym, Brage og Aleksander for dataingeniør-utdanningen. Prosjektet går ut på å utvikle et web-basert arkivsystem som skal kunne håndtere informasjon og bilder av trestokker. Dette blir gjort i form av en nettside der man skal kunne logge inn for å legge inn ny informasjon eller søke opp gammel informasjon ved hjelp av tagger.

Postadresse
NTNU i Ålesund
Postboks 1517
N-6025 Ålesund

Besøksadresse
Larsgårdsvegen 2
Internett
www.ntnu.no

Telefon
70 16 12 00
Epostadresse
postmottak@ntnu.no

Telefax
70 16 13 00

Bankkonto
7694 05 00636
Foretaksregisteret
NO 947 767 880

INNHOLD

1 INNLEDNING 3
2 BEGREPER 3
3 PROSJEKTORGANISASJON 3
3.1 PROSJEKTGRUPPE 3
3.2 STYRINGSGRUPPE (VEILEDER OG KONTAKTPERSON OPPDRAGSGIVER) 4
4 AVTALER 4
4.1 AVTALE MED OPPDRAGSGIVER 4
4.2 ARBEIDSSTED OG RESSURSER 4
4.3 GRUPPENORMER – SAMARBEIDSREGLER – HOLDNINGER 4
5 PROSJEKTBESKRIVELSE 4
5.1 PROBLEMSTILLING - MÅLSETTING - HENSIKT 4
5.2 KRAV TIL LØSNING ELLER PROSJEKTRESULTAT – SPESIFIKASJON 4
5.3 PLANLAGT FRAMGANGSMÅTE(R) FOR UTVIKLINGSARBEIDET – METODE(R) 4
5.4 INFORMASJONSINNSAMLING – UTFØRT OG PLANLAGT 5
5.5 VURDERING – ANALYSE AV RISIKO 5
5.6 HOVEDAKTIVITETER I VIDERE ARBEID 5
5.7 FRAMDRIFTSPLAN – STYRING AV PROSJEKTET 5
5.8 BESLUTNINGER – BESLUTNINGSPROSESS 6
6 DOKUMENTASJON 6
6.1 RAPPORTER OG TEKNISKE DOKUMENTER 6
7 PLANLAGTE MØTER OG RAPPORTER 6
7.1 MØTER 6
7.2 PERIODISKE RAPPORTER 6
8 PLANLAGT AVVIKSBEHANDLING 6
9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING 7
10 REFERANSER 7
VEDLEGG 7

1 INNLEDNING

Denne oppgaven ble valgt fordi den var interessant og ganske unik hvor det blir tatt i bruk interessant teknologi. Gruppen har erfaring innen webutvikling og databaser, slik at oppgaven ikke blir alt for fremmed samtidig som at den blir litt utfordrende.

Oppdragsgiveren er Luleå tekniska Universitet i Skellefteå. De har en ledende plassering innen stokkskanning, tørking, varmebehandling, fuktdynamikk og mer. Dette er gjort ved hjelp av en **CT-skanner** og et klimakammer.

Hvert forskningsprosjekt der tomografen benyttes, genererer en stor mengde informasjon i form av data og et stort antall bilder. All denne informasjonen håndteres svært manuelt, i form av filer på en harddisk/filserver med liten grad av søkemuligheter og bearbeiding/uthenting av informasjon.

Formålet med oppgaven er å utvikle et web-basert arkivsystem som skal brukes til å gjøre ulike typer søk i alt datamateriale som er samlet inn under skanning. Informasjonen og bildene er primært organisert etter prosjekter, men ved hjelp av **metadata** skal brukere være i stand til å kunne organisere data og bilder slik de vil ha det. Brukeren skal selv kunne definere nye tagger og kategorier osv.

2 BEGREPER

CT-skanner: En CT-skanner eller Computertomografi er en maskin som ved hjelp av radiologisk undersøkelse som tar snittbilder som blir behandlet digitalt vil vi kunne framstille ett bilde av innsiden til det som ble tatt bilder av. Ofte brukt i medisin, men kan også brukes til å kunne se innsiden av for eksempel tømmer.

Metadata: Metadata er data som beskriver en annen data. For eksempel så kan metadataen til ett bilde si når bildet ble tatt, eller hvor det ble tatt.

3 PROSJEKTORGANISASJON

3.1 Prosjektgruppe

Studentnummer(e)
476078
507876
507875

3.1.1 Oppgaver for prosjektgruppen - organisering

Vi planlegger ikke å ha noen fast lederrolle, men heller jobber på prosjektet som utviklere. Sekretær rollen gir vi ut til en på gruppen før ett møte slik at det alltid er en som skriver notater under møtet og vi kan variere på hvem som skriver.

3.1.2 Oppgaver for øvrige medlem(mer)

Trym, Aleksander og Brage har alle ansvar for utvikling av arkivsystemet for CT-Scan filene der vi skal utvikle en nettbasert løsning med en server. Arbeidsoppgavene går ut på planlegging, utvikling, testing og dokumentering av prosjektet.

3.2 Styringsgruppe (veileder og kontaktperson oppdragsgiver)

Arne Styve er veileder for gruppen og Lars Hansson er Oppdragsgiver. Hvis vi har tekniske eller praktiske spørsmål angående prosjektet så burde vi kontakte Arne Styve for hjelp. Hvis vi har spørsmål angående prosjektet og prosjektets sluttprodukt så skal vi henvise oss til Lars Hansson og høre hva han mener.

4 AVTALER

4.1 Avtale med oppdragsgiver

4.2 Arbeidssted og ressurser

Arbeidsplass:

- NTNU Ålesund datalab
- Hjemme

Ressurser:

- Egne datamaskiner
- JetBrains IntelliJ
- Postman (Program for testing av server)
- Jira og Confluence
- GitKraken
- GitHub

Tilgang til personer:

- Arne Styve (Veileder)
- Lars Hansson (Oppdragsgiver)

Avtalt rapportering:

- I slutten av hver sprint:
  - Sprint retrospective
  - Sprint review-rapport
- Forprosjektsrapport: 29. januar
- Bachelor rapport: 20. mai

4.3 Gruppenormer – samarbeidsregler – holdninger

Noen normer som vi har, er å samarbeide og hjelpe hverandre hvis mulig. Hvis noen har ett spørsmål eller trenger hjelp til noe så prøver vi å løse problemer som en gruppe. Vi prøver å ha god kommunikasjon og si ifra hvis noen ting ikke er mulige å fullføre, hvis noen tidspunkt ikke passer, gi beskjed i god tid hvis det er noe resten av gruppen trenger å vite osv.

Vi prøver også å møte opp til riktig tidspunkt og være presise når vi skal begynne å jobbe klokken 10:15. Da skal vi jobbe fram til 16:00 så lenge vi føler vi har noe vi kan gjøre. De dagene vi føler vi burde jobbe 8 timer så kan vi avtale på forhånd om vi vil starte 08:15 eller slutte 18:00 for å få en lengre arbeidsdag.

5 PROSJEKTBESKRIVELSE

5.1 Problemstilling - målsetting - hensikt

Hovedmål:

I oppgavebeskrivelsen beskrives målet med prosjektet slik:
«Målet med dette prosjektet er å utvikle et web-basert arkivsystem som gjør det enkelt for brukere å gjøre ulike typer søk i alt datamateriale som er samlet inn under skanning.» Det skal være mulig å registrere nye CT-scans og redigere eksisterende informasjon som ligger i systemet. Gjennom løsningen vil en bruker kunne søke på diverse metadata og tagger for å finne informasjonen de søker etter.

Delmål:

Vårt første delmål blir å utvikle en backend med en server løsningen.

Deretter blir det neste delmålet å utvikle en frontend med en webbasert løsning.

Effektmål:

Effektmålet er å gjøre det enklere og kjappere for de som jobber med bilder og informasjon som er skaffet gjennom CT-skanning av tre.

Resultatmål:

Resultatmålet i dette prosjektet er at vi har en webserver som kjører på en Docker container. Denne serveren skal kunne lese filer som ligger på maskinen som Docker-containeren kjører på. Det skal også være en nettside som frontend der brukere skal kunne logge inn for å legge inn nye prosjekter eller søke opp gamle prosjekter. Om en bruker kan legge inn nye prosjekter eller kan lese visse prosjekter vil variere etter hvilken rolle en bruker har.

Prosessmål:
Vi skal gjennomføre prosjektet ved å bruke agile arbeidsmetode, her Scrum. Målet er at vi skal lære å planlegge og gjennomføre et større prosjekt. Vi skal ha god kommunikasjon med arbeidsgiver og holde han oppdatert under arbeidet.

5.2 Krav til løsning eller prosjektresultat – spesifikasjon

I dette prosjektet er det forventet at vi skal forbedre et arkivsystem ved å lage en web-basert løsning der man skal kunne søke etter prosjekter ved hjelp av tagger. Man skal også måtte være innlogget for å bruke systemet der man kan ha forskjellige roller. Rollene kan være student, professor eller administrator. Som en student skal man kun se visse prosjekter. Som professor skal man også, kunne legge til nye prosjekter.

Vår hoved oppgave vil bli å lage en prototype som noen andre skal kunne overta når vi er ferdige. Siden noen andre skal ta over prosjektet så er det viktig at vi kommenterer koden bra og lager en guide på hvordan man setter opp serveren og eventuelt andre deler av prosjektet der man kan trenge det.

5.3 Planlagt framgangsmåte(r) for utviklingsarbeidet – metode(r)

Framgangsmåten vår baserer seg på å jobbe med Scrum metoden. Vi har valgt å jobbe i 5-dagers sprinter med sprint review, sprint retrospective og sprint planning på fredag.

Styrker med Scrum er effektiv tidsbruk, det deler opp prosjektet i mindre deler og gjør det enklere å implementere tilbakemeldinger.
Svakheter kan være at det er viktig å følge rammeverket ordentlig for å unngå problemer, det kan bli for mange møter og at det er avhengig av jevnt arbeid fra alle.

5.4 Informasjonsinnsamling – utført og planlagt

Etter litt research og tips fra veileder har vi funnet noen lignende løsninger som kan være smart å hente inspirasjon fra. Blant annet så kan vi se mot Adobe Lightroom og Google Photos. Det virker som Adobe Lightroom er en av de mest dominerende programmene innenfor bilde organisering, men det trengs et abonnement å bruke. Google Photos er gratis for alle og er ekstremt brukervennlig, men kan ikke gjøre like mye som Lightroom.

Når vi kommer inn i arbeid med prosjektet så vil vi se mer nøye på lignende løsninger for å trekke inspirasjon om hvilke verktøy vi burde inkludere og hvordan vi burde implementere de. Vi kommer til å trekke mest verktøy fra apper som ligner på Adobe Lightroom, og mest brukergrensesnitt fra apper som ligner Google Photos.

5.5 Vurdering – analyse av risiko

Siden gruppen har liten erfaring med store prosjekt og noen av de tekniske temaene vi må innenfor så er det vanskelig å si hvor lang tid prosjektet kommer til å ta. Vi er optimistiske og håper vi blir ferdige innenfor tidsrammen vår. På grunn av liten erfaring innenfor områdene er vi heller ikke sikre på hvor vi kunne sette ytterligere avgrensninger.

Potensiell løsning: Vi må være bevisst på denne trusselen når vi begynner på nye deler av prosjektet slik at vi ikke ender opp med å ta på oss for mye jobb som vi ikke klarer eller ikke rekker.

Prosjektet har en veldig fast tidsramme, som betyr at sløsing av tid kan ende opp med at vi ikke blir ferdige med prosjektet eller at vi blir nødt til å kutte ned på størrelsen av prosjektet. Derfor er det en reel trussel om at vi ikke

får tatt i bruk de forskjellige teknologiene raskt nok og effektivt nok, ettersom det kan føre til at prosjektet ikke kommer like langt som vi hadde tenkt. Et konkret eksempel på dette kan være at vi ikke klarer å sette opp en bra nok frontend på grunn av at vi har lite erfaring, og derfor ikke vet hvilke teknologier vi burde ta i bruk. Det samme gjelder for backend, men vi har litt erfaring med servere og tror derfor at det ikke er like sannsynlig at det blir et problem.

Potensiell løsning: Vi må planlegge godt for å velge de beste rammeverkene til å løyse oppgaven som også er enkle å bruke. Om vi innser vi ikke har tid til å bli ferdige må vi ha et møte og planlegge hva som er viktigst for å gi et best mulig resultat med den tiden vi har.

Prosjektet skal bli brukt videre av kunden, så derfor er det viktig at prosjektet er enkelt å sette opp for dem. Her er det en risiko med at vi vil bruke Docker til å kjøre serveren. En risiko med dette blir at vi ikke får til å sette opp Docker og vi ikke får den til å fungere.

Potensiell løsning: Vi bør være nøye når vi lærer oss docker slik at vi ikke gjør noe feil og prosjektet blir enkelt å ta over senere av noen andre.

En annen risiko er at prosjektet tar lengre tid enn forventet og/eller prosjektet blir for stort. En annen risikofaktor kan være dårlig planlegging. Hvis vi ikke har lagt en fast plan kan det føre til lite effektiv jobbing og bortkastet tid.

Potensiell løsning: For å gjøre det lettere å kontrollere tidsbruken vår må vi planlegge sprintene ordentlig så vi vet hva som er målet med arbeidet hver uke. Om vi ser at tid blir et problem må vi også planlegge i god tid for å finne ut hva som er viktigst for å få best mulig resultat som vi kan på den tiden vi har.

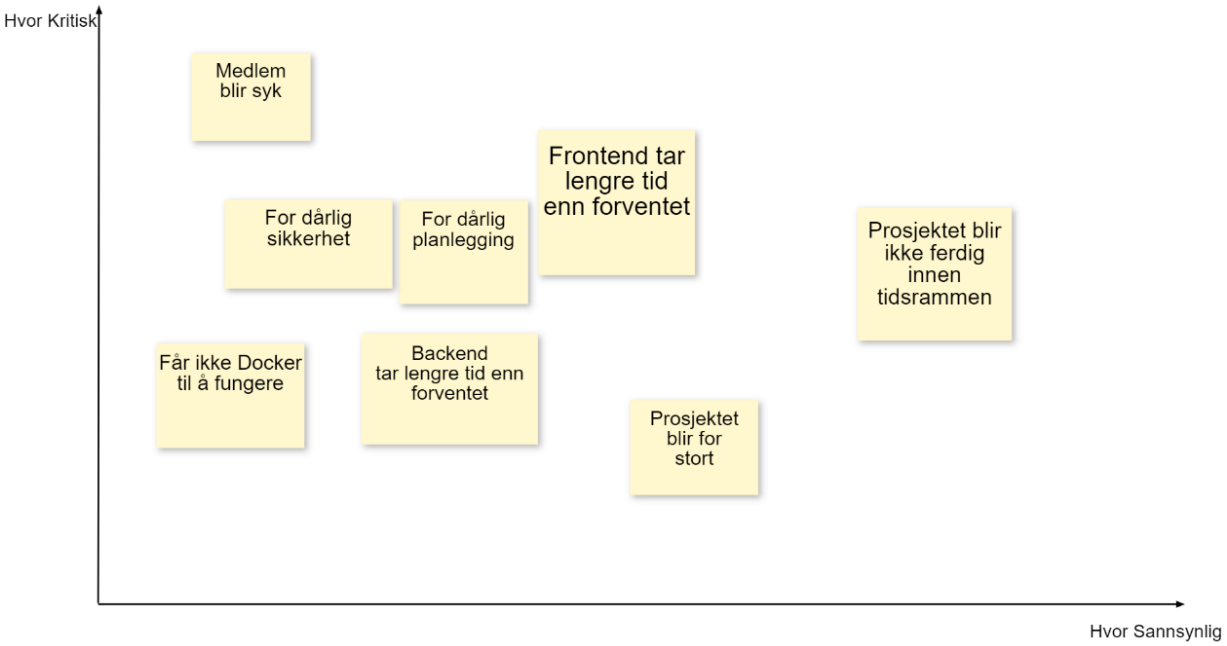
Mulige risikoer i prosjektet kan være sniffing på nettforespørslene.

Potensiell løsning: Dette kan bli løst med å sikre serveren med HTTPS kryptering ved hjelp av for eksempel Buypass (ACME) som er et gratis HTTPS krypterings sertifikat. For å sikre brukarene må vi sannsynligvis sikre dem ved å lagre de i en database med krypterte passord.

Hvis et medlem av gruppa blir syk og ikke får til å jobbe videre, kan det ha stor effekt på prosjektet. Hvis vi har fordelt arbeidsoppgaver jevnt mellom medlemmene må de flyttes over på de andre. Da kan det bli mer arbeid på hver person enn det de rekker. Det kan hende at dette forsinker prosjektet så det ikke blir ferdig som planlagt innen tidsrammen som er satt.

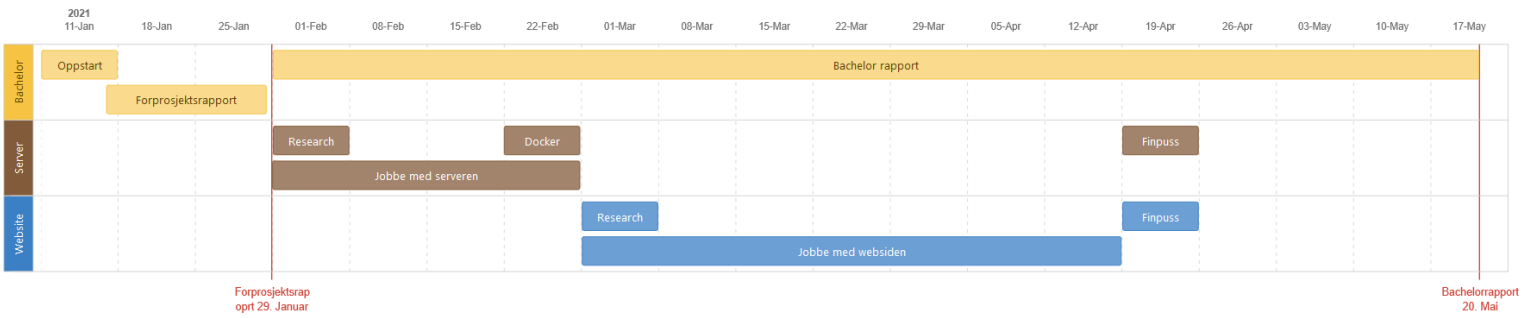
Potensiell løsning: Etter hvor alvorlig det er må vi kutte ned på prosjektet slik de resterende medlemmene ennå kan komme med et fungerende resultat. For å minke sjansen må vi prøve å ikke dra ut på unødvendige turer der vi kan havne i risiko for å få sjukdom som Covid-19.

Tabell 1: Risikotabell



5.6 Hovedaktiviteter i videre arbeid

Nr	Hovedaktivitet	Ansvar	Kostnad	Tid/omfang
A11	Skrive Bachelor rapport	Alle studenter	0.0kr	02.01 – 05.20
A21	Jobbe med Backend	Alle studenter	0.0kr	02.01 – 02.28
A22	Jobbe med Docker	Ukjent	0.0kr	02.22 – 02.28
A31	Jobbe med Frontend	Alle studenter	0.0kr	03.01 – 03.18



5.7 Framdriftsplan – styring av prosjektet

5.7.1 Hovedplan

Fra 12 til 29 januar skal vi skrive forprosjekt rapporten, skrive kravspekket, jobbe med confluence og planlegge litt om hva rammeverk og teknologi vi skal bruke.

Fra 1 til 27 februar skal vi fokusere på backenden av prosjektet der den siste uken skal vi prøve å få satt opp et docker image med serveren.

Fra 1 mars til 16 april skal vi fokusere på frontend som er nettsiden. Den siste tiden i april er satt opp for å finpusse og fikse på prosjektet.

Vi har planlagt å jobbe med selve bachelor rapporten gjennom hele perioden og fokusere mest på den fra slutten av april til leverings fristen den 20 mai.

Milepæler:

- 29. januar – Innleveringsfrist forprosjektrapport
- 27.februar – Være ferdig med det meste av backend
- 27.februar – Satt opp docker image med serveren
- 16.april – Være ferdig med det meste av frontend
- 20. mai – Innleveringsfrist for hovedrapporten

5.7.2 Styringshjelpemidler

Vi kommer til å bruke JIRA og Confluence som hoved hjelpemidlene våre gjennom dette prosjektet. Gjennom JIRA vil vi kunne starte og avslutte sprints, lage stories og tasks for å holde styr på oppgaver, hvem som er tildelt oppgavene og hva statusen til oppgavene er. JIRA kommer til å være plattformen vi bruker for arbeidsfordeling og ansvar. Vi kan også skrive rapporter inne på JIRA.

På JIRA vil vi holde styr på planleggingen av prosjektet når det kommer til aktiviteter, tid og milepæler. Confluence fungerer bra som en plass der vi kan notere og vise fram notater og rapporter.

Vi vil også ta i bruk GitHub til å holde på prosjektet for å holde styr på versjoner og enkelt kunne lage nye greiner for hver issue vi jobber med. Til å lage nye greiner vil vi ta i brukt GitFlow. Til versjonskontroll verktøy skal vi bruke GitKraken som vi alle har god erfaring med etter mye tidligere bruk.

5.7.3 Utviklingshjelpemidler

- Backend verktøy:
    - Spring Boot som rammeverk
    - Postman for api testing
- Frontend verktøy:
    - React, Angular eller Vue som rammeverk
- Server verktøy:
    - Docker til å kjøre serveren på
    - Tomcat eller Payara Micro til å kjøre serveren
- Andre verktøy:
    - IntelliJ som text editor
    - GitKraken / GitHub til versjonskontroll

5.7.4 Intern kontroll – evaluering

Kontroll og oppfølging av framdrift vil bli gjort ved hjelp av JIRA og Confluence gjennom sprint og task systemet som gjør det lett å se hvem som har gjort hva og når det ble gjort. Vi skal også ha ett sprint review, retrospective og planning møte i slutten av hver sprint. Hver andre sprint vil vi ha et sprint review-møte med veileder og oppdragsgiver. Versjonskontroll av selve prosjektet vil bli gjort gjennom Git.

Ett kjennetegn på at ett mål/delmål er nådd er når vi har gjort mesteparten av arbeidet på en del av prosjektet og vi beveger oss til å jobbe på en ny del. Praktisk for dette prosjektet så kommer disse til å være når vi blir ferdige hovedarbeidet med serveren og hovedarbeidet med websiden.

5.8 Beslutninger – beslutningsprosess

Beslutninger om avgrensning ble tatt sammen i gruppa, basert på diskusjon i gruppa og fra informasjon om oppgaven vi fikk fra arbeidsgiver.

Beslutninger vil bli avgjort som en gruppe der alle medlemmene vil være å bestemme. Om vi er usikre vil vi høre med kontaktpersonen vår. Beslutninger om selve oppgaven vil derimot bli diskutert med arbeidsgiveren.

6 DOKUMENTASJON

6.1 Rapporter og tekniske dokumenter

Dokumentasjon til kunden:

- Forklaring på hvordan prosjektet er satt opp og løsningens forretningslogikk
- Dokumentert kode
- Docker image med serveren
  - Forklaring på hvordan docker imaget blir satt opp

Distribusjon

Vi vil levere det ferdige produktet som et Docker image som kunden enkelt kan sette opp og bruke. Dokumentasjonen vi lager skal forklare alt som IT-avdelingen trenger for å få den i gang.

Vedlikehold

Dokumentasjonen skal være detaljert og omfattende slik at IT-avdelingen skal kunne vedlikeholde og videreutvikle produktet.

7 PLANLAGTE MØTER OG RAPPORTER

7.1 Møter

7.1.1 Møter med styringsgruppen

Hver 2. fredag, fra og med 22. Januar så vil det være ett sprint review møte blant studentene, veileder og oppdragsgiver (Hvis Lars har mulighet). Til dette bruker vi altså Scrum rammeverket. Møtet tar plass fra 09:00 til 09:30.

7.1.2 Prosjektmøter

Prosjektet kjøres i form av Scrum med sprinter som varer 1 uke.

Vi planlegger å jobbe på mandag, tirsdag og fredag, i tillegg til når det passer på onsdag og torsdag. Onsdag og torsdag vil derimot bli vanlige arbeidsdager etter faget Ingeniørfaglig systemteknikk og systemutvikling er ferdig. Vi har også planer om å sette av en dag i uken på å jobbe med selve bacheloroppgaven. Planen er å jobbe 6 timer hver dag, fra 10:15 til 16:00 så lenge vi har noe å jobbe med. De dagene vi føler vi burde jobbe 8 timer så kan vi avtale på forhånd om vi vil starte 08:15 eller slutte 18:00 for å få en lengre arbeidsdag.

7.2 Periodiske rapporter

Etter hver sprint skal vi skrive to rapporter: Sprint review rapport og sprint retrospective. Sprint review rapporten vil være møtereferatet fra sprint review. Vi vil benytte Confluence for å lage disse. Rapportene skal forklare kort hva vi har gjort denne uken og drøfte hvordan det gikk med tanke på planen.. Dette er for å lett kunne holde oss oppdatert på hva andre på gruppen har gjort.

7.2.1 Framdriftsrapporter (inkl. milepæl)

Vår plan er nå i februar skal vi bruke til planlegging av hvordan nettsiden skal bli lagt og bli ferdig med det meste av serveren og få satt opp docker med serveren. Dette skal bli gjort ved hjelp av en docker dokument til å automatisk sette den opp.

Mars og april skal bli brukt for det meste for å sette opp nettsiden og funksjonaliteten til den som for eksempel tag søking.

Mai skal bli brukt for siste innspurt for å teste og fullføring av prosjektet.

Rapport datoer:

- 29. januar skal forprosjektrapporten leveres.
- 20. mai skal hovedrapporten leveres

8 PLANLAGT AVVIKSBEHANDLING

Dersom prosjektet ikke går som planlagt så skal vi først prøve å løse problemet innad i gruppen. Om problemet vedvarer så skal gruppen ta kontakt med veilederen og gå videre med problemet derfra.

Hvis gruppen føler at noe må endres så skal dette diskuteres innad i gruppen. Ved uenigheter eller ingen løsning blir funnet fram så skal veileder kontaktes. Hele gruppen har ansvar for å finne løsninger og kontakte veileder om nødvendig.

9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

I dette prosjektet av vi bare planer om å bruke ressurser som vi allerede har tilgang til via skolelisenser. Dvs. alle produktene inkludert i Office 365. Produktene vi har planer om å bruke med skolelisenser er IntelliJ og GitKraken. Produkter vi har tenkt å bruke med gratiskonto er Postman.

Vi har derimot ingen planer om å kjøpe noen produkt eller lisenser.

10 REFERANSER

VEDLEGG

CT-Wood Arkivsystem

NTNU

Kunnskap for en bedre verden