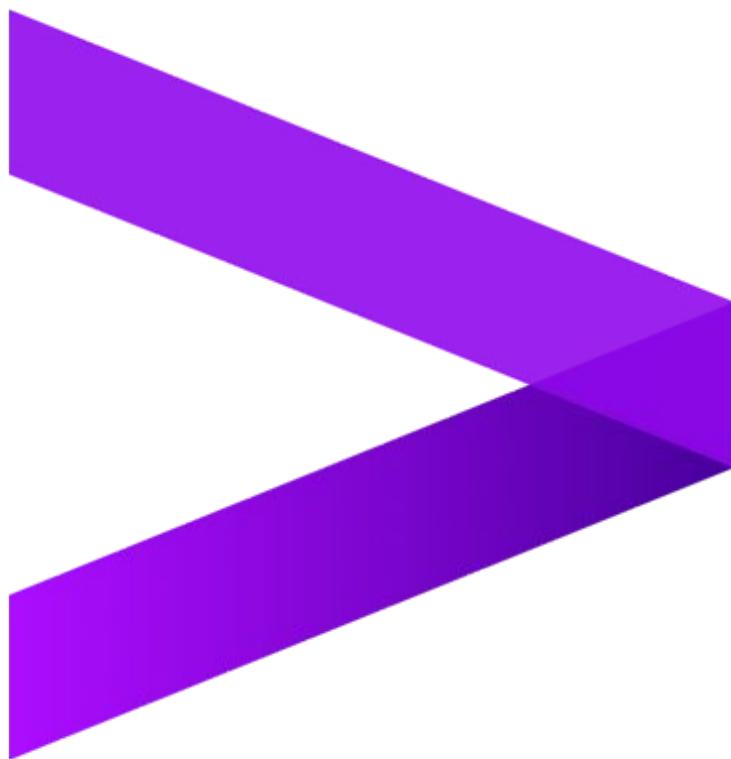


Prosjektrapport

Accenture - Webapplikasjon for spillgruppen

Bacheloroppgave
Gruppe 37

25. Mai 2020



Gruppemedlemmer:

Rany Tarek Bouorm
Eirik Bøyum
Markus Hellestveit

Veileder fra OsloMet

Qien Meng

Veiledere fra Accenture

Adam Aaskali

Mira Lilleholt Vik

BACHELORPROSJEKT

HOVEDPROSJEKTETS TITTEL	DATO
Accenture - Webapplikasjon for spillgruppen	25/05/2020
PROSJEKTDELTAKERE	ANTALL SIDER / BILAG
Rany Tarek Bouorm Eirik Bøyum Markus Hellestveit	275 inkl. Forside og tittelseite

OPPDAGSGIVER	KONTAKTPERSON
Accenture AS	Daniel Meinecke daniel.meinecke@accenture.com
	Kjell Olav Dale kjell.olav.dale@accenture.com
	Marius Torsrud marius.torsrud@accenture.com

SAMMENDRAG
<p>Denne oppgaven dokumenterer utviklingen av en webapplikasjon for Accenture sin spillgruppe. Gruppen består av 50 ansatte hos Accenture med interesse for spill. Applikasjonen har som hensikt å tilby funksjonaliteter for utlån av spill og spillutstyr til spillgruppens medlemmer.</p> <p>Gruppen har benyttet prosessmodellen Scrum som rammeverk for utviklingsarbeidet. Arbeidet ble delt opp i iterative sprinter på 2-3 uker. Mot slutten av hvert sprint ble produktet vist frem til veiledere slik at gruppen fikk tilbakemeldinger på om produktet ble utviklet etter oppdragsgivars ønsker.</p> <p>Den endelige applikasjonen er en webapplikasjon bygget med MERN-stakken (React, Node.js, Express og MongoDB), som gruppen lærte for utførelsen av oppgaven.</p>

3 STIKKORD
Webapplikasjon
JavaScript
Scrum

Forord

Høsten 2019 kom gruppen i kontakt med Accenture på Næringslivsdagene OTS ved OsloMet. Accenture presenterte muligheter for å skrive bacheloroppgaver hos dem, og viste fram flere alternativer til oppgaver. Gruppen hadde et intervju med Accenture 29. Oktober hvor vi ble kjent med hverandre og fikk høre mer om oppgaven vi var interessert i. Den 5. Desember signerte gruppen kontrakt med Accenture om utførelse av bacheloroppgave i deres regi.

Gjennom dette prosjektet har gruppen tilegnet seg mye kunnskap i alt fra prosjektutvikling i større skala til å lære seg nye teknologier. Vi har lært hvordan vi kan bruke det vi har tilordnet oss under studiene og vi har dekket opp manglende kunnskap med gode kurs på nett. Accenture var behjelpelege ved å tilby en del av kurs gjennom sin utbredte kompetanse. Gruppen fikk blant annet en introduksjon i FORM(se kapittel 3.2), som er Accenture sin fremgangsmåte i å håndtere og forbedre en problemstilling. Gruppen hadde store forventninger til oppgaven og Accenture som oppdragsgiver, og begge deler har innfridd.

Vi ønsker å takke alle som har hjulpet oss gjennom denne prosessen. Først ønsker vi å takke veiledere og produkteiere fra Accenture, Adam Asskali og Mira Lilleholt Vik, for deres uvurderlige feedback, veiledning og støtte. Videre ønsker vi å takke bacheloroppgaveansvarlige fra Accenture, Daniel Meinecke, Kjell Olav Dale og Marius Torsrud som tok oss inn til intervju, valgte oss som gruppe og la ting til rette for prosjektets gang. Til sist vil vi rette en stor takk til veileder fra OsloMet, Qian Meng, som ga feedback og hjalp oss med å fokusere på rapporten.

Innhold

I Prosessdokumentasjon	13
1 Innledning	14
1.1 Bakgrunn	14
1.2 Problemstilling	14
1.3 Prosjektmål og begrensninger	14
1.4 Kvalitetssikring	15
2 Metodikk	16
2.1 Smidig utvikling	16
2.1.1 Scrum	17
2.2 Clean Code	19
2.3 Fremgangsmåte	21
2.4 Risikoanalyse	21
2.5 Verktøy	21
2.6 Testing	22
2.6.1 Testplan	23
3 Kravutvikling	24
3.1 Oppstart	24
3.2 FORM workshop	24
3.2.1 Idémyldring	24
3.2.2 Hvordan kan vi...	25
3.2.3 Round robin	25
3.2.4 Verdi/vanskelighetsgrad-matrise	26
3.3 Definisjon av MVP	26
3.4 Kravspesifikasjon	26
3.5 Use-case diagram	27
3.6 Brukerhistorier	27

4 Teoretisk Grunnlag	28
4.1 Webtjenester	28
4.1.1 RESTful Webtjenester	28
4.2 Databaser	29
4.2.1 Relasjonell databaser	30
4.2.2 Dokumentorienterte databaser	30
4.2.3 Relasjonell eller dokumentorientert databasemodell?	30
4.3 Versjonskontroll	31
4.4 Sikkerhet	33
4.4.1 Identifikasjon, autentifikasjon og autorisasjon	33
4.4.2 Hashing	34
4.4.3 Sikkerhet i dybden	34
4.5 Testing	34
4.5.1 Automatiske og manuelle tester	35
4.5.2 Enhetstester	35
4.5.3 Integrasjonstester	35
4.5.4 Systemtester	36
4.5.5 Akseptansetester	36
4.5.6 White box testing	36
4.5.7 Black box testing	36
5 Teknologivalg	37
5.1 Frontend	37
5.1.1 React.js	37
5.2 Backend	39
5.2.1 REST-API	39
5.2.2 Database	40
5.3 Sikkerhet	42
5.3.1 JSON Web Tokens for autentisering og autorisering	42
5.3.2 Bcrypt for hashing og salting av passord	43
6 Utviklingsprosessen	44
6.1 Sprint 0 - Oppsett og arkitektur	44
6.1.1 Brukerhistorier	44
6.1.2 Databasearkitektur	44
6.1.3 Skjelett for applikasjonen	45
6.1.4 Rapportskriving	45
6.1.5 Sprint review	45

6.1.6	Sprint retrospektiv	45
6.2	Sprint 1 - Produktoversikt, registrering og innlogging	46
6.2.1	Sprintmål:	46
6.2.2	Brukerhistorier	46
6.2.3	Backend	47
6.2.4	Frontend	47
6.2.5	Coronavirus	48
6.2.6	Sprint review	48
6.2.7	Sprint retrospektiv	49
6.3	Sprint 2 - Adminfunksjonalitet og utlån	49
6.3.1	Sprintmål	49
6.3.2	Brukerhistorier	49
6.3.3	Sprint review	50
6.3.4	Sprint retrospektiv	50
6.4	Sprint 3 - Siste utviklingssprint	51
6.4.1	Sprintmål	51
6.4.2	Brukerhistorier	51
6.4.3	Sprint review	52
6.4.4	Sprint retrospektiv	52
6.4.5	Brukertest	53
II	Produktdokumentasjon	54
7	Produktbeskrivelse	56
7.1	Brukervansjonalitet	56
7.1.1	Registrering av brukere	56
7.1.2	Verifisering av e-post	57
7.1.3	Logg inn	58
7.1.4	Oversikt over produkter	59
7.1.5	Brukervarslag	62
7.1.6	Oversikt over utlån	63
7.1.7	Brukerverprofil	64
7.2	Administratorfunksjonalitet	65
7.2.1	Administrering av produkter	66
7.2.2	Administrering av kategorier	69
7.2.3	Administrering av utlån	70
7.2.4	Administrering av brukere	72
7.2.5	Administrering av forslag	74

7.2.6 Administrering av driftsmeldinger	74
8 Applikasjonsarkitektur	76
8.1 Backend	76
8.2 Frontend	77
8.3 Swagger dokumentasjon av REST API	77
8.4 Biblioteker	78
8.4.1 Frontend	78
8.4.2 Backend	79
8.4.3 Eksterne avhengigheter	80
8.5 Filstruktur	80
8.5.1 Frontend	80
8.5.2 Backend	81
8.6 Gjenbruksbare komponenter	83
9 Sikkerhet	85
9.1 Identifisering, autentisering og autorisering	85
9.1.1 JSON Web Token	85
9.1.2 Registrering	86
9.1.3 Innlogging	86
9.1.4 Aksessering av en beskyttet ressurs	87
9.1.5 Administratorfunksjonalitet	87
9.2 Lagring av passord	88
9.3 Passordkompleksitet	88
9.4 Validering	89
10 Validering	90
10.1 Validering	90
10.1.1 Frontend	90
10.1.2 Backend	90
11 Sentrale datastrukturer	92
11.1 Rentals	92
11.2 Users	97
11.3 Product	97
11.4 Service Message	98
11.5 Suggestion	99
11.6 Autentiseringstoken	99
12 Serverkonfigurasjon	101

12.1 Tilgang til server	101
12.2 Installasjon av programvare	101
12.3 Environment-variabler	102
12.4 Kjøring av applikasjonen	104
12.4.1 Frontend	104
12.4.2 Backend	104
13 API Dokumentasjon	106
13.1 Motivasjon for API dokumentasjon	106
13.2 Swagger.io	106
13.3 API Dokumentasjonen	107
13.4 Swagger-autentisering	108
14 Testdokumentasjon	109
14.1 Testverktøy	109
14.1.1 Jest	109
14.1.2 Supertest	109
14.2 Testing	110
14.2.1 Enhetstester	110
14.2.2 Integrasjonstester	110
14.2.3 Systemtester	111
14.2.4 Akseptansetester	111
14.3 Testdekningsrapporter	112
III Avslutning	114
15 Videre utvikling	115
15.1 Videreutvikling basert på brukertest	115
15.2 HTTPS til fordel for HTTP	116
15.3 Produktbilder	116
15.4 Notifikasjoner på telefon	117
15.5 Mobilapplikasjon	117
15.6 CAPTCHA	117
15.7 Innloggingsforsøk og brukeraktivitet	118
15.8 Dynamisk utlånsvarighet	118
15.9 Venteliste	118
15.10 Forlengen lånefrist	119
16 Diskusjon	120

16.1 Kravspesifikasjon og produkt	120
16.2 Videre diskusjon	120
17 Oppsummering	124
Vedlegg	129
A Gruppekontrakt	129
B Framdriftsplan	132
C Risikoanalyse	140
D Round Robin	145
E Kravspesifikasjon	150
F Use-case diagram	163
G Swagger dokumentasjon	165
H Brukeraksettansetest	210
I Integrasjons og enhetstester	239
J Sekvensdiagram av utlånslivssyklusen	245
K Ukesrapporter	250
L Møtereferater	267

Terminologi

API	“Hjelpeverktøy ved programmering, et grensesnitt mot en eller flere tjenester i et operativsystem, en databasetjener eller lignende.” [4]
Administrator	En administrator (admin) har flere rettigheter enn vanlige brukere.
Backend	Backend er delen av programvaren som ligger nærmest databasen og kommuniserer med denne. Kommunikasjonen mellom frontend og backend gjøres via et API [5].
Backlog	Brukt i iterative utviklingsmetoder som f.eks SCRUM. Backlog er oppgaver som ikke er utført enda.
Branch (git)	En gren med kode som kan skille seg fra hovedgrenen. Under utvikling lages det ofte en ny gren for å utvikle nye funksjonaliteter. En funksjonalitet kan dermed bli utviklet uten å påvirke den stabile delen av applikasjonen. Når grenen er klar, kan den flettes inn i hovedgrenen.
Brukergrensesnitt	Grensesnittet mellom en applikasjon og brukeren. I et grafisk grensesnitt er knappene en kan trykke på og informasjonen som vises på skjermen en del av brukergrensesnittet.
Bug	En feil i kildekoden til et program som gjør at koden ikke oppfører seg som tiltenkt.
Caching	Mellomlagring av data for raskere tilgang ved gjentatte, identiske forespørsler.
Commit (git)	Å committe kode vil si å lagre kodens tilstand i øyeblikket, sammen med en melding om hvilke endringer som er gjort.
DOM	Forkortelse for Document Object Model. DOM er en datamodell og et API for HTML-elementer. HTML-strukturen blir formet som objekter lagret i en trestruktur, hvor hver node kan ha flere barn. APIet blir ofte brukt av Javascript for å dynamisk endre en nettside.
Eksekvere	Relatert til programkode. Å kjøre en kodesnutt fra start til slutt.
Enhetstester	Testing av kode utført på mindre enheter. Tester enkeltfunksjoner/metoder i isolasjon for å bekrefte forventet oppførsel.

Frontend	Delen av programvaren som ligger nærmest brukeren. Koden former brukergrensesnittet som brukeren interagerer med. Kommuniserer ofte med backend-kode via et API.
Hashing	En matematisk funksjon som benyttes for å omforme data fra en tilfeldig størrelse, til en fastsatt størrelse. Brukes blant annet til å lagre passord i databaser slik at disse ikke blir lagret i klartekst.
Hosting	Publisering av filer og programvare på internett.
Inkrementelt	Brukes om inkrementell utvikling, hvor produktet leveres i små deler/inkrementer av gangen.
Interoperabilitet	“Interoperabilitet er en egenskap ved et produkt eller et system. Det innebærer at dets grensesnitt er fullstendig forstått, slik at det kan arbeide sammen med andre produkter eller systemer, nåværende eller fremtidige, i en hvilken som helst implementasjon eller tilgang, uten noen restriksjoner” [13]
Iterativt	Gjentagende/repeterende. Brukes om prosessmodeller som SCRUM, hvor utviklingen utføres i gjentagende faser.
Middleware	Deler av et program som fungerer som lim mellom to deler. Innenfor Express blir middleware brukt for å utføre operasjoner på en forespørsel, før forespørselen sendes videre til neste funksjon.
Mock	Å mocke er brukt innenfor programvaretesting. Det er ofte nødvendig å mocke avhengigheter som databasekall eller API-responser når en enhetstester kode.
Open source	Også kjent som åpen kildekode. Kildekode fra programvare som er frigitt under en lisens som tillater brukere rettigheten til å se, gjøre endringer og distribuere koden [16].
Parameter	Variablene en funksjon tar imot som inngangsverdier.
Produksjon	En applikasjon som brukes av sluttbrukere er sagt å være i et produksjonsmiljø.
Rammeverk	Brukes om programvare. Et rammeverk er en abstraksjon oppå et programmeringsspråk. Rammeverk legger til grunn en standardisert måte å bygge programvare på hvor kode blir gjenbrukt. Legger til rette for større sikkerhet og raskere utviklingstid i programvaresystemer.
Refaktorering	Endre på koden, gjerne for å forbedre kvaliteten på koden, uten å legge til ny funksjonalitet.
Render	Begrepet brukes ofte om frontend-kode som analyserer HTML-kode og viser dette på skjermen.

Routing	Innenfor webutvikling brukes routing for å rette en HTTP-forespørrelse mot koden som skal håndtere forespørsele.
Salting	Tilfeldig data som legges til som input til hashfunksjoner. Brukes f.eks for å hindre at to like passord får samme hashverdi.
Smidig utvikling	Smidig utvikling tar utgangspunkt i at programvareprosjekter er uforutsigbare og under stadig endring. Smidige utviklingsmetoder legger til rette for endringer underveis i et prosjekt, og reduserer risikoen ved kavenderinger sammenlignet med eldre utviklingsmetoder.
Spam	Uønsket informasjon.
Teknologistakk	En samling av teknologier som brukes sammen for å utføre en oppgave. Brukes som et samlebegrep om frontend, backend og database.
Token	Bruk innenfor blant annet autentisering. Et token inneholder informasjon som er signert og som vi kan stole på kilden til.
Virtuell maskin/VM	Emulasjon av en datamaskin, ofte i forbindelse med en skytjeneste. En virtuell maskin oppfattes av brukeren som en fullverdig datamaskin, men kan i prinsippet være en kraftig maskin med flere hundre virtuelle maskiner som er virtuelt adskilt.
Webapplikasjon	Et program som kjøres i en nettleser og som ikke krever installasjon av programvare.

Del I

Prosessdokumentasjon

Kapittel 1

Innledning

1.1 Bakgrunn

Accenture er et stort internasjonalt selskap som tilbyr tjenester innen strategi, teknologi, konsultasjon, operasjon, digitalisering og sikkerhet. Selskapet har 492 000 ansatte på verdensbasis i tilsammen 52 land fordelt på 200 byer. I Norge har Accenture 1100 ansatte og har hovedkontor på Fornebu. Tonje Sandberg er pr. Januar 2020 administrerende direktør i Norge.

Accenture har idag en interessegruppe kalt spillgruppa med omtrent 50 medlemmer. Gruppen holder arrangement og leier ut spillutstyr som konsoller og brettspill til ansatte i Accenture.

I dag har spillgruppa én leder og to underledere. Arrangementer avtales i [Microsoft Teams](#), og uteleie av utstyr håndteres manuelt av gruppens leder, f.eks ved bruk av notat-app på telefonen. Det finnes ingen god oversikt over hvilke utstyr som er tilgjengelige for medlemmene.

Ledere av spillgruppen opplever mye manuelt arbeid i forbindelse med utlån av utstyr, og har uttrykt ønske om en alternativ løsning.

1.2 Problemstilling

Accenture ønsker at gruppen utvikler en applikasjon for administrasjon av inventar og uteleie av utstyr til ansatte.

1.3 Prosjektmål og begrensninger

Prosjektet har som mål å forbedre dagens situasjon ved å utvikle en brukervennlig applikasjon for uteleie av utstyr. Applikasjonen skal ha bruker- og adminfunksjonalitet. Den skal fungere godt på

både mobil og PC.

Oppgaven er begrenset til utleie av utstyr og vil ikke ta for seg arrangementplanlegging, som spillgruppen også utfører. Applikasjonen skal være tilgjengelig for Accenture sine ansatte og vil potensielt bli brukt og videreutviklet etter prosjektetslutt.

Accenture setter ikke krav til bruk av teknologi. Gruppen står derfor fritt til å velge den teknologistakken som løser oppgaven på tilstrekkelig vis. Det er heller ikke satt krav til at en mobilapplikasjon skal utvikles. En webapplikasjon vil derfor være tilstrekkelig.

1.4 Kvalitetssikring

For å sikre kvaliteten på sluttproduktet har det blitt holdt jevnlige møter med veiledere. Møtene har gitt gruppen tilbakemeldinger på applikasjonen, og prosjektdokumentasjonen har bidratt til økt kvalitet på arbeidet. Det har også blitt utviklet enhetstester og avholdt brukertest for å avdekke eventuelle feil eller mangler med brukergrensesnittet.

Kapittel 2

Metodikk

2.1 Smidig utvikling

Gjennomføringen av dette prosjektet følger en smidig utviklingsmetodikk. Smidig programvareutvikling er en metodologi som oppsto som en motsetning til tradisjonell arbeidsmetodologi. Smidig utvikling fokuserer på utviklere og hvordan utviklere jobber sammen. Løsninger utvikles gjennom samarbeid mellom selvorganiserende, multifunksjonelle team, hvor hvert team har sin egen fremgangsmåte tilpasset deres oppgave [39]. Sjefens rolle i teamet er å passe på at gruppemedlemmene har det som skal til for å oppnå deres mål. Hvis dette ikke er tilfellet påser sjefen at gruppemedlemmene får de ressursene de trenger for å få utført sitt arbeid. Teamet finner selv ut av hvordan de skal løse problemet.

Smidig utvikling er et tenkesett utledet av det smidige manifestets 12 prinsipper. Prinsippene er en veileding på hvordan jobbe smidig. Grunnleggerne av manifestet oppsummerer dette: "We are uncovering better ways of developing software by doing it and helping others do it." [23]. Smidig utvikling skal følge disse prinsippene [25]:

1. Høyeste prioritet er å tilfredstille kundens behov ved tidlig og kontinuerlig levering av programvare.
2. Endring av kravspesifikasjon, selv sent i prosjektet er tillatt. Dette aspektet skaper et konkurransefortrinn for kunde.
3. Det skal leveres fungerende programvare til kunde ofte, gjerne ved et par ukers mellomrom opptil et par måneder.
4. Ansatte fra både utvikler og forretningssiden skal jobbe daglig sammen i hele prosjektet.

5. Bruk motiverte ansatte i prosjektet og gi dem miljøet de trenger for å lykkes og stol på at de for jobben gjort.
6. Formidling av informasjon inn og innad i prosjektet bør foregå ansikt til ansikt.
7. Den primære indikatoren på fremskritt er fungerende programvare.
8. Smidig utvikling promoterer bærekraftig programutvikling. Interessenter bør kunne holde et jevnt tempo under hele prosjektet.
9. Stort fokus på teknikk og design fremmer smidig utvikling.
10. Simpelhet - det er signifikant å maksimere mengden jobb som ikke blir gjort.
11. De beste arkitekturene, kravene og design kommer fra selvorganiserende grupper.
12. Teamet møtes regelmessig for å reflektere hvordan bli mer effektive og gjør nødvendige justeringer etter behov.

Ved å følge disse prinsippene, vil gruppen være bedre rustet til å levere fungerende programvare i jevnlige inkrementer og få kontinuerlige tilbakemeldinger fra oppdragsgiver. Det vil også være enklere å takle endringer i krav underveis sammenlignet med bruk av mer tradisjonelle utviklingsmetoder, noe som vil resultere i et bedre og mer tilpasset sluttprodukt.

2.1.1 Scrum

Gruppen har valgt å bruke prosessmodellen Scrum. Dette er et lettvekts, iterativt og inkrementelt rammeverk for utvikling av kompleks programvare. Styrken til Scrum er at den er lagt opp til å facilitere fortløpende forandringer i krav i tillegg til håndtering av uforutsette utfordringer [31].

Scrum deler opp prosjektarbeidet i inkrementer, kalt sprinter, med varighet på to til tre uker. Resultatet av hver sprint skal være et ferdig integrert, testet og dokumentert produkt. Etter hver sprint vil gruppen holde et møte med veildere der det reflekteres over arbeidet som har blitt gjort, i tillegg til å demonstrere produktet gruppen har utviklet.

I Scrum har en produkteier hovedansvaret for produkt-backlog-en. Dette er en liste med kravene til applikasjonen, ofte formulert som brukerhistorier. En brukerhistorie formulerer et krav sett fra sluttbrukerens perspektiv. Dette gjør at utviklingsteamet vet hvorfor de utvikler det de utvikler, og verdien dette gir [35]. En brukerhistorie er typisk formulert slik: "Som en [aktør] ønsker jeg [funksjon] for å oppnå [verdi]".

En person i teamet har rollen Scrum Master. Personen skal bistå Scrum-teamet i å nå målene sine ved å legge opp til optimale arbeidsforhold. Dette går blant annet ut på å fjerne distraksjoner

for teamet, å lære dem opp i Scrum-prinsipper og å hjelpe produkteieren med å administrere produkt-backlog-en [31]. Medlemmene i gruppen vil bytte på å være Scrum Master mellom hvert sprint.

Gruppen bruker webapplikasjonen Trello for å holde oversikt over utviklingen av produktet med en oppgavetavle.

De fire SCRUM-møtene

Scrum definerer fire møter som tar del i livssyklusen til en sprint:

1. Sprint planning

Et møte som holdes i begynnelsen av hvert sprint.

Deltakere: Produksjoner og utviklingsteamet

Hensikt: Å gå gjennom den prioriterte produkt-backlog-en og utarbeide en backlog med elementer som teamet skal fullføre innen sprintets slutt. Dette er sprintmålet. Sprintmålet skal være slik at produktet kan vises fram på slutten av sprintet.

2. Daily Scrum

Et daglig møte som avholdes for å oppdatere teamet om arbeidet og identifisere eventuelle hindringer.

Deltakere: Scrum teamet og Scrum Master

Hensikt: Hver utvikler bør svare på følgende spørsmål:

- Hva gjorde du igår?
- Hva skal du gjøre idag?
- Er det noe som hindrer ditt arbeid?

Dersom noen hindringer blir oppdaget, er det Scrum Master sitt ansvar å fjerne disse, slik at teamet kun trenger å fokusere på sprintmålene.

3. Sprint Review

Et møte på slutten av en sprint der man gjennomgår arbeidet som er blitt gjort.

Deltakere: Scrum teamet, produksjoner og eventuelt andre interesser og kunder som blir berørt av de utviklede funksjonalitetene.

Hensikt: Å vise fram arbeidet og få tilbakemeldinger slik at produktet kan bli ytterligere tilpasset til formålet. Arbeidet som blir vist fram skal møte “definition of done”, som vil si at det skal være helt ferdig.

4. Sprint Retrospective

Et endelig teammøte hvor teamet ser tilbake på arbeidet som er gjort og identifiserer elementer som kunne vært forbedret.

Deltakere: Utviklingsteamet og Scrum Master. Eventuelt produkteier.

Hensikt: Diskutere arbeidet som ble vist fram i Sprint Review og diskutere mulige forbedringsområder. Diskutere hva som går bra, hva som kunne gått bedre og komme med forslag til endringer. Målet er å drive kontinuerlig forbedring.

2.2 Clean Code

Clean Code er et tankesett som omhandler god kodeskikk og er et stort, gjennomgående fokusområde for gruppen. Robert Cecil Martin utga i 2008 boka Clean Code: A Handbook of Agile Software Craftsmanship [24]. Boken har blitt godt kjent innen programmering for å sette prinsipperne i detalj. Videre følger et utvalg av konsepter fra boka.

Navngivning er en endeløs utfordring. Et navn til en klasse, funksjon eller variabel bør besvare tre spørsmål:

Navnet skal gi deg en indikasjon på hvorfor det eksisterer, hva den gjør og hvordan den brukes. Dersom navnet gir behov for å legge til en kommentar, beskriver den med andre ord ikke intensjonen til hvorfor den eksisterer.

```
int t; //Time in seconds
```

Variabelnavnet t gir ingen indikasjon på hvorfor den eksisterer. Et bedre navn ville vært:

```
int timeMeasuredInSeconds;
```

Ved å ha beskrivende navn er det lettere å forstå og endre koden. Hva gjør denne metoden?

```
public boolean check(int number){  
    if(number >= 18){  
        return true;  
    } else {  
        return false;  
    }  
}
```

Ved å stille følgende spørsmål, blir det klart hvorfor navnet på metoden ikke er av god kvalitet:

- Hva er intensjonen med metoden?
- Hva betyr returverdiene true eller false?

- Hvilken betydning har tallet 18 i metoden?
- Hva er tallet “number” som metoden tar som parameter?

Det er her klart at metoden har forbedringspotensiale. Se på denne omskrevede metoden med samme funksjonalitet:

```
public boolean isOfLegalAge(int ageOfPerson){  
    int legalAge = 18;  
    if(ageOfPerson >= legalAge){  
        return true;  
    } else {  
        return false;  
    }  
}
```

Den nye metoden er vesentlig lettere å lese og gjør det enkelt å svare på spørsmålene stilt over. Metodenavnet forklarer hva metoden gjør og hvordan responsen skal tolkes. Tallet 18 representeres nå med en variabel og et forklarende variabelnavn. Det er tydelig at metoden returnerer true hvis alderen til personene er lik eller høyere enn myndig alder basert på metodens navn “isOfLegalAge”. Det bør ungås å velge navn som kan mistolkes. Med gode, meningsfylte navn er det lettere for andre gruppemedlemmer å sette seg inn i din kode, noe som fører til et mer effektivt team.

Å skrive gode funksjoner er et annet viktig aspekt innen clean code. Martin viser til to regler i boka: For det første skal funksjonen skal være liten, for det andre skal funksjonen være mindre enn det igjen [24]. Hva betyr så dette og hvordan kan gruppen få nytte av dette?

For å få til dette er det hensiktsmessig å skrive funksjoner som gjør én ting og kun én ting. Dette kan virke som en vag retningslinje, for det er klart det skjer mer enn en ting i en funksjon. En bedre måte å forklare dette på er at en funksjon skal kun omhandle ting et abstraksjonsnivå under funksjonsnavnet. Hele hensikten med en funksjon er å dekomponere større konsepter. Hvis det er mulig å abstrahere ut en ny funksjon med et navn som ikke bare er en re-iterasjon av nåværende funksjon, vet man om funksjonen gjør flere ting. Ved god bruk av denne retningslinjen er det lettere for gruppen å lese koden, gjøre endringer og finne bugs. Viktigheten av dette vokser eksponentielt med størrelsen på prosjektet. Det er også viktig å unngå repetisjon, da dette potensielt kan duplisere feil i koden.

En god kommentar er uvurderlig, men skal på ingen måte erstatte dårlig kode. Ved behov for mye kommentarer er det mest sannsynlig for å veie opp for dårlig kode, og en mer konstruktiv tilnærming ville vært å skrive om koden. Koden skal være selvforklarende slik illustrert i funksjonen

```
boolean isOfLegalAge
```

Å skrive en kommentar i dette tilfellet er overflødig.

2.3 Fremgangsmåte

Gruppen har valgt å bruke en teknologistakk som ingen på gruppen tidligere har erfaring med. Av den grunn ble det satt av én måned, fra 14. januar til 14. februar, til å lære oss teknologiene vi bruker i prosjektet. Vi brukte platformen Udemy for å finne kurs om Node.js og React.js. Denne tidsinvesteringen førte til at vi kom senere i gang med utviklingen. Likevel hadde vi god oversikt over teknologiene vi skulle bruke da vi startet med utvikling den 17. februar.

Gruppen avtalte arbeidstid og skrev gruppekontrakt i begynnelsen av prosjektet (vedlegg A). Dette gjorde at medlemmene hadde visse retningslinjer å jobbe etter, og har bidratt til å minimere utfordringer underveis.

En framdriftsplan ble utarbeidet og er inkludert i vedlegg B. Planen er presentert slik den ble utarbeidet i begynnelsen av prosjektet. Grunnet at det ikke ble planlagt påskeferie i planen, ble sprint 2 utvidet med en uke. Dette gjorde at sprint 3 ble forskjøvet med en uke og at designsprintet ble kuttet ut til fordel for mer tid til rapportskriving. Mer om dette i kapittel 6.

Underveis har gruppen skrevet ukesrapporter og møtereferater av møter med veiledere for å loggføre så mye som mulig av arbeidet. Disse finnes henholdsvis i vedlegg K og L

2.4 Risikoanalyse

En risikoanalyse ble tidlig utarbeidet for å avdekke eventuelle faremomenter ved prosjektet. I korte trekk kom gruppen fram til at det å jobbe etter smidig metodikk minimerer risiko ved endring av krav. Det vil si at funksjonalitet prioriteres av produkteier og demoer avholdes etter hvert sprint. Et annet viktig tiltak er at alle gruppemedlemmer har oversikt over både frontend og backend-delen av teknologistakken. Dette minimerer risikoen ved kort- eller langvarig sykdom hos en av gruppemedlemmene. Risikoanalysen finnes i vedlegg C.

2.5 Verktøy

I løpet av prosjektet har det blitt benyttet flere verktøy for den praktiske gjennomføringen av arbeidet. Under følger en liste over de mest brukte:

Verktøy	Beskrivelse
---------	-------------

Git	Git er et distribuert versjonshåndteringssystem. Det er vidt brukt innen fagfeltet for å følge endringer av kildekode i programvare. Gruppen har brukt Git i kombinasjon med Github , som gir online samarbeidsfunksjonalitet for Git-prosjekter.
Trello	Et gratis verktøy for å opprette tavler, lister og kort for å organisere arbeid. Verktøyet er tatt i bruk for å organisere og prioritere arbeidet underveis i prosjektet.
Lucidchart	En webapplikasjon for tegning av diagrammer. Verktøyet har mange ferdiglagde komponenter laget for velkjente IT-diagrammer som f.eks UML.
Slack	Kommunikasjonsverktøy med funksjonalitet for oppretting av kanaler. Verktøyet har blitt brukt for å fordele informasjonsflyten i prosjektet i relevante kanaler (f.eks frontend og backend) slik at en enkelt kan finne tilbake til relevant informasjon når det trengs. Verktøyet ble også brukt til kommunikasjon med veiledere.
Visual Studio Code	Teksteditor som hovedsaklig har blitt brukt for å utvikle kode i prosjektet. Editoren har også innebygget funksjonalitet for git, noe som gjorde det enkelt for gruppens medlemmer å se over endringer før commits.
Discord	Gratis kommunikasjonsverktøy hvor det er mulig å kommunisere med andre via chat, bilde, lyd eller video. Primært brukt til talekommunikasjon mellom gruppemedlemmer, men også med veiledere.
Teamviewer	Et verktøy som primært er laget for fjernstyring av datamaskiner, men brukes i bachelorprosjektet til presentasjon og samarbeid via deling av skrivebord.

2.6 Testing

Testing er en viktig del av utviklingen til et programvaresystem. Hensikten er å sørge for at systemet er fri for defekter og at de faktiske resultatene stemmer overens med de forventede. I praksis gjøres tester ved at man eksekverer softwarekomponenten eller systemkomponenten som skal testes for å evaluere en eller flere egenskaper som er av interesse [45]. Testing kan utføres både manuelt og automatisk, men noen tester eigner seg mer for å automatiseres. Automatiske tester utføres ved å skrive testkode som eksekverer en begrenset del av koden som skal testes. Testene kan settes til å kjøre hver gang koden endres for å verifisere at endringene ikke gjør at noen av testene feiler. Dersom en test feiler, er det en god indikator på at kodeendringen kan ha uønskede effekter på deler av applikasjonen. I slike tilfeller må koden endres slik at testen lykkes. Eventuelt må testen skrives om for å reflektere endringene i funksjonalitet, dersom disse var med hensikt.

2.6.1 Testplan

Gruppen vil under prosjektet ha hovedfokus på testing av backend. Dette vil sørge for at backend fungerer som tiltenkt, at databasen holdes i konsistent form til enhver tid og at alle endepunkter som skal ha begrenset tilgang faktisk er sikret. Gruppen har satt seg som mål å ha minst 75% testdekning av backend. Testing utføres i Jest [Jest](#), et rammeverk for testing i Javascript. Når det kommer til frontend vil denne være i stadig endring under prosjektet, og gruppen ble derfor enige om å ikke legge vekt på automatisk testing her, da det ville ta verdifull tid fra utviklingen av produktet.

Kapittel 3

Kravutvikling

3.1 Oppstart

I oppstartsfasen av prosjektet ble det avholdt et møte med Accenture. I løpet av møtet fikk gruppen en oversikt over oppgaven og Accenture sin rolle i prosessen. Et overordnet bilde av dagens situasjon ble dannet. Gruppen fikk så utdelt veiledere og produkteier hos bedriften.

3.2 FORM workshop

Samme uke holdt Accenture en workshop for gruppen sammen med andre bachelorgrupper. Workshopen tok utgangspunkt i en rekke konsepter under begrepet FORM. Dette er Accenture sin samling av et felles tankesett for innovativ og ansvarsfull utvikling. FORM inkluderer forskjellige konkrete og iterative arbeidsmetoder som er ment for å forme innovasjon og å avdekke nye muligheter.

Hensikten med workshopen var å gå fra det store bildet og dagens situasjon, til å se muligheter og utarbeide konkrete idéer som kan forbedre situasjonen. Denne dagen dannet et godt utgangspunkt for prosjektets utvikling.

3.2.1 Idémyldring

I første del av workshopen ble det holdt en sesjon med idémyldring på post-it lapper. Idémyldringen ble gjort individuelt for å ikke påvirke hverandres kreativitet. Hensikten med denne metoden var å få ned så mange elementer som gruppen tror vil være en del av prosjektet. Her var det fritt frem for å skrive alt fra produktkrav til krav fra universitetet.

Deretter ble elementene gruppert i relaterte grupper på en tavle. Dette avdekket følgende hoved-

grupper:

- Funksjonaliteter for brukere og admin
- Funksjonaliteter på serverside
- Ikke-funksjonelle krav som sikkerhet, grensesnitt og ytelse.

3.2.2 Hvordan kan vi...

Den neste delen i den kreative prosessen heter “How might we...” og går ut på å stille seg forskjellige spørsmål på formen “Hvordan kan vi...?” som relaterer seg til utfordringene man har. Spørsmålene blir så gruppert og generalisert. Gruppen satt til slutt igjen med tre spørsmål som ble tatt med videre:

- Hvordan lage en app som er lett å overta?
- Hvordan forbedre dagens løsning for admin og brukerne?
- Hvordan skape gode brukeropplevelser?

3.2.3 Round robin

Spørsmålene som ble tatt opp i forrige prosess, ble med videre i en ny metode kalt “Round Robin”. Round Robin går ut på å finne løsninger på hver utfordring, for så å kritisere løsningen og komme fram til en revidert versjon.

Denne prosessen ble utført ved å fylle ut et skjema med følgende punkter:

- Challenge statement
- Proposed solution
- Why the solution will fail
- Final concept

Skjemaet blir sendt til neste person i gruppen for hvert punkt, slik at meningene til flere på gruppen blir med i det endelige konseptet. Utifra konseptene som kom frem i Round Robin, tok gruppen en ny runde med å lage post-it lapper ut av konseptene, før lappene ble lagt til på den samme tavlen brukt i idémyldringen over.

3.2.4 Verdi/vanskelighetsgrad-matrise

Siste runde i kravutviklingen foregikk ved å plukke ut konsepter fra tavlen og plassere de langs en x-akse hvor de mest verdifulle konseptene ble plassert lengst til høyre. Deretter ble det tegnet en y-akse hvor vanskelighetsgrad ble tatt i betrakning. Lappene skulle da flyttes utifra forventet vanskelighetsgrad. Konseptene med høy verdi og lav vanskelighetsgrad ble kandidater for en MVP (Minimum Viable Product).

3.3 Definisjon av MVP

MVP står for Minimum Viable Product (minste brukbare produkt), og definerer det enkleste produktet som kan lages for å tilfredsstille nøkkelfunksjonalitetene ønsket av produkteier. Utifra verdi/vanskelighetsgrad-matrisen (se 3.2.4) definerte gruppen en MVP med følgende funksjonaliteter:

- Brukere kan logge inn.
- Brukere kan sende forespørsel om å låne spillutstyr.
- Brukere kan sette seg på venteliste for utlån.
- Brukere kan se oversikt over sine lån.
- Brukere kan søke blant inventaret.
- Administrator kan logge inn.
- Administrator kan godkjenne eller avslå utlån.
- Administrator kan få oversikt over utlån.
- Administrator kan administrere inventaret.

Kravene definert her ble prioritert høyest i utviklingsprosessen.

3.4 Kravspesifikasjon

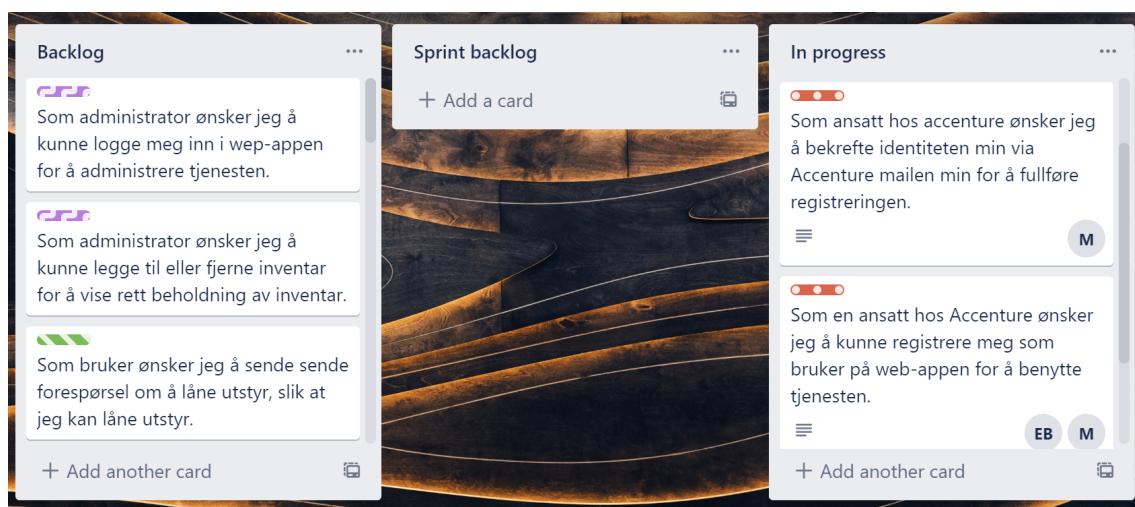
En kravspesifikasjon ble utarbeidet ut ifra verdi/vanskelighetsgrad-matrisen. Kravspesifikasjonen går mer detaljert inn på funksjonaliteter og deler de funksjonelle kravene inn i de to identifiserte primæraktørene **bruker** og **administrator**. Kravspesifikasjonen følger som vedlegg E. Dokumentet ble godkjent av produkteier og dannet grunnlaget for brukerhistoriene som ble utarbeidet og brukt i utviklingsfasen.

3.5 Use-case diagram

Det siste steget i kravutviklingen var å lage et use-case diagram. Use-case-diagrammet gir en enkel oversikt over applikasjonens funksjonalitet, hvilke aktører som initierer use-case og hvilke aktører som er nødvendig for å fullføre et use-case (sekundære aktører). Use-case diagrammet finnes i vedlegg F.

3.6 Brukerhistorier

Som nevnt i kapittel 2.1.1, er det i SCRUM-metodikk normalt å definere kravene i form av brukerhistorier for å se produktet fra brukerens perspektiv. Med kravspesifikasjonen som utgangspunkt ble det utviklet brukerhistorier fra disse. Noen av kravene var store, slik at det ble utviklet flere brukerhistorier ut ifra ett krav. Det ble også avdekket at det manglet noen krav i kravspesifikasjonen, slik at denne ble noe endret i forbindelse med prosessen. Brukerhistoriene ble lagt inn i en produkt-backlog i Trello, slik at de var klare før planleggingen av første utviklingssprint (kapittel 6.2). I figur 3.1 vises et utsnitt fra Trello-tavlen i slutten av sprint 1. Brukerhistoriene er listet opp i kapittel 6 for hvert sprint.



Figur 3.1: Brukerhistorier i slutten av et sprint

Kapittel 4

Teoretisk Grunnlag

I dette kapittel går vi dypere inn på teorien bak webtjenester, databaser, versjonkontroll, sikkerhet og testing.

4.1 Webtjenester

En webtjener kjører på en server og eksponerer et felles grensesnitt som ulike systemer, uavhengig av hvilket språk de er skrevet i, kan kommunisere med hverandre over internett. Webtjenester muliggjør løs kobling av systemer, som bidrar til interoperabilitet. Dette er mulig fordi kommunikasjonen alltid foregår med samme protokoller, som f.eks Hyper Text Transfer Protokoll (HTTP) og Simple Mail Transfer Protokoll (SMTP). Dataen som sendes i pakkene er som oftest på et definert format som XML eller JSON. Slik kan for eksempel en webtjener for netthandel konsumeres av en nettleser, en mobilapplikasjon, en nettbrettapplikasjon og fremtidige systemer som alle kan være skrevet i forskjellige språk.

Det finnes hovedsakelig to typer webtjenester, Simple Object Access Protocol (SOAP) og Representational State Transfer (RESTful) webtjenester. [37]

4.1.1 RESTful Webtjenester

RESTful webtjenester er ikke en protokoll, men en arkitekturstil. Roy Fielding definerte REST i sin doktorgradsavhandling. REST har ingen rigid struktur, så RESTfulle webtjenester kan for eksempel bruke flere type filformat for å sende data, som HTML, JSON, XML. Arkitekturstilen kommer med et sett av begrensninger for hvordan webtjenester skal utvikles. Hvis man innfører disse kravene, kalles webtjenesten RESTful. Samsvær med disse begrensningene gir ikke-funksjonelle gevinster for webtjenesten, som blant annet hastighet [27].

For at et API skal bli defunert som RESTful, må det oppfylle følgende krav:

Klient-Tjener arkitektur

Kravet om en klient-tjener-arkitektur kommer fra prinsippet “separation of concerns” [38] som går ut på å separere dataene til et system fra hvordan det presenteres i klienten. Dette leder til interoperabilitet, da webtjenesten blir platformuavhengig.

Ingen tilstandslagring på server

Serveren skal ikke lagre tilstand mellom klientforespørsler. Med andre ord blir hver forespørsel til webtjeneren behandlet som om det er en ny forespørsel.

Dersom det er behov for tilstandslagring er det klienten som har ansvaret for å sende all tilstrekkelig informasjon for å muliggjøre dette. For å fasilitere for eksempel autentisering og autorisasjon så kan JSON Web Token (JWT) brukes (for mer informasjon om JWT les: [9.1.1.](#).)

Mulighet for caching

Caching blir brukt for å minimere last på serveren og for å bidra til raskere responstid på klientsiden. Hver respons fra serveren skal inneholde et felt som sier om responsen er mulig å cache, og isåfall, hvor lenge responsen kan caches på klientsiden. Denne begrensningen bidrar til hastighet og tilgjengelighet, men bakdelen er at klienten kan sitte på data som er utdatert.

Lagdelt system

REST gjør det mulig å lagdele backend-arkitekturen slik at APIet ligger på en server, mens persistenslagringen (databasen) ligger på en annen server. Dette enkapsuleres i backend, slik at klienten ikke er klar over implementasjonsdetalsjene på serveren.

Uniformt grensesnitt

Uniformt grensesnitt er en begrensning som krever at det skal være en uniform måte å interagere med webtjenesten på, uavhengig av platform [\[27\]](#).

4.2 Databaser

“En database er en samling data lagret på et elektronisk medium. Datasamlingen er organisert og strukturert etter en bestemt strategi eller modell - en databasemodell.” [\[12\]](#).

4.2.1 Relasjonell databaser

Relasjonelle databaser er digitale databaser som er basert på den relasjonelle datamodellen, foreslått av E. F. Codd i 1970 [10]. Data i en relasjonell modell er organisert i tabeller som består av rader og kolonner. Tabellene, også kjent som relasjoner, lagrer informasjon om ulike klasser av data, som for eksempel kunder eller produkter. Rader er unike instanser av disse entitetene, som for eksempel kunde Per Petterson, 1994. Kolonnene er attributter til tabellene. I attributtene lagres data om entitetene som navn, alder, gate, osv.

4.2.2 Dokumentorienterte databaser

I dokumentorienterte databaser følges ikke den relasjonelle modellen, men dokumentmodellen. I denne modellen kalles tabellene/entities for Kolleksjoner. I motsetning til relasjonelle tabeller, som har en bestemt struktur, eller schema, så har kolleksjoner en dynamisk struktur. Radene i en relasjonell modell er erstattet med dokumenter. En kolleksjon er en samling av dokumenter. Et dokument er et ordnet set med nøkkel-verdi par. Et dokument kan inneholde et annet dokument, hvilket muliggjør en hierarkisk struktur. Dette leder til raskere spørninger, da man unngår kostbare join operasjoner [8].

4.2.3 Relasjonell eller dokumentorientert databasemodell?

Relasjonelle databaser bruker Structured Query Language (SQL) for å generere og håndtere data. SQL er et veldig allsidig språk som muliggjør avanserte spøringer. En mulig bakdel med SQL er at det krever at strukturen på data skal defineres før data kan legges til eller hentes. All data må derfor følge denne samme strukturen. En naturlig konsekvens av dette er at databasemodellingsprosessen blir veldig fortung, altså kreves det mye arbeid i begynnelse før man får en database man kan integrere i systemet. I programvareprosjekter hvor kravene stadig endrer seg, kan det være gunstig å bruke en databasemodell som tillater fleksibel og dynamisk datalagring [30].

Skalering av en database vil si å oppgradere en database for lagring av større mengder data eller håndtering av en større mengde forespørsler enn det for tiden er kapasitet til. Skalering er enten horisontal eller vertikal. Vertikal skalering handler om å oppgradere til en kraftigere server med bedre CPU, RAM, SSD osv. Horisontal skalering, derimot, går ut på å dele dataen opp mellom flere servere. Logistikken i vertikal skalering er enklere, men det er dyrere enn horisontal skalering. Dersom databehovet blir stort nok vil det til slutt ikke finnes en kraftig nok server å oppgradere til. Da blir det behov for horisontal skalering. Dokumentorienterte databasemodeller, i motsetning til relasjonelle, er bygget for å enkelt legge opp til horisontal skalering [30][8].

4.3 Versjonskontroll

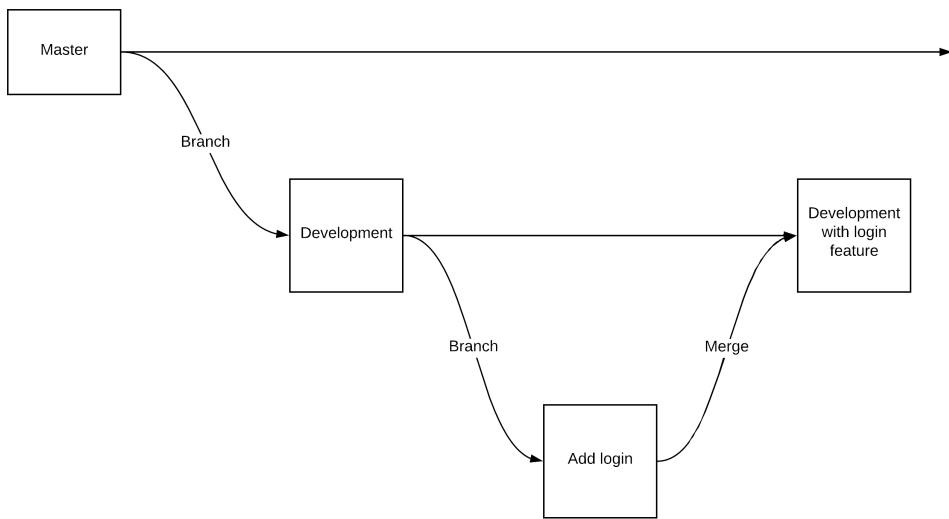
Git er et gratis og open-source-distribuert versjonshåndteringssystem [14]. Et versjonshåndteringssystem er en kategori av programvare som hjelper til med å loggføre endringer av filer og endringer av kode i filen [36]. Et repository er en database av endringer og inneholder alle modifikasjoner og versjoner av et prosjekt. Copy of work/checkout er din personlige kopi av alle filene i prosjektet. Denne kopien kan editeres uten at det påvirker andre som jobber i prosjektet. Endringer gjort kan så bli lastet opp ved å gjøre en commit.

Et distribuert versjonshåndteringssystem inneholder flere repositories hvor hver bruker har sitt eget repository med tilhørende copy of work. En commit her vil kun vise til endringer gjort i sitt eget lokale repository. For at disse endringen skal bli en del av et sentralt repository må det utføres en push. Andre brukere kan da utføre en pull for å laste ned din push. Kort sagt må fire ting skje: Du utfører en commit, så en push. De gjør så en pull og eventuelt en nødvendig update.

Et distribuert versjonshåndteringssystem har en rekke fordeler:

- Versjonshåndtering har oversikt over tidligere utgaver av prosjektet. Dette kan bli benyttet som en forsikring i tilfelle en pc blir ødelagt og/eller data går tapt. Da er det mulig å hente ut data fra siste versjon.
- Ved feil er det mulig å gå tilbake til en tidligere versjon. Eksempelvis gå tilbake til en stabil versjon.
- Det er mulig å fjerne spesifikke commits uten å endre andre commits.
- Det gir komplett oversikt over hvem, hvorfor og når en commit ble utført. Dette gjør blant annet prosessen ved feilsøking enklere.
- Flere kan jobbe på samme prosjekt samtidig. Hver og en jobber på sin copy of work og bestemmer selv når endringen blir delt med teamet.
- Det gjør det mulig for hver enkelt å jobbe på flere maskiner.
- Det tillater kontinuerlig oppdatering av prosjekt fra teamet. Ved en merge konflikt, som kan oppstå dersom det er gjort endringer på samme linje i samme fil av forskjellige medlemmer, ber versjonshåndteringssystemet om hjelp til å bestemme hva som skal gjøres.

Git bruker en branching model [2]. En branch er en versjon av repository. Det er en uavhengig gren av prosjektet. Git tillater med dette å ha flere lokale branches som er uavhengige av hverandre. Det er da mulig å ha en branch som inneholder hva som går til produksjon og egne branches for testing. Branches kan også være features som skal implementeres. Branches har mange bruksområder.

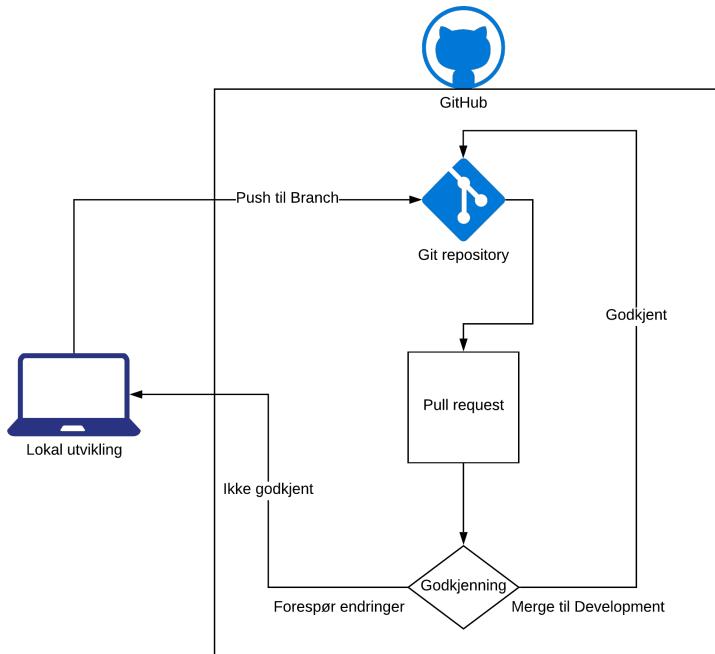


Figur 4.1: Git eksempel på branch

Figur 4.1 viser hvordan gruppen bruker et Git repository med branches. Master branch inneholder en stabil utgave av applikasjonen. Development-branchen er en klone av master branchen og kan inneholde nye funksjonaliteter som fortsatt er under utvikling. Ved utvikling av nye funksjonaliteter kan man lage en ny branch av Development, for eksempel en “Add login” branch for å lage login-funksjonalitet. Når denne funksjonaliteten er ferdigutviklet, kan den flettes inn i Development ved å utføre en merge. Skulle en feature branch vise seg å inneholde feil eller være uønsket er det enkelt å fjerne mergen ved å utføre en omvendt commit som fjerner endringene.

Github er en webtjeneste som hoster git repositories [15]. De tilbyr også en rekke andre nyttige tjenester slik som å kunne få oversikt over commits i et grafisk grensesnitt. Ved å benytte Github får man også mulighet til å lage regler for et repo, f.eks at det kreves en pull request for å kommit ny kode til spesifikke branches. Det betyr at når en på teamet har ferdigutviklet en funksjonalitet som ønskes flettet inn i development, må denne godkjennes av en eller flere andre medlemmer. Personene som godkjener går da gjennom koden for å kommentere og eventuelt be om endringer. Dette gjør det mulig å kontrollere hverandres kode og øke kodekvaliteten i et prosjekt. Når koden er godkjent, kan den flettes inn automatisk.

Figur 4.2 illustrerer hvordan gruppen benytter seg av GitHub. Et gruppemedlem jobber lokalt på sin pc i en feature branch. Gruppemedlemmet pusher så sin commit til en feature branch i git repository. Gruppemedlemmet ønsker deretter å pushe sin commit til development branch og må opprette en pull request. Blir den ikke godkjent må endringer utføres. Blir den godkjent flettes branchen med development.



Figur 4.2: Git og GitHub arbeidsflyt

4.4 Sikkerhet

I dette kapittelet forklares sikkerhetsterminologien som er gjeldende for vår applikasjon, samt teorien bak disse sikkerhetskonseptene.

4.4.1 Identifikasjon, autentifikasjon og autorisasjon

Identifikasjon

Identifisering vil si å hevde at man har en identitet. Systemet som ber om identiteten, i form av f.eks personnr, email, brukernavn, bruker dataen for å unikt identifisere den som ønsker å bruke systemet.

Autentifikasjon

Autentifisering vil si å bekrefte at man har den identitet man hevder å ha. Dette kan gjøres ved å sende inn informasjon som kun den som har identiteten kan vite, som f.eks et passord.

Autorisasjon

Etter at en bruker har gitt sin identitet og bekreftet den overfor et system, beskriver autorisering de rettighetene denne autentiserte brukeren har. Autorisering er knyttet til handlinger som brukere

kan eller ikke kan utføre avhengig av autorisasjonsnivået deres [28].

4.4.2 Hashing

Hashing er en en-veis matematisk funksjon som tar input på vilkårlig størrelse og produserer en output på en fast størrelse. Egenskaper ved hashingalgoritmer er at de er deterministiske, pre-image resistente og kollisjonsresistente [11]. Hashing brukes blant annet til å lagre passord i en database slik at det ikke er mulig å finne ut av passordet selv om innholdet i databasen lekkes.

- Determinisme innebærer at en hashfunksjon alltid skal produsere samme hashverdi h for samme input melding m .
- Pre image resistant vil si at dersom du kjenner til en hashverdi h , så er det beregningsmessig nesten umulig å regne ut meldingen m som produserte hashverdien $h(m) = h$.
- Kollisjon vil si at to verdier (x, y) har samme hashverdi $h(x) = h(y)$. Dette er umulig å unngå, siden en hashfunksjon konverterer en variabel størrelse input, til en verdi med fast størrelse. Men det skal være beregningsmessig veldig vanskelig for en angriper å finne to verdier som genererer samme hashverdi.

4.4.3 Sikkerhet i dybden

Lagdeling av sikkerhetstiltak, også kjent som sikkerhet i dybden, er et viktig prinsipp innenfor datasikkerhet. Dersom et av sikkerhetstiltakene svikter, vil forhåpentligvis de resterende tiltakene kompansere for det sviktende laget [44]. Lagdelt sikkerhet motvirker fallgruven kjent som “single point of failure” [1].

4.5 Testing

I programvareutvikling så kan mye gå galt. Feil i kodelogikken kan blant annet krasje programmet, føre til uønsket funksjonalitet eller åpne opp for sikkerhetshull som angripere kan utnytte. Av den grunn er det viktig å teste koden for å verifisere at den produserer et tilsvikt resultat. Testing kan klassifiseres etter hvordan testene blir utført, da enten automatisk eller manuelt av en tester. Testene kan også gruppere etter koden som skal testes, da deles de inn i enhetstester, integrasjonstester, systemtester og akseptansetester. Tester kan ytterligere deles inn i to hovedkategorier basert på om koden er synlig for testerne eller ikke. Disse kalles henholdsvis white box og black box testing.

4.5.1 Automatiske og manuelle tester

Automatiserte tester tar i bruk automasjonsrammeverk for testing. Testingen foregår ved at automasjonsrammeverket kjører testkode som automatisk tester funksjonaliteten i programmet. Automatiske tester er velegnet for prosjekter som krever test av den samme funksjonaliteten gjentatte ganger [41]. Utviklere eller testere skriver testkoden. Eksempler på automasjonsrammeverk for testing er Xunit for C#, JUnit for Java og Jest for Javascript.

Manuelle tester utføres av mennesker uten bruk av automasjonsverktøy. En fordel med manuell testing er at et menneske kan plukke opp implikasjoner av et testresultat som en automatisk test ville oversett. Manuell testing er mer tidskrevende, men man slipper å skrive og vedlikeholde testkode [48].

4.5.2 Enhetstester

En enhetstest tester at isolerte deler/enheter av programvaren oppfører seg som forventet. Det er viktig at enhetene er isolerte, altså ikke har eksterne avhengigheter som databaser eller filer. En enhet er den minst testbare funksjonaliteten i programvaren. I praksis er dette funksjoner, metoder, klasser og grensesnitt. Enhetstester kan skrives enten før eller etter programvaren som testes er skrevet. Figur 4.3 viser en enhetstest av kvadratrotfunksjonen til .NET Math klassen. Her testes det ved å sammenligne det forventede utfallet av å ta kvadratroten av heltallet 4 med det faktiske utfallet fra funksjonskallet[46].

```
[Fact]
public void Math_Sqrt_GivenPositiveInteger_ComputesCorrectly()
{
    //Arrange
    var expected = 2;

    //Act
    var actual = System.Math.Sqrt(4);

    //Assert
    Assert.Equal(expected, actual);
}
```

Figur 4.3: Enhetstest med rammeverket Xunit

4.5.3 Integrasjonstester

En integrasjonstest tester ulike komponenter av programvaren som en gruppe, for å teste hvordan de interagerer med hverandre. Integrasjonstester fokuserer dermed hovedsakelig på dataflyt mellom komponenter [18]. En integrasjonstest kan for eksempel teste at man blir omdirigert til en viss side etter man har logget inn eller at et API interagerer med en database og en klient.

4.5.4 Systemtester

En systemtest tester den fullstendige integrerte programvaren som en helhet. Det er under systemtesting at de tekniske kravene til applikasjonen blir testet. Funksjonelle og ikke-funksjonelle krav blir verifisert via systemtesting. En systemtest skjer etter integrasjonstester og før akseptansetester [33].

4.5.5 Akseptansetester

En akseptansetest tester hele systemet. Her bekrefter man om koden tilfredsstiller brukernes behov. Dette er det siste steget av testing før produktet leveres til sluttbrukerne. En akseptansetest er ikke en systemtest (som tester at systemet innfrir de tekniske kravene til programvaren), men den tester at systemet fungerer for brukeren.

Akseptansetester kan utføres av sluttbrukerne til applikasjonen eller av medlemmer av organisasjonen som forespurte utviklingen av programvaren [3].

4.5.6 White box testing

White box testing er testing hvor programvarens interne struktur, design og kode testes. I denne formen for testing er koden synlig for testeren. White box tester skrives som regel av utviklere [47]. Utvikleren skriver kode for å eksekvere samtlige linjer i koden som testes og verifisere at en gitt input gir forventet output.

4.5.7 Black box testing

Black box testing er en teknikk hvor funksjonaliteten til applikasjonen testes uten å se på den interne kodestrukturen, implementasjonsdetaljer og vitenhet om de interne stiene i applikasjonen. Denne formen for testing er basert på systemets krav og spesifikasjoner [42].

Kapittel 5

Teknologivalg

Accenture har gitt gruppen frihet til å velge teknologi til utviklingen av webapplikasjonen. Teknologi har derfor blitt valgt ut ifra gruppemedlemmene s ønske og erfaring.

5.1 Frontend

Gruppen vurderte React, Vue og Angular for å lage en frontend-applikasjon til å kommunisere med REST-APIet i backend. Accenture hadde mest erfaring med React-applikasjoner og gruppemedlemmene ønsket å lære React. Derfor ble det bestemt å benytte dette for applikasjonen.

5.1.1 React.js

React er et JavaScript-bibliotek til utvikling av brukergrensenett i frontend. Biblioteket er utviklet og vedlikeholdt av [Facebook](#) [26].

React oppgave er å rendre data til DOM, og byr ikke på ferdigbygget funksjonalitet slik biblioteker som f.eks Angular tilbyr. Dette gjør at React ofte må brukes med andre javascript-biblioteker for å utvide funksjonaliteten. Et bibliotek som er vanskelig å komme utenom i React-applikasjoner er [React Router](#), som tilbyr Routing-funksjonalitet.

I sammenheng med React, brukes det som oftest JSX, som er en syntaksutvidning av Javascript. JSX ligner på HTML, men er egentlig funksjonskall som evalueres til Javascript-objekter når koden eksekveres [19]. Dette tillater å kombinere relatert kode for logikk og brukergrensesnitt i samme fil og øker lesbarheten. Figur 5.1 viser en funksjon som returnerer JSX basert på om **user** er definert.

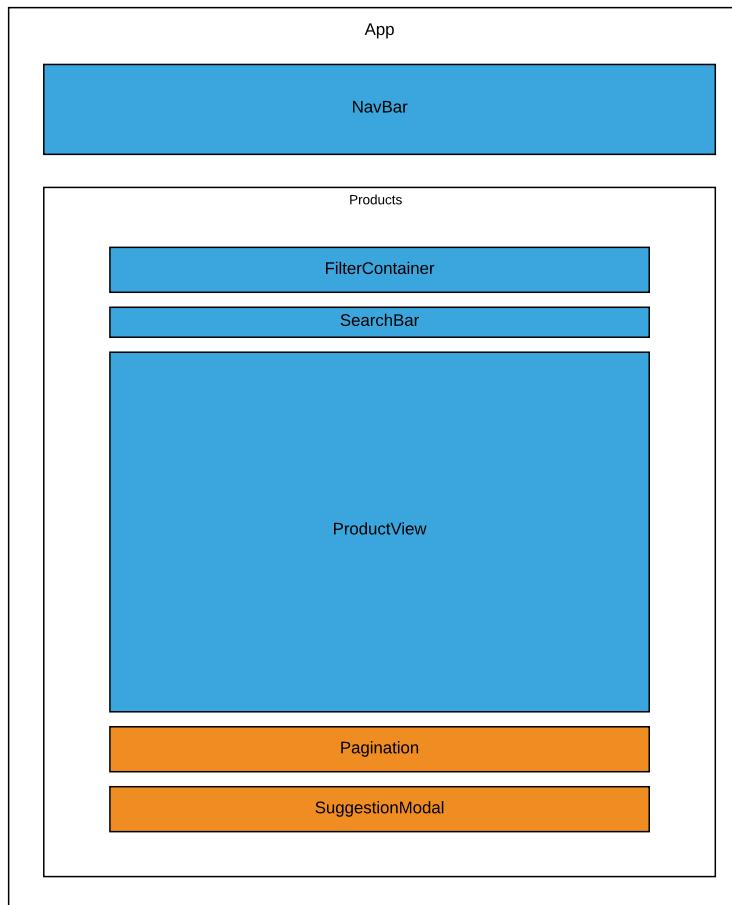
React sies å være komponentbasert. En komponent kan inneholde flere underkomponenter. Et eksempel på dette fra vår ferdigbygde applikasjonen ses på figur 5.2. Her ser vi hovedkomponen-

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

Figur 5.1: Eksempel på JSX [19]

ten App. Denne inneholder underkomponentene NavBar og Products. Products inneholder igjen underkomponentene FilterContainer, SearchBar og ProductView. I tillegg ser vi komponentene Pagination og SuggestionModal, som kun vises dersom noen kriterier er oppfylt. Her er Pagination en sidetallsvelger som vises dersom antall produkter overskrider et visst antall.

Hver klassekomponent inneholder et objekt kalt state. Objektet kan brukes til å lagre tilstand. State i Products-komponenten brukes blant annet til å lagre hvilken kategori som er valgt i FilterContainer, hvilket søkeord som er skrevet i SearchBar og hvilket sidenummer som er trykket på i Pagination. Dette lagres i Products istedenfor i de respektive komponentene som tar imot inputen. Grunnen til dette er at data da kan bli delt med alle underkomponentene som krever dette. F.eks er sidenummer som velges i Pagination relevant for hvilke produkter som skal vises i ProductView. Når state-objektet oppdateres, utfører react en ny render som oppdaterer alle komponenter som forandrer seg som følge av endret state. For eksempel blir ProductView-komponenten rendret på nytt hver gang søkeordet forandrer seg.



Figur 5.2: Komponenter og underkomponenter

5.2 Backend

Backend-arkitekturen for applikasjonen består av et RESTful API som benytter seg av JSON-standarden. Data lagres i en MongoDB-database. Tilstand vil alltid bevares i frontend, og alle forespørsler til APIet inneholder et JWT-token for å gjennomføre forespørselen dersom det trengs autentisering og/eller admintilgang for operasjonen.

5.2.1 REST-API

For å utvikle REST-APIet falt valget for en server utviklet i [Node.js®](#) med rammeverket [Express](#) på toppen.

Node.js

Node.js er et runtimemiljø for javascript på serversiden. JavaScript koden kjøres på server ved hjelp av Google sin V8 motor.

Styrken ved Node.js er evnen til å benytte seg av asynkrone I/O operasjoner, noe som maksimerer båndbredde og gjør applikasjonen svært skalerbar. Når en forespørsel behøver å vente på en I/O operasjon vil denne ikke blokkere andre forespørsler. Dette utføres med asynkron programmering og uten bruk av tråder, som tradisjonelt sett har vært tatt i bruk for blokkerende operasjoner.

Node.js er derimot kun et kjøremiljø, noe som gjør at det ofte blir benyttet med rammeverk på toppen for å utvikle webservere, da det vil være tidkrevende å skrive en webserver fra bunn.

Express

Express er et lettvekts rammeverk som kjører oppå Node.js. Express kommer med flere funksjonaliteter for webapplikasjoner. Blant annet funksjonaliteter for konstruering av REST/RESTful APIer. APIet til express er forholdsvis lite og enkelt å forstå. Rammeverket tilbyr ulike HTTP metoder for å enkelt håndtere ruting, forespørsler og svar fra klienter.

Middleware er en sentral del av Express. I bunn og grunn er dette funksjoner som har tilgang på objektene request, response og next. En middleware-funksjon kan endre på request-objektet, stoppe en forespørsel eller sende forespørselen videre til neste middleware i stakken ved å benytte next-funksjonen. Når en forespørsel kommer inn på serveren vil den kunne rutes gjennom flere middleware-funksjoner som har ulike funksjonaliteter, som for eksempel validering og autentisering. Disse stopper forespørselen og gir relevant respons dersom noe er feil. En type middleware som har blitt benyttet i prosjektet er for autentisering. Funksjonen tar imot request-objektet med et JWT-token i header, dekoder det og validerer signaturen. Dersom tokenet er gyldig legger den til et nytt felt for bruker i request-objektet og kaller på neste middleware. Den siste middleware-funksjonen man treffer på er som regel for håndteringen av selve responsen. Denne tar kun inn request og response-objektet og håndterer forespørselen. Her formes res-objektet basert på forespørselens gyldighet. Normalt sett vil denne returnere en HTTP 200-status med JSON-data.

5.2.2 Database

Gruppen valgte å bruke en dokumentorientert database. Valget falt her på [MongoDB](#), det mest populære dokumentorienterte databasehåndteringssystemet(DBMS). Bakgrunnen for valget faller blant annet på at gruppen hadde en del erfaring med relasjonsdatabaser fra tidligere, og ønsket derfor å få erfaring med dokumentorienterte databaser. I tillegg ønsket vi muligheten til å lagre ustrukturert data, som “details” feltet i vår produktkolleksjon. Det er en uspesifisert mengde med nøkkel-verdi par som varierer fra dokument til dokument. Dette er kun mulig med en dokumentorientert databasemodell, da relasjonelle databaser ikke tillater arrayer og objekter i et felt.

Dokumentene i MongoDB er på et format som ligner på JSON. I tillegg til vanlige create, read, update, delete(CRUD) operasjoner, så har MongoDB andre fordeler [21]:

- Dokumentmodellen tilsvarer javascript-objektene i koden vår, hvilket gjør det svært enkelt å behandle data.
- Horizontal skalering av databasen, ved bruk av sharding.
- Det er open-source og gratis.

MongoDB definerer to designmønstre som designeren av en database bør ta hensyn til når det kommer til relasjon mellom dokumenter. **Embedded Documents** og **Reference Documents**. Embedded documents er en av de styrkene til MongoDB, sammenlignet med tradisjonelle relasjonsdatabaser.

Embedded documents

Under følger et eksempel av et dokument i en kolleksjon med personer. Under telefon-attributten er det en liste med telefonnumre. Ved å gjøre et søk på en person, får du telefonnumre til personen i en og samme forespørsel til databasen.

```
// Kolleksjon: Personer
{
  _id: "507f1f77bcf86cd799439011",
  navn: "Geir Halvorsen",
  telefon: [
    {
      type: "mobil",
      nummer: "98765432"
    },
    {
      type: "jobb",
      nummer: "45454545"
    }
  ]
}
```

Reference documents

Reference documents minner mer om hvordan en ville løst lignende problemstillinger i en relasjonell database, hvor det ikke er mulig å ha variabel struktur på formen i eksempelet over. I eksempelet under ser du den samme informasjonen fordelt utover tre dokumenter i to kolleksjoner. For å hente en person med tilhørende telefonnummer, må det gjøres to forespørsler til databasen, en for å hente personinfo, og en for å hente telefonnummer med tilhørende person_id.

```
// Kolleksjon: Personer
{
    _id: "GH1",
    navn: "Geir Halvorsen",
}

// Kolleksjon: Telefoner
{
    {
        person_id: "GH1",
        type: "mobil",
        nummer: "98765432"
    },
    {
        person_id: "GH1",
        type: "jobb",
        nummer: "45454545"
    }
}
```

Valg av Embedded vs. Reference documents

For å velge mellom Embedded og Reference documents, er det hensiktsmessig å ta stilling til hvordan dataene skal brukes. Dersom man i frontend skal vise telefonnumre til en person hver gang dokumentet hentes fra databasen, er det hensiktsmessig å ha denne innebygd i person-kolleksjonen for å minimere antall forespørsler. Dersom man sjeldent får bruk for dataene, eller at antall telefonnumre er stort, kan det være hensiktsmessig å bruke Reference documents isteden.

5.3 Sikkerhet

5.3.1 JSON Web Tokens for autentisering og autorisering

For å kunne autentisere brukere har gruppen valgt å benytte seg av JWT (JSON Web Tokens). Dette er et token som generes i backend etter at en bruker har logget seg inn. Tokenet inneholder brukerinfo og sendes til klienten som har logget seg inn. Den autentiserte klienten vil sende dette tokenet sammen med alle etterfølgende forespørsler til serveren. Tokenet vil unikt identifisere brukeren som sender forespørselen til serveren. JWT er ikke kryptert, bare encoded, slik at hvem som helst kan lese hva den inneholder og endre på innholdet. For å sørge for integritet, inneholder også tokenet en digital signatur som består av en hashing og kryptering av payloaden (innholdet)

i tokenet. Kryptering er gjort på serversiden med en privat nøkkel, derfor er det bare serveren som kan generere en gyldig digital signatur.

Vi bruker npm-pakken [JsonWebToken](#) for å generere, lese og validere JWT.

5.3.2 Bcrypt for hashing og salting av passord

Bcrypt er en npm pakke som tilbyr hashing. Passordene må hashes for å forhindre at en angriper som får tilgang til databasen også får tilgang til alle passord som er lagret der. Dette i seg selv er ikke tilstrekkelig, da en angriper kan utføre et såkalt regnbuetabellangrep [34]. Som mottiltak til dette har bcrypt en saltingfunksjon, slik at hvert hashet passord er unikt. Bcrypt hashingen er også fremtidsrettet da funksjonens treghet er parameterisert. Dette gjør den resistent mot brute force angrep. Mens prosesseringshastigheten på pcer øker(Moore's law), så kan man i samsvar øke tregheten på algoritmen [6].

Kapittel 6

Utviklingsprosessen

Dette kapittelet gjør rede for arbeidet som har blitt utført, i en kronologisk rekkefølge

6.1 Sprint 0 - Oppsett og arkitektur

Det første sprintet ble satt igang etter at gruppen var ferdige med utvikling av krav (kapittel 3.4). Sprintet hadde en varighet på to uker, hvor målet var å sette opp overordnet arkitektur for prosjektet og planlegge arbeidsprosessen.

6.1.1 Brukerhistorier

I begynnelsen av sprintet ble det skrevet brukerhistorier ut ifra kravspesifikasjonen. Dette gjorde at gruppen var mer rustet til første utviklingssprint (sprint 1).

6.1.2 Databasearkitektur

Det ble laget et utkast av arkitekturen til databasen. Her ble det erfart at det å lage ER-modeller av databasen, som er en teknikk som tradisjonelt blir brukt for å designe relasjonsdatabaser(kapittel 4.2.1), ikke er like hensiktsmessig når en jobber med dokumentdatabaser, da det ikke finnes gode verktøy med støtte for dette. Dette blir spesielt tydelig når en velger å bruke embedded struktur i et dokument som diskutert i kapittel 5.2.2. Arkitekturen ble derfor planlagt ved å opprette modeller i mongoose, som er et objektmodelleringstverktøy for MongoDB i Node.js.

Under diskusjon med veiledere hos Accenture kom det fram at applikasjonen kan komme til å brukes til mer enn bare uteleie av spill og brettspill. Accenture har andre grupper som også kan være interessert i å benytte seg av appen. Dette tok gruppen til seg, og designet av databasen ble derfor generalisert for å gjelde generell uteleie av produkter.

6.1.3 Skjelett for applikasjonen

Det ble satt opp to separate repositories i GitHub for utvikling av prosjektets frontend og backend. Her satt vi opp en hensiktsmessig mappestruktur for koden og opprettet et kjørbart backend-API og frontend.

Kjernen i applikasjonen er produktene som skal kunne lånes. Dermed ble det naturlig å starte utviklingen med endepunkter i backend. Vi opprettet et produkt-endepunkt, med funksjonalitet for oppretting av nye produkter, henting, redigering og sletting. Integrasjonstester for endepunktet ble utviklet parallelt. For å gjøre videreutvikling av applikasjonen enklere, ble det også skrevet dokumentasjon av endepunktet i swagger.

6.1.4 Rapportskriving

I dette sprintet ble det også brukt en del tid på å skrive på sluttrapporten for oppgaven, noe gruppen hadde satt seg som mål å jobbe med kontinuerlig under prosjektet.

6.1.5 Sprint review

I slutten av sprintet ble det avholdt et møte med veileder og produkteier fra Accenture for å vise frem arbeidet så langt. Det ble lagt vekt på å verifisere at databasearkitekturen sto i tråd med applikasjonens funksjonalitet. Det ble her verifisert at vi kunne fortsette, da produkteier var fornøyd med løsningen.

6.1.6 Sprint retrospektiv

Et retrospektiv-møte ble avholdt mellom gruppemedlemmene etter dette for å se hva vi gjorde bra og hva vi kan forbedre. Noen punkter fra dette møtet:

Positive punkter:

- Vi har skrevet mye på sluttrapporten allerede
- Vi har allerede grunnleggende arkitektur oppe og et endepunkt med tester.
- API-dokumentasjonen (Swagger) synes å være veldig bruktbart for videre utvikling.
- Medlemmene tar selv initiativ til arbeid.
- Medlemmene er fornøyd med å ha tatt opplæring først.

Forbedringspotensiale:

- Vi opplevde at arbeidsoppgaver i Trello ikke ble oppdatert ofte nok slik at vi ikke hadde

oversikt over hvem som gjorde hva. Vi ble enige om å bli flinkere på dette.

- Github: Gruppére relaterte commits og commite oftere slik at historikken blir ryddigere.
- Mye tid går på lunsj, her kuttes det ned fremover.
- Vi opplevde å bli forstyrret av hverandre ofte med spørsmål. Tiltaket her ble at alle gjør et reelt forsøk på å løse et problem alene før man spør.
- Frontend design: Mer utfordrende enn antatt. Her måtte vi lære oss Bootstrap-grid litt grundigere og tegne mer for hånd.

Etter dette møtet opplevde gruppen raskt bedring på flere av punktene.

6.2 Sprint 1 - Produktoversikt, registrering og innlogging

Sprint 1 er det første utviklingssprintet. Her ble mye av hovedstrukturen for applikasjonen utviklet. Da endepunktet for produkter allerede var utviklet i forrige sprint, kunne vi enkelt starte å vise produkter i frontend. Et sprint-planning-møte ble avholdt med produkteier og veileder fra Accenture, hvor gruppen kom fram til sprintmål og brukerhistorier som skal utvikles.

6.2.1 Sprintmål:

“Implementere oversikt over produkter, søk, samt registrering og innlogging.”

6.2.2 Brukerhistorier

Under dette sprintet har gruppen satt seg som mål å implementere brukerhistoriene listet under. Ved hjelp av [planning poker](#) diskuterte vi antatt kompleksitet på oppgavene. Det ble observert at vi kan ha overvurdert tiden vi har og tatt med for mange brukerhistorier i sprintet. Likevel var det enighet om at det var bedre med for mange oppgaver enn for få. Brukerhistoriene er listet opp i synkende prioritert rekkefølge.

Brukertilfelle	kompleksitet
Lage endepunktet for kategorier (categories)	8
Som bruker ønsker jeg oversikt over inventaret slik at jeg letttere kan finne fram til det jeg ønsker å låne.	13
Som bruker ønsker jeg å kunne søke gjennom inventaret for å utforske hva jeg kan låne.	13
Lage endepunktet for brukere (users)	13
Lage endepunktet for autentisering (auth)	20

Som en ansatt hos Accenture ønsker jeg å kunne registrere meg som bruker på web-appen for å benytte tjenesten.	8
Som ansatt hos accenture ønsker jeg å bekrefte identiteten min via Accenture mailen min for å fullføre registreringen.	40
Som en registrert bruker ønsker jeg å kunne logge meg inn på web-appen for å benytte tjenesten.	20

6.2.3 Backend

I backend ble det laget endepunkter for brukere, autentisering og kategorier.

Implementasjonen av kategori-endepunktet fikk betydning for produkt-endepunktet fra forrige sprint. Her måtte vi gjøre endringer slik at produkt-endepunktet sjekker kategori-ID og fyller inn kategorinavnet fra databasen ved opprettelse av nye produkter.

Endepunktet for brukere tar for seg registrering av nye brukere og sender en epost til brukeren for verifisering av epost. Brukeren kan ikke logge inn før mailen har blitt verifisert. Det ble også implementert en konfigurasjonsvariabel slik at en kan skru av og på om verifisering trengs for innlogging. Dette ga muligheten til å skru av sikkerhetsfunksjonaliteter under utvikling av applikasjonen.

Endepunktet for autentisering håndterer innlogging. Ved suksessfull innlogging sendes det respons med et signert JWT-token med brukerID og navnet på brukeren. Tokenet må sendes til backend med hver forespørsel til beskyttede endepunkter.

Det ble laget flere gjenbrukbare middleware (se kapittel 5.2.1). Et av disse er kalt “auth”. Denne brukes som middleware for hvert endepunkt som krever autentiseringstoken med forespørselen, altså at brukeren skal være innlogget. Middlewaret stopper forespørselen og sender relevant HTTP-feilmelding ved feil.

6.2.4 Frontend

I frontend ble det laget en oversikt over produkter, registreringsside, samt innlogging og utloggingsfunksjonalitet.

Oversikten over produkter ble implementert med søkefunksjonalitet, kategorifiltrering og sortering. Her opplevde vi at kategorifiltreringen var mer kompleks enn forventet. Grunnen til dette er den valgte databasearkitekturen for underkategorier. Ett produkt-dokument har kun ett kategorifelt. Dersom valgt filtreringskategori er en hovedkategori, må også alle produkter som har underkategori tilhørende denne hovedkategorien vises. Dette ble til slutt løst i frontend fremfor å endre på arkitekturen for produkt-dokumentet.

Registreringssiden for nye brukere ble implementert ved å lage en gjenbrukbar klasse for skjemaer. Komponenten har også innebygget funksjonalitet for validering av inputfelt og visning av feilmeldinger. Ved vellykket registrering får brukeren beskjed om at mail har blitt sendt til oppgitt epost-addresse. Brukeren kan logge seg inn etter at han har trykket på lenken i eposten.

Innlogging og utloggingsfunksjonalitet ble implementert ved å håndtere JWT-token som serveren sender ved korrekte innloggingsdetaljer. Tokenet blir lagret som en cookie ved innlogging og slettet ved utlogging. Ved hver forespørsel til backend inkluderes cookien med forespørselen for autentisering.

6.2.5 Coronavirus

I begynnelsen av mars fikk gruppen mail fra Accenture om at vi ikke kunne være på kontorene lenger grunnet spredningen av [SARS-CoV-2 viruset](#) og anbefalning fra myndighetene om å holde seg hjemme. Lokalene ved OsloMet ble også stengt for å bremse spredningen.

Da det ikke var mulig å sitte på Accenture eller OsloMet i tillegg til at helsemyndighetene anbefalte hjemmekontor, måtte gruppen finne alternative måter å kommunisere på. Tidligere ble det benyttet [Slack](#) for tekstlig kommunikasjon, deling av nyttige linker og planlegging av møter med veiledere. Slack ble hyppigere brukt ved hjemmekontor, men gruppen trengte også å kommunisere via tale og dele skjerm for å blant annet vise kode. For kommunikasjon via tale falt valget på [Discord](#). Til å dele skjerm ble [TeamViewer](#) benyttet. Felles for disse er at de er enkle å sette opp og har mange funksjoner uten at man må betale for disse.

Disse verktøyene ble hyppig brukt og sikret prosjektets fremgang. Daily standups ble utført via discord hvor hvert gruppemedlem forklarte hva som hadde blitt gjort, hva som skulle gjøres idag og eventuelle utfordringer som kunne oppstå. For å støtte opp under dette var det hjelpsomt å benytte TeamViewer for å kunne vise fremskrift, parkode, planlegge sprint og gjennomføre planning poker.

6.2.6 Sprint review

Et sprint-review møte ble avholdt med produkteier og veileder etter sprintenes slutt. Produkteier var stort sett fornøyd med løsningen vår hittil. Designet var bra, enkelt og minimalistisk.

Vi fikk kommentar om at det kan være nyttig å endre til rollebasert tilgangsstyring istedenfor bruk av en isAdmin variabel, slik at flere roller kan legges til senere. I tillegg fikk vi tilbakemelding vedrørende manglende feilmelding på innloggingssiden når en skriver feil passord.

Hittil har vi brukt egen epostserver til å sende verifikasjonsepst til brukeren. Produkteier ba oss ta kontakt med Kjell-Olav for å høre om vi kan få tilgang til en egen mailserver for appen.

6.2.7 Sprint retrospektiv

Under retrospektiv møte etter sprintets slutt diskuterte vi hvordan sprintet hadde gått. På den positive siden var gruppen fornøyd med at vi fikk fullført alle brukerhistoriene som var planlagt å fullføres. Vi har jobbet både sammen som et team og vært selvstendige der det trengs. Overgangen til hjemmekontor kom brått på, men vi klarte raskt å justere oss til dette med bruk av riktige verktøy. Ellers var vi enige i at Swagger-dokumentasjonen var hendig å ha under utvikling i frontend, samt at det å ha skrevet tester gjør det enklere å refaktorere kode uten å ødelegge funksjonalitet.

Under forbedringspunkter har vi blant annet forbedringspotensiale på rapportskriving. Her må vi skrive rapport litt oftere for å komme oss i mål. Mer konkret skal vi lese mer gjennom hverandres arbeid og kommentere oftere, samt at det skal være lav terskel for å redigere andres tekst hvis vi finner feil.

Ellers var gruppen enige om å planlegge funksjonalitet mer i fellesskap før vi begynner å kode, slik at vi minsker sannsynligheten for designfeil.

6.3 Sprint 2 - Adminfunksjonalitet og utlån

Sprint 2 er det andre utviklingssprintet. Etter dette sprintet var det planlagt at en MVP av appen skulle være klar. Under sprint-planning møte fikk vi avklart at utlån og tilbakelevering vil kunne foregå på forskjellige måter, slik at produkteier ønsket å ha med egne felt for hente- og returninstrukser.

6.3.1 Sprintmål

”Implementere funksjonalitet for utlån og tilbakelevering på admin og brukersiden, samt funksjonalitet for å administrere inventaret”.

6.3.2 Brukerhistorier

Følgende brukerhistorier ble planlagt fullført i løpet av sprintet. I tillegg til disse var det planlagt mer rapportskriving og noen bugfikser relatert til forrige sprint.

Brukerhistorie	kompleksitet
Som administrator ønsker jeg å kunne logge meg inn i wep-appen for å administrere tjenesten.	5
Som administrator ønsker jeg å kunne legge til eller fjerne inventar for å vise rett beholdning av inventar.	20
Endepunkt for utlån	20

Som bruker ønsker jeg å sende sende forespørsel om å låne utstyr, slik at jeg kan låne utstyr.	13
Som bruker ønsker jeg å få en oversikt over mine utlån for å holde styr på utlånen.	8
Som bruker ønsker jeg å markere utstyr som returnert slik at produktet blir tilgjengelig for andre.	8
Som administrator ønsker jeg å få oversikt over utlånsforespørslar for å best mulig administrere utlånstjenesten.	5
Som administrator ønsker jeg å kunne godkjenne eller avslå forespørslar om utlån av utstyr for å skape gode brukeropplevelser.	13
Som administrator ønsker jeg å få oversikt over aktive utlån for å best mulig administrere utlånstjenesten.	3
Som administrator ønsker jeg å kunne behandle retur av produkter slik at de blir tilgjengelige for utlån for andre.	8
Som bruker ønsker jeg å få påminnelse om kommende innleveringsfrist slik at jeg husker å returnere det jeg har lånt i tide.	13

6.3.3 Sprint review

Et møte ble holdt med veilederne for å gå over sprintets resultater. Iløpet av disse ukene fikk gruppen laget hovedfunksjonaliteten i applikasjonen. For å implementere utelefunksjonaliteten, benyttet gruppen seg av sekvensdiagrammer for å definere de forskjellige livssyklusene et utlån kan være i. Diagrammene finnes i vedlegg J.

Alle brukerhistoriene ovenfor ble fullført. Gruppen fikk tilbakemelding på at applikasjonen virket bra og som tiltenkt og produkteier syns vi har løst problemene på tilfredsstillende vis.

Gruppen fikk noen tilbakemeldinger på design, da dette ikke har blitt brukt mye tid på. Det ble kommet til enighet om å prioritere funksjonalitet fremfor design, men at vi burde høre med intern veileder hva som er prioritert. Det ble også avgjort at applikasjonen kan hete "BorrowMyTech", da dette samsvarer med navngivning av applikasjoner Accenture har fra før.

6.3.4 Sprint retrospektiv

Gruppen er fornøyde med å ha fullført mesteparten av punktene som ble satt opp til MVP (kapittel 3.3). Alle punktene bortsett fra punktet: "Brukere kan sette seg på venteliste for utlån" ble utført. Under sprint review ble det avdekket at denne funksjonaliteten ikke trengs å prioriteres.

Det har blitt skrevet en del på produktdokumentasjonen allerede, og vi føler vi ligger godt an her.

Gruppen er enige om at det har vært utfordrende med påtvunget hjemmekonto grunnet koronavirus-situasjonen. Det har vist seg å være mer utfordrende å holde konsentrasjonen oppe utover arbeidsdagen når vi ikke er fysisk i samme rom og motiverer hverandre. Vi har brukt parprogrammering i mindre grad enn tidligere, og det har blitt vanskeligere å skrive konsekvent og strukturert kode, spesielt på frontend. Dette står i kontrast til hvordan det ble jobbet da vi satt hos Accenture, da vi samarbeidet i større grad om kodingen.

Det ble kommet til enighet om å ha flere møter utover dagen som et tiltak for å føle mer ansvar. Møteansvarlig tar større ansvar for å ha agenda klart til møtet, og alle har ansvar for å se hvordan vi har løst ting tidligere for å prøve å være mer konsekvente med koden.

6.4 Sprint 3 - Siste utviklingssprint

Sprint 3 er det tredje og siste utviklingssprintet. Sprint 2 ble forskjøvet med litt over en uke, da det ble kommet til enighet om å ta påskeferie. Dette gjør at gruppen velger å kutte ut det siste planlagte designsprintet på 2 uker, slik at tiden kan brukes på skriving av rapport.

6.4.1 Sprintmål

”Implementere funksjonalitet for administrering av brukerinfo, brukerroller og en forslagsboks til nytt utstyr”

6.4.2 Brukerhistorier

Et sprint-planning møte ble avholdt med produkteier hvor funksjonalitetene for siste sprintet ble planlagt. Under dette sprintet blir det behov for mer tid til rapportskriving, i tillegg til at to av gruppemedlemmene har et prosjekt i et annet fag. Dermed blir det tatt med færre brukerhistorier enn tidligere i sprint-planen.

Brukerhistorie	kompleksitet
Som registrert bruker ønsker jeg å kunne bytte passord for å sikre kontoen.	8
Som registrert bruker ønsker jeg å kunne redigere min profil.	5
Som en bruker av web-appen ønsker jeg å få tilsendt ved behov en epost med link til å opprette nytt passord slik at jeg får tilgang om jeg har glemt passordet mitt.	13
Som administrator ønsker jeg å søke blant eksisterende brukere.	5
Som administrator kan jeg legge til nye administratorer slik at de også kan få samme rettigheter som meg.	8
Som administrator kan jeg deaktivere kontoen til registrerte brukere slik at de ikke kan logge seg inn lenger.	5

Som administrator kan jeg legge inn driftsmelding som dukker opp på toppen av applikasjonen	5
Som bruker vil jeg sende inn forslag om utstyr spillgruppen skal kjøpe inn, slik at jeg kan få tilgang til utstyr jeg har lyst på.	8
Som administrator ønsker jeg å kunne slette forslag om nytt utstyr ettersom jeg har behandlet ønskene slik at de ikke forblir i oversikten.	8

6.4.3 Sprint review

Det ble avholdt et sprint review-møte med veiledere ved sprintets slutt. Veilederne hadde denne gangen utført en brukertest som førte dem gjennom alle applikasjonens funksjonaliteter. Gruppen mottok gode tilbakemeldinger på de nye funksjonalitetene. Blant annet mente produkteier at det var svært intuitivt å legge til nye administratorer og å deaktivere brukere, og at administrator-funksjonaliteten generelt var godt utviklet. Han følte ikke behovet for en brukerveiledning for å skjonne hvordan applikasjonen brukes.

Brukerhistorien om å søke blant eksisterende brukere ble strøket vekk. Her ble det observert at det var nok å ha sortering etter alle kolonnene i brukeroversikten. Produkteier var delvis enig i dette, men poengterte at det kan være greit å ha når antall brukere av applikasjonen vokser.

6.4.4 Sprint retrospektiv

Under retrospektiv-møtet diskuterte gruppen hvordan sprintet hadde gått. Dette er siste sprintet som vil utføres, da det siste planlagte designsprintet utgår grunnet manglende tid.

Gruppen er enige om at det var et godt valg å inkludere færre oppgaver i siste sprint, slik at vi fikk mer tid til å skrive på rapporten. De planlagte brukerhistoriene ble godt utført, og en god del bugs ble fikset slik at applikasjonen ble klar for produksjon.

På den negative siden opplevde gruppen at arbeidet falt litt ut av struktur mot slutten av sprintet. Sprintet ble dratt utover i fjerde uke da vi ikke avholdt sprint-review-møtet på sluttdatoen. En del arbeid som ikke sto på sprint-backlogen, som fiksing av bugs og restrukturering av kode, ble utført i fjerde uke. Dette gjorde at gruppen mistet noe oversikt over hva alle jobbet med. En bedre løsning ville vært å avholde sprint-review-møtet på den siste planlagte dagen i sprintet og hatt et eget sprint for fiksing av bugs. Dette står mer i tråd med hvordan dette gjøres i Scrum.

Motivasjon og disiplin til å arbeide effektivt var fortsatt utfordrende under dette sprintet grunnet situasjonen rundt coronavirus og hjemmekontor. Et mulig tiltak her er å dele opp sine arbeidsoppgaver i mindre punkter og å sette seg et mål om hva en skal være ferdig med i løpet av arbeidsdagen,

da det gir mer motivasjon å krysse av små arbeidsoppgaver utover arbeidsdagen.

6.4.5 Brukertest

En spørreundersøkelse ble tatt i forbindelse med en brukertest og markerte avslutningen på sprint 3. Undersøkelsen hadde to deltagere, hvor den ene er produkteier og den andre veileder. Det er verdt å nevne at spørreundersøkelsen ikke er ideell, da deltagerne allerede kjenner til produktet. En optimal brukertest ville inkludert flere testere med forskjellig kunnskapsnivå og kjennskap til produktet. Resultatene fra undersøkelsen finnes i vedlegg [H](#).

I resultatene av undersøkelsen ble det avdekket at det var en generell tilfredshet med funksjonaliteten til webapplikasjonen. Kommentarene til forbedring hadde hovedsaklig designmessig tema. En bruker kommenterte fargevalget: "Noen fargevalg hadde nok gjort siden enda lettere å lese, ettersom nå er det mye hvitt på hvitt.". Dette understrekkes av at en annen bruker svarte "Det visuelle." på spørsmålet "Hva likte du minst?" Under hypotetisk videre arbeid ville derfor det kosmetiske aspektet være i fokus. Tilbakemeldingene på de funksjonelle aspektene var hovedsakelig positive, men en bruker skriver at fraværet av søkefunksjonalitet i brukerlisten er en mangel. Gruppen valgte å ikke implementere denne brukerhistorien, men er enig at dette er viktig ettersom antall brukere øker.

Videre får gruppen positive tilbakemeldinger om at webapplikasjonen er intuitiv å bruke, og at den fungerer slik som forventet. Applikasjonen blir vurdert som tydelig og gir liten forvirring om hvordan man går frem for å bruke den. Brukerne mente det var rett fram å leie et produkt og at administratorfunksjonaliteten var nyttig og intuitiv. Dette er viktig, da denne funksjonaliteten er kjernen i applikasjonen. På spørsmål om hvordan brukerne vurderer helheten av webapplikasjonen, var gjennomsnittet 9/10. Dette er gruppen godt fornøyd med.

Grunnet at spørreundersøkelsen ble utført sent i prosjektet, har ikke gruppen hatt tid til å utføre endringer i henhold til tilbakemeldingene. I kapittel [15.1](#) diskuteres det hvilke endringer som foreslås basert på tilbakemeldingene.

Del II

Produktdokumentasjon

Innledning

Hensikten til produktdokumentasjonen er i gi leseren teknisk innsikt til webapplikasjonen, både i frontend og backend. For å lettere forstå produktdokumentasjonen anbefales det å lese kapittel 4 og 5 om henholdsvis teoretisk grunnlag og teknologivalg. Målgruppen for denne delen er dataansvarlig som skal installere, forvalte og videreutvikle systemet.

Kapittel 7

Produktbeskrivelse

Accenture har en spillgruppe hvor ansatte kan låne utstyr til sosiale arrangementer. Det har blitt utviklet en webapplikasjon for å tilby en digital løsning for utlån av utstyr. Applikasjonen byr på funksjonaliteter for registrering av brukere, innlogging, produktoversikt, utlån og retur av produkter. Applikasjonen har også et adminpanel, hvor en bruker registrert med administratorrolle kan legge til nye produkter og kategorier, samt godkjenne utlån og returer.

Hensikten med webapplikasjonen er å gjøre det enklere for ansatte å vite hvilke utstyr som er tilgjengelige for utlån, samt å forenkle arbeidet til spillgruppens leder.

Dette kapittelet kan brukes som en bruksmanual for både administratorer og brukere.

7.1 Brukerfunksjonalitet

Brukersiden av applikasjonen er ment for Accenture sine ansatte som ønsker å låne utstyr. I dette delkapittelet følger en beskrivelse av funksjonalitetene som er implementert på brukersiden.

7.1.1 Registrering av brukere

For å registrere en bruker i webapplikasjonen, forutsettes det at brukeren registrerer seg med en e-post med Accenture-domene. Dette var et krav fra oppdragsgiver, da kun ansatte i Accenture skal ha tilgang til applikasjonen.

Figur 7.1 viser registreringskomponenten. Den består av flere felt som lagres i brukerdatabasen. Feltene valideres ved hjelp av valideringsbiblioteket [Joi](#). I figuren ser vi hvordan brukeren får opp en feilmelding hvis dataene ikke tilfredstiller valideringskravene. I tillegg til feilmeldingen blir også “Create User” knappen deaktivert. Tilsvarende validering blir også brukt i backend for å være

The screenshot shows a mobile application interface titled "Loan App". At the top right is a menu icon. Below it, the title "Register here" is displayed in large, bold letters. The form consists of several input fields: "First name" (containing "Ola"), "Last name" (containing "Nordmann"), "Phone" (containing "98765432"), "Email" (containing "ola@accenture.com"), and "Password" (containing "*****"). A red error box is overlaid on the password field, stating: "'Password' length must be at least 8 characters long". At the bottom is a blue "Create User" button.

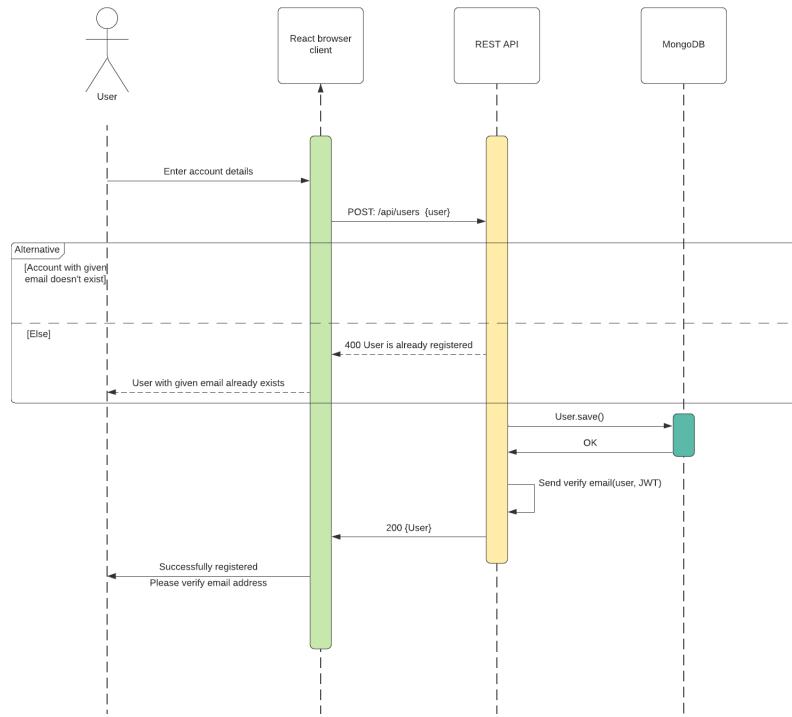
Figur 7.1: Registrering av ny bruker

sikkert på at en bruker som går direkte via det offentlige APIet, ikke kan registrere seg med ugyldig data. Figur 7.2 viser flyten i registreringsfunksjonaliteten.

Passordet som brukeren oppgir lagres ikke i klartekst. Npm biblioteket bcrypt brukes for å hashe og salte passordet. Det er hashen som lagres i databasen.

7.1.2 Verifisering av e-post

E-post-verifisering brukes for å bekrefte at brukeren faktisk har tilgang til den e-postadressen brukeren registrerte seg med. Etter registreringen sendes en e-post til adressen brukeren oppga da den registrerte seg. Mail sendes ved bruk av Node-modulen Nodemailer. E-posten inneholder en link tilbake til webapplikasjonen. Denne linken inneholder også en JWT for å kunne identifisere hvilken bruker det er som verifiserer e-postadressen. Node-modulen `jsonwebtoken` blir brukt for å generere, dekode og verifisere JWT. Når brukeren klikker på linken i e-posten, verifiseres brukeren og kontoen blir aktivert. Etter dette kan brukeren logge seg inn.



Figur 7.2: Sekvensdiagram for registrering

7.1.3 Logg inn

En bruker kan logge inn etter registrering og verifiserering av e-post. Det brukes ikke noen form for validering i input feltene for innlogging. Dersom brukeren enten skriver feil e-post eller passord, vises en generisk feilmelding ”Wrong username/password”. Dette er vist i figur 7.3. I backend brukes bcrypt for å sammenligne det hashede passordet med passordet som brukeren oppgir ved innloggingsforsøk.

Loan App

Log in

Email address

ranytb@gmail.com

Password

.....

Wrong username/password

Login

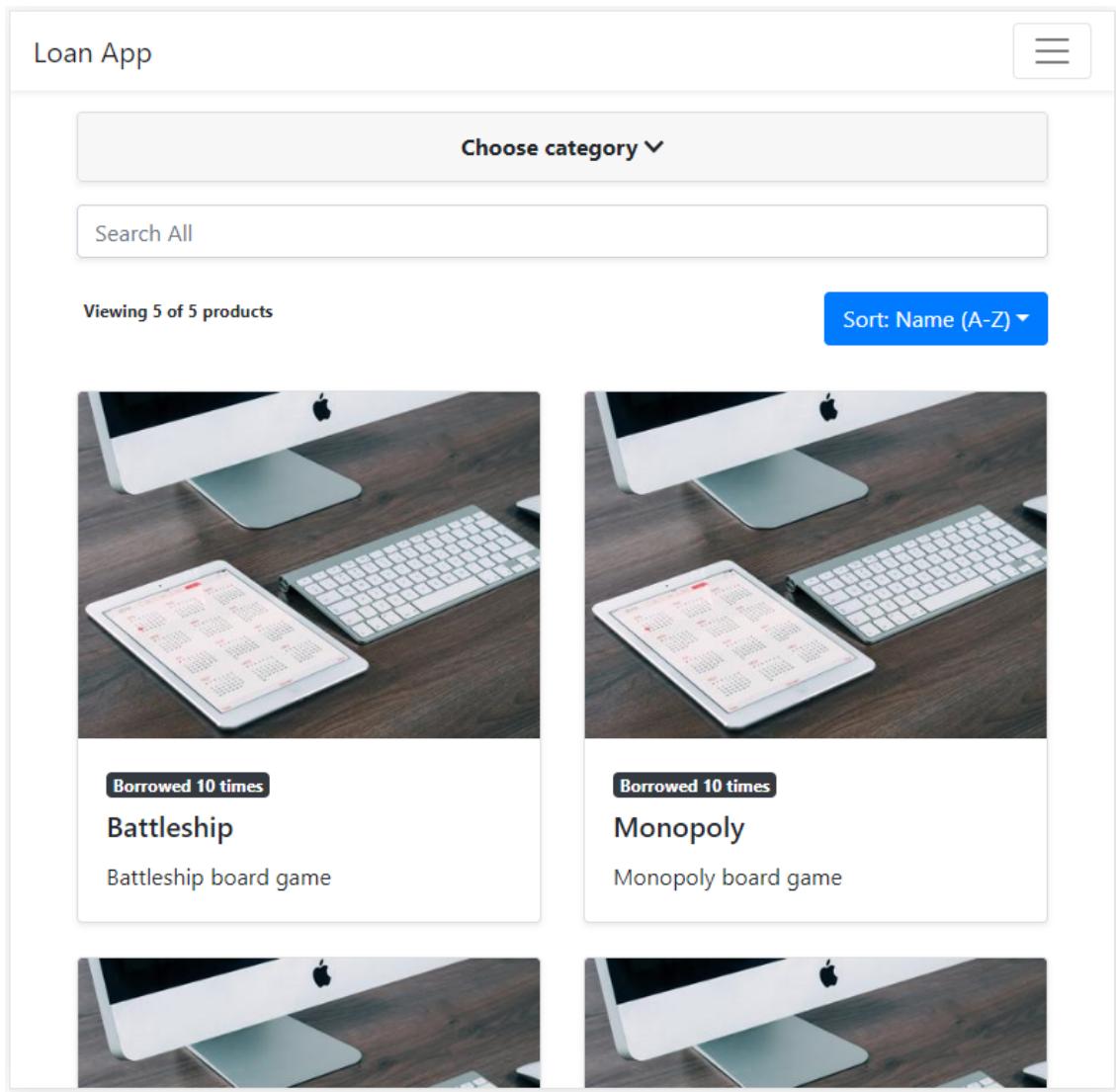
Not registered? [Register here](#)

Forgot password? [Reset here](#)

Figur 7.3: Log in

7.1.4 Oversikt over produkter

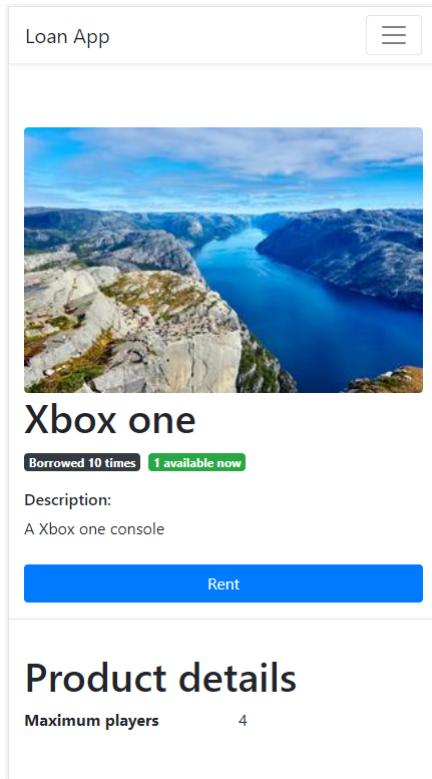
Etter innlogging blir brukeren automatisk navigert til produkt siden. Her får brukeren en oversikt over alle produktene som kan lånes i webapplikasjonen. Figur 7.4 viser oversikten i nettbrettformat. Hvert produkt som vises kan klikkes på og vil navigere deg til en side som inneholder mer informasjon om produktet. Her kan man også velge å låne produktet dersom det er tilgjengelig.



Figur 7.4: Oversikt over produkter

Applikasjonen tilbyr en rekke funksjonaliteter for å søke etter, filtrere og sortere produkter. Søking skjer i søkefeltet. Filtrering får man opp hvis man trykker "choose category". Her kan man velge for eksempel brettspill-kategorien, som resulterer i at kun brettspill vises. Applikasjonen sorterer som standard etter populærheten til hvert produkt, altså hvor mange ganger et produkt har vært lånt ut. Denne sorteringen kan endres ved å klikke på sort-knappen.

Klikker man seg inn på et produkt, vises en beskrivelse av produktet, antall tilgjengelige entiteter for utlån og detaljene til produktet. Dette er vist på mobilformat i figur 7.5. På denne siden så har man også mulighet til å låne produktet.



Figur 7.5: Produktdetaljer

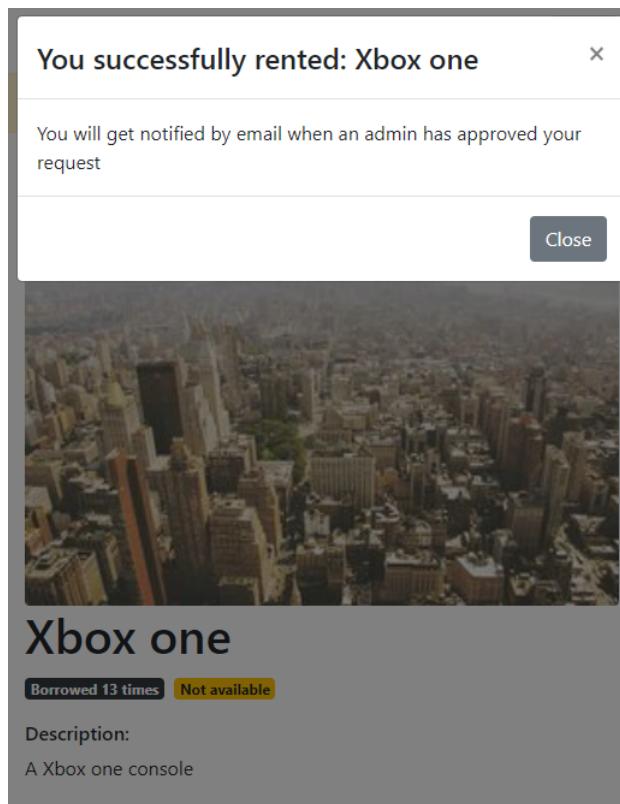
Når brukeren trykker på Rent-knappen, får brukeren beskjed om å vente på godkjenning av administrator som vist i figur 7.6. Alle registrerte administratorer får en e-post om at en bruker har forespurt et utlån. Når en admin har godkjent utlånet, får brukeren en e-post med henteinstrukser. E-posten ser slik ut:

“Hello Ola,

Your rental request for Xbox One has been approved.

Pick up instructions: Pick up at Accenture Fornebu, locker number 23. Passcode 2423.

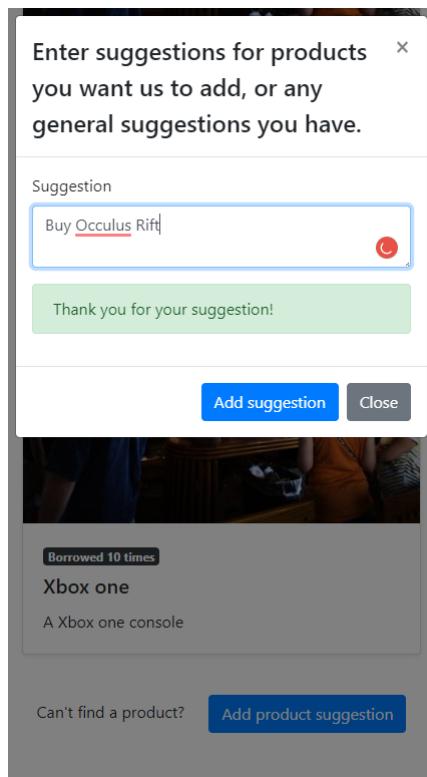
Return instructions: Return to the same locker.”



Figur 7.6: Bekreftelse på utlånsforespørsel

7.1.5 Brukerforslag

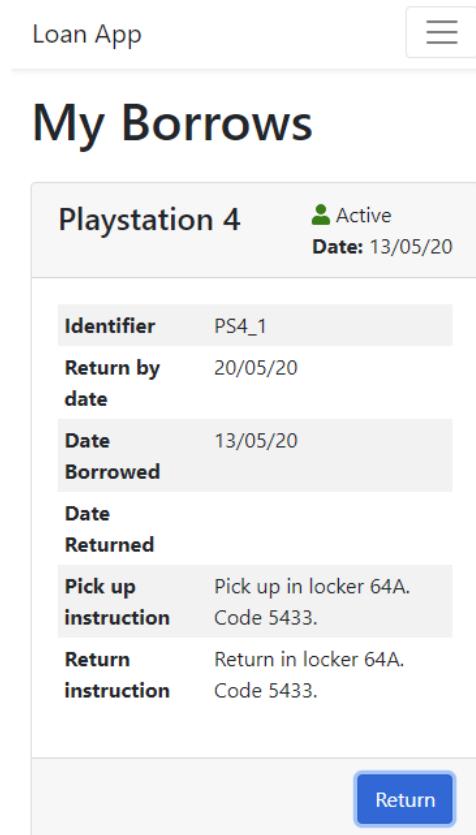
På bunnens av siden med produktoversikten har brukeren mulighet til å trykke på en knapp for å sende inn forslag til produkter som kan kjøpes inn. Når denne knappen trykkes, kommer vinduet vist på figur 7.7 opp.



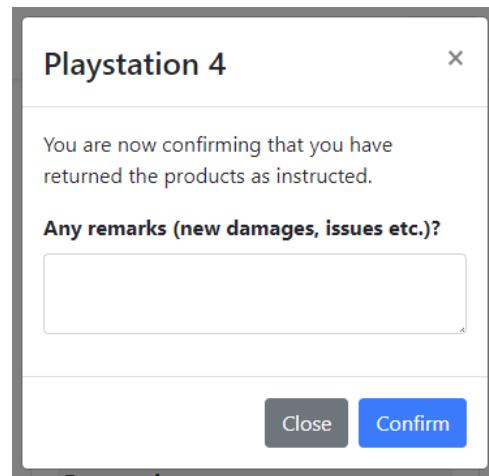
Figur 7.7: Form for å legge til et forslag

7.1.6 Oversikt over utlån

I fanen “My borrows” får brukeren oversikt over sine lån og status på disse. Figur 7.8, i telefonformat, viser hvordan dette ser ut for et aktivt lån. Her kan brukeren også returnere et lån og legge til kommentarer knyttet til returnen. Figur 7.9 viser hvordan en retur-forespørsel ser ut. Når en retur er forespurt, vil utlånet få en returdato i databasen, men variabelen “confirmedReturned” blir ikke satt før administrator godkjenner dette. Mer om livssyklusen til et utlån i kapittel 11.1.



Figur 7.8: My borrows fanen



Figur 7.9: Returforespørrelse

7.1.7 Brukerprofil

Fanen "My profile" tillater brukere å endre brukerinformasjon og å lage nytt passord. Figur 7.10 viser hvordan dette ser ut.

Loan App

My profile

First name
Rany Tarek

Last name
Bouorm

Phone
98866114

Update info Reset password

Figur 7.10: My profile

Når brukeren trykker på reset passord, sendes det en e-post med lenke for å endre passordet. Lenken inneholder et JWT-token som er gyldig i 4 timer. Figur 7.11 viser siden man kommer inn på ved å trykke på denne.

Loan App

Reset password for

New password
Type your new password

Repeat password
Repeat password

Reset password

Figur 7.11: Resetting av passord

7.2 Administratorfunksjonalitet

Administratorfunksjonalitet aksesseres ved å trykke på Admin-fanen i hovedmenyen. Fanen vises kun hvis innlogget bruker er administrator. Dette sjekkes opp mot JWT, hvor alle brukere har et boolean-felt **isAdmin**. Hvis denne er true vises Admin-fanen i hovedmenyen. Dersom en vanlig bruker forsøker å aksessere en adminlenke, eksempelvis /admin/products, vil en få opp feilmeldingen "not found". Figur 7.12 viser hovedsiden for administrator med oversikten over produkter.

Title	Category	Number of loans	Number of entities	Action
Playstation 4	ps4	5	3	View product Delete product
Xbox one	Xbox	6	1	View product Delete product
Nintendo Switch	switch	8	1	View product Delete product
Monopoly	Board Games	15	1	View product Delete product
Battleship	Board Games	4	2	View product Delete product

Figur 7.12: Admin dashboard landindsside og side for administrering av produkter

7.2.1 Administrering av produkter

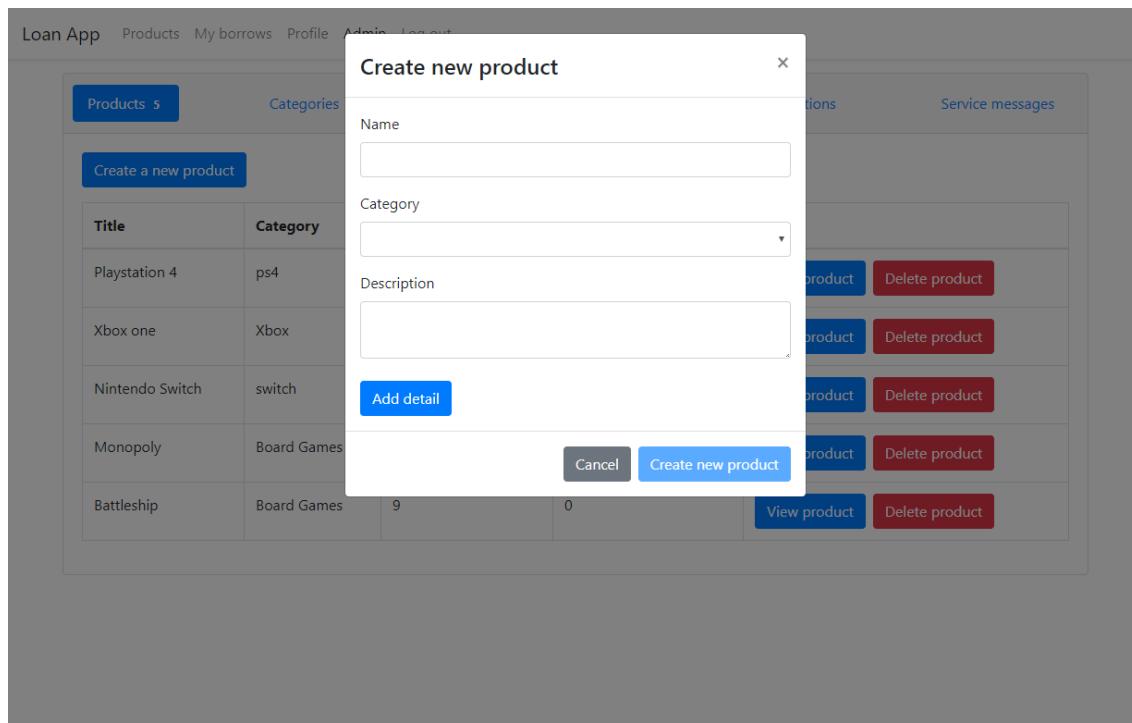
Administrering av produkter gjøres via Products-fanen i admin dashboard. Her vises en tabell med alle produkter, hvilken kategori de tilhører, antall utlån og hvor mange entiteter hvert produkt har. “Create a new product”-knappen fører til et skjema hvor en oppretter et nytt produkt. Se figur 7.13.

Name, Category og Description må fylles ut, mens detaljer er valgfritt. Category er et dropdown-felt som viser kategorier fra databasen. Nye kategorier kan opprettes i Categories-fanen. Ønsker administrator å gjøre endringer på produktet kan dette gjøres ved å klikke på “View product”. Her får en oversikt over produktinfo og entiteter som vist på figur 7.14.

“Modify product” gir administrator muligheten til å gjøre endringer på produktet, eksempelvis redigere beskrivelsen. Det er også mulig å fjerne eller legge til nye detaljer. Se figur 7.15.

For å gjøre et produkt tilgjengelig for utlån må minst én entitet legges til. “Add new entity” viser administrator et skjema som må fylles ut for å lage en ny entitet. Et produkt kan ha ingen eller flere entiteter. Figur 7.16 viser hva som må utfyllses. Available for rental er en dropdown med valg true eller false.

Det kan ha skjedd endringer underveis med entiteten, kanskje den har blitt ødelagt eller det har oppstått feil eller mangler. Da kan administrator gjøre endringer på entiteten ved å trykke på



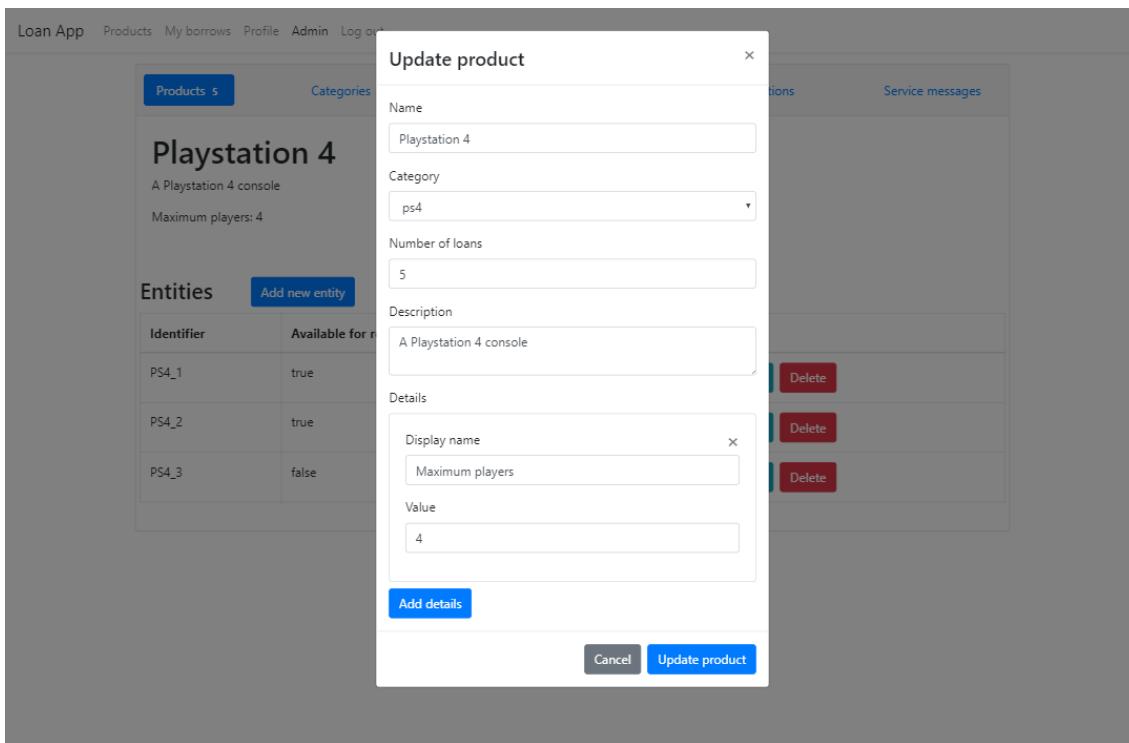
Figur 7.13: Modal for å lage et nytt produkt.

The screenshot shows an admin interface for a product named 'Playstation 4'. It includes a summary section with details like 'A Playstation 4 console' and 'Maximum players: 4'. Below this is a table titled 'Entities' with columns for 'Identifier', 'Available for rental', 'Remarks', and 'Action'.

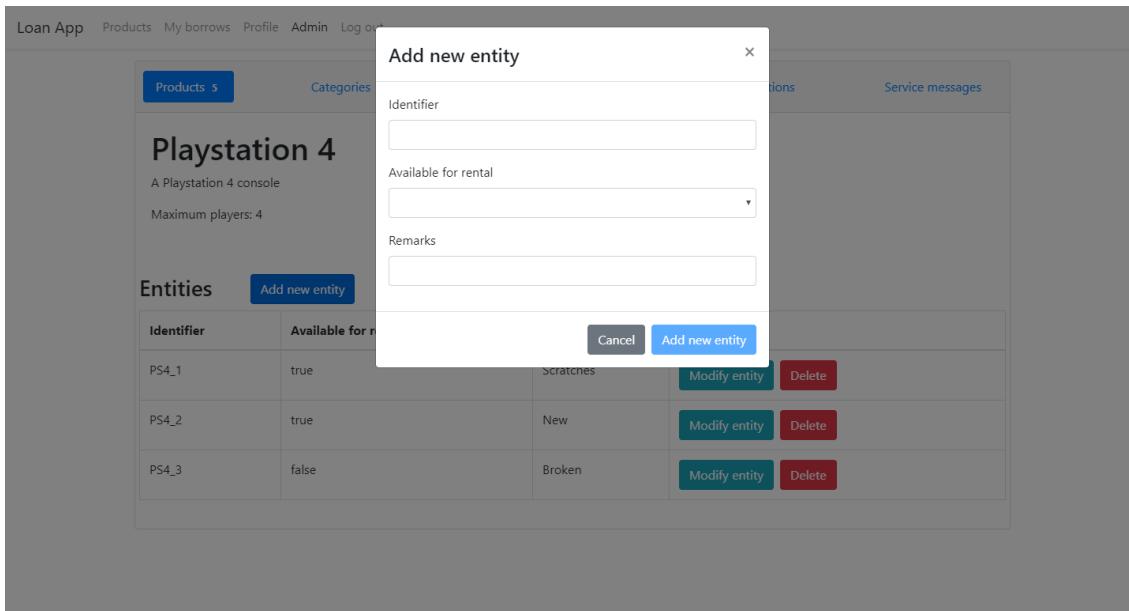
Identifier	Available for rental	Remarks	Action
PS4_1	true	Scratches	Modify entity Delete
PS4_2	true	New	Modify entity Delete
PS4_3	false	Broken	Modify entity Delete

Figur 7.14: Admin oversikt over et produkt og entiteter.

“Modify entity”. Da får administrator opp et vindu med felter som er forhåndsutfyld slik entiteten ser ut i databasen. Disse feltene kan endres på, figur 7.17, og modifikasjoner blir lagret i databasen



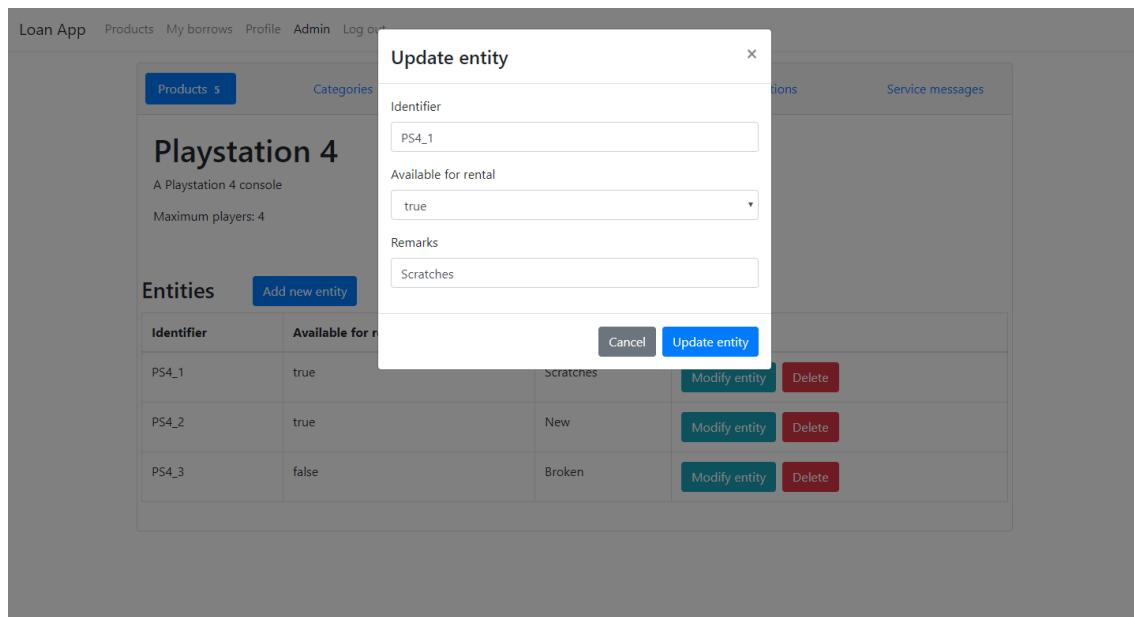
Figur 7.15: Admin modifisering av produkt modal.



Figur 7.16: Admin legge til nye entiteter.

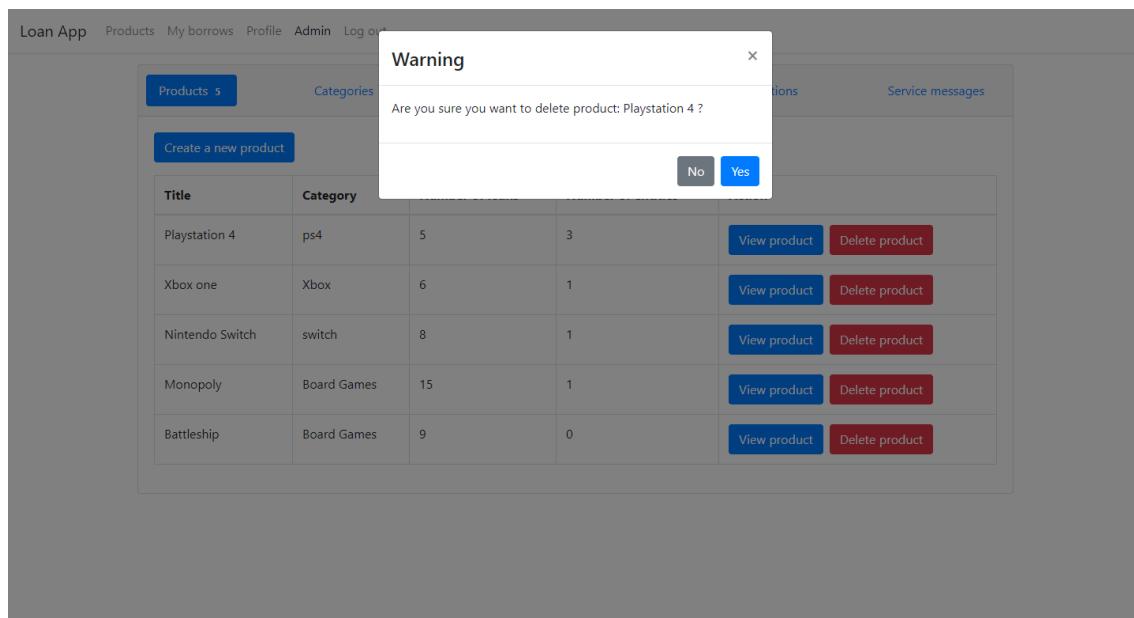
ved at administrator trykker på “Update entity”.

En administrator må også ha muligheten til å slette entiteter. Trykker administrator på “Delete” kommer det en advarsel om at spesifikk entitet blir slettet slik at det er mulig å angre.



Figur 7.17: Modifisering av entitet.

Produkter kan slettes i produktoversikten. Admin blir da presentert for tilsvarende advarsel slik som for sletting av entiteter, nå med advarsel om sletting av produkt (figur 7.18).



Figur 7.18: Advarsel om sletting av produkt.

7.2.2 Administrering av kategorier

Administrering av kategorier gjøres i Categories-fanen. Her får admin en oversikt over alle kategorier og hvilken hovedkategori disse eventuelt hører under (figur 7.19).

Loan App Products My borrows Profile Admin Log out

Name	Parent Category
Gaming Consoles	
Board Games	
ps4	Gaming Consoles
Xbox	Gaming Consoles
switch	Gaming Consoles

Figur 7.19: Oversikt over kategorier.

For å legge til nye kategorier velges "Create a new category". Kun navn er obligatorisk å fylle inn i dette skjemaet, figur 7.20. "Parent category" er en dropdown-liste over hovedkategorier som er valgfritt å legge til, eksempelvis "Gaming Consoles" eller "Board Games".

The dialog box is titled "Create new category". It contains two input fields: "Name" with the placeholder "Enter name" and "Parent category" with the placeholder "None". At the bottom are two buttons: "Cancel" and "Create new category".

Figur 7.20: Admin legge til nye kategorier.

7.2.3 Administrering av utlån

Utlånsadministrering er tilgjengelig via rentals-fanen i admin-dashboardet. På denne siden får man opp tre tabeller. Hver tabell representerer en av livssyklusene til utlånet. Livssyklussene til utlån blir beskrevet i kapittel 11.1. Figur 7.21 viser de tre tabellene.

Loan App Products My borrows Profile Admin Log out

Products 5		Categories 5		Rentals		Users		Suggestions	
<h2>Rentals</h2>									
<input type="checkbox"/> Show requested rentals only									
User	Product	Identifier	Date Requested ▾	Pick Up Instructions	Return Instructions	Date out	Action		
markus	Nintendo Switch	Switch_1	01/05/20	Cafeteria	Office	01/05/20	<button>Confirm Return</button>		
markus	Playstation 4	PS4_4	01/05/20				<button>Manage Rental</button>		
markus	Xbox one	XB1_1	01/05/20				<button>Manage Rental</button>		
markus	Monopoly	MP1	01/05/20				<button>Manage Rental</button>		
<h2>Requested Returns</h2>									
User	Product	Identifier	Date Returned ▾	Remarks From Customer	Confirmed Returned	Action			
markus	Playstation 4	PS4_1	01/05/20		false	<button>Confirm Return</button>			
markus	Playstation 4	PS4_3	01/05/20	Does not work	false	<button>Confirm Return</button>			
<h2>Processed Returns</h2>									
User	Product	Identifier	Date Returned ▾	Remarks From Customer	Confirmed Returned				
markus	Battleship	BS1	01/05/20		true				

Figur 7.21: Administrering av utlån

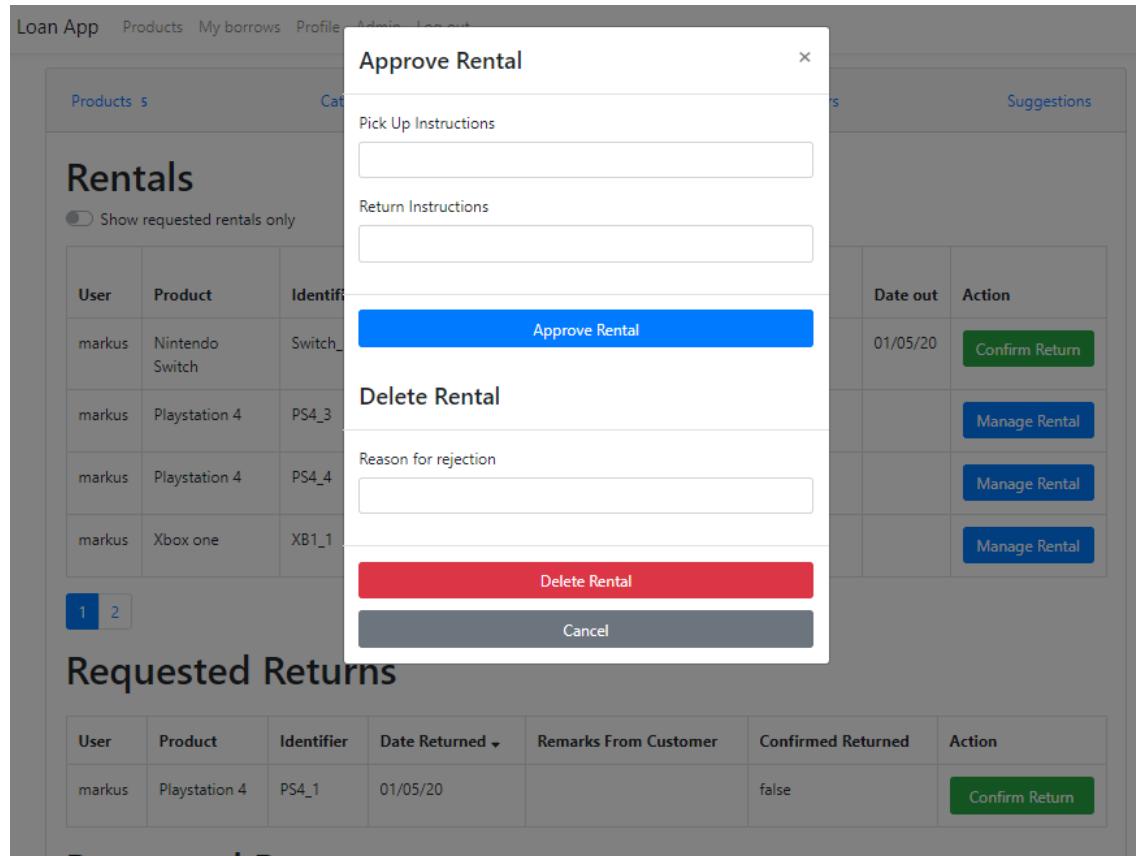
Rentals tabellen

Rentals tabellen viser en oversikt over utlån som har blitt forespurt og aktive utlån. For forespurte utlån vises "Manage Rental"-knappen. Her kan administrator godkjenne eller avslå lånet som vist i figur 7.22. En mail blir sendt til brukeren med hente- og returninstrukser når administrator godkjenner det. En mail blir også sendt ved avslag.

Når administrator godkjenner utlånet blir "Date out" og hente-/returninstrukser satt på utlånet. Slik skiller det mellom låneforespørsler og aktive utlån. Lånet er gyldig i 7 dager. Brukeren får tilsendt mail med påminnelse om retur dagen før innleveringsfristen.

Informasjonen om utlånet forblir i denne tabellen helt til enten admin trykker på confirm return knappen, eller til en bruker har forespurt en retur. Ved bruk av "confirm return" blir utlånet

fullført uten at låneren trenger å gjøre noe i applikasjonen.



Figur 7.22: Behandling av låneforespørrelse

Requested returns-tabellen

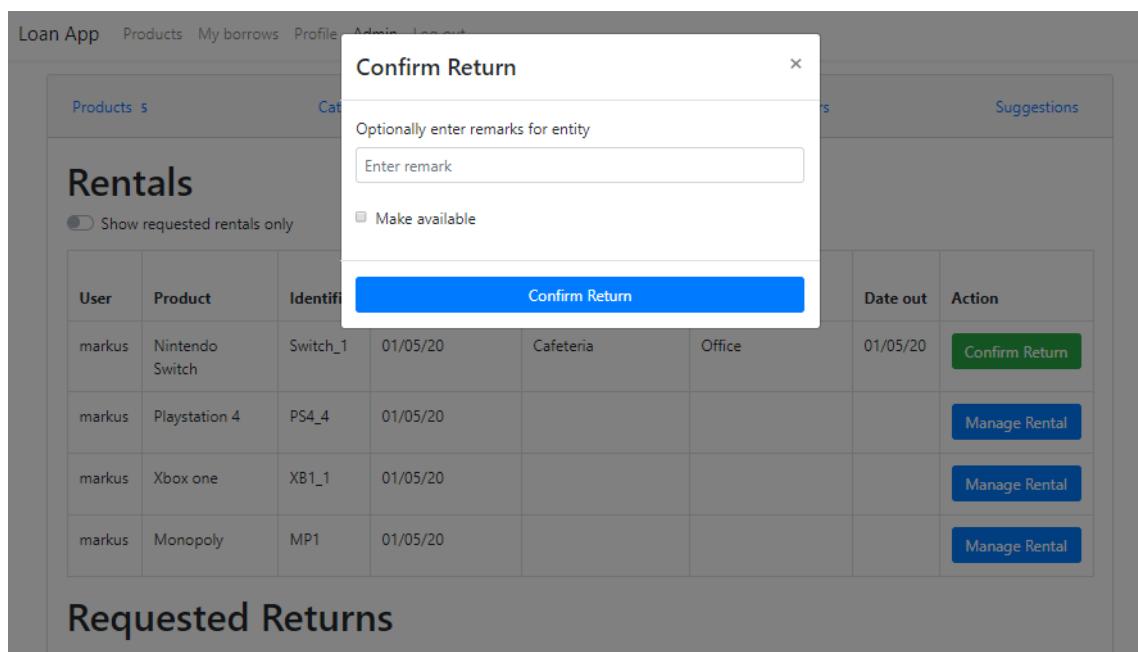
Requested returns-tabellen viser utlån som har blitt markert som returnert av en bruker. Her får administrator også se om brukeren har skrevet noen bemerkninger om produktet. Figur 7.23 viser vinduet som kommer opp når administrator godkjenner et utlån. “Make available” må krysses av for at den utlånte entiteten skal bli tilgjengelig for andre. Admin kan her redigere bemerkninger om entiteten dersom det er noen nye mangler eller lignende.

Processed returns tabellen

Den siste tabellen viser en oversikt over fullførte utlån. Det vil si returer som har blitt godkjent av administrator.

7.2.4 Administrering av brukere

Users-fanen (figur 7.24) i admin dashboard gir en oversikt over alle brukere. Kolonnene kan sorteres. Status-kolonnen viser om en bruker har tilgang til tjenesten. Status blir først aktiv når bruker trykker på bekreftelseslinken som sendes på e-post. Eventuelt kan en administrator gjøre dette



Figur 7.23: Returbekreftelse

manuelt ved å trykke på status-knappen for den gjeldende brukeren. En brukerkonto kan på samme måte deaktivertes eller slettes, eksempelvis dersom en bruker gjentatte ganger ikke har overholdt leveringsfristen for lånenene sine.

Users						
First name ▾	Last name	Phone	Email	Status	Role	Action
John	Doe	90909090	john.doe@accenture.com	<input type="radio"/> Inactive	<input type="radio"/> User	<button>Delete user</button>
Jane	Doe	40404040	jane.doe@accenture.com	<input checked="" type="radio"/> Active	<input checked="" type="radio"/> Admin	<button>Delete user</button>

Figur 7.24: Administrering av brukere.

En administrator kan forfremme andre administratorer i "Role"-fanen. For å endre bruker klikker admin på Role-knappen som skifter mellom admin og vanlig bruker. Brukeren må logge av og på for at endringen skal tre i kraft.

7.2.5 Administrering av forslag

Forslag som er innsendt fra brukere kan administreres av en administrator (se figur 7.25). Administreringssiden for forslag kan navigeres til ved å klikke på suggestions-fanen i admin dashboardet. Siden viser en tabell med forslag. Forslagene er sortert etter dato, men kan også sorteres etter andre kolonner. Administratoren kan slette forslag.

The screenshot shows a web-based admin dashboard for a loan application. At the top, there is a navigation bar with links: 'Loan App', 'Products', 'Categories', 'Rentals', 'Users', 'Suggestions' (which is highlighted in blue), and 'Service messages'. Below the navigation bar is a table titled 'Suggestions'. The table has four columns: 'Suggestion', 'Date', 'User', and 'delete'. There are five rows in the table, each representing a proposal from a user named 'Markus' on May 4, 2020. Each row contains a red 'Delete' button in the 'delete' column.

Suggestion	Date	User	delete
Buy Valve Index	04/05/20	Markus	Delete
Buy Express Monopoly	04/05/20	Markus	Delete
Buy HTC Vive	04/05/20	Markus	Delete
Buy Occulus Quest	04/05/20	Markus	Delete

Figur 7.25: Administrering av forslag

7.2.6 Administrering av driftsmeldinger

I Service messages fanen i admin dashboard kan administrator administrere driftsmeldinger, se figur 7.26.

Her får administrator en oversikt over nåværende driftsmeldinger, muligheten til å legge til nye og slette gamle. Klikkes "Add new service message" blir admin presentert vinduet vist i figur 7.27.

Dersom flere driftsmeldinger legges til til dukker de opp under eksisterende meldinger slik som i figur 7.26. Driftsmeldingene lagres i local storage, noe som gjør at brukere kan krysse ut driftsmeldingen uten å få den opp igjen ved neste innlogging.

The screenshot shows a web interface for managing service messages. At the top, there are two yellow banners: one with general information about COVID-19 and another for the Accenture portal. Below these are navigation links: Products (5), Categories (5), Rentals, Users, Suggestions, and Service messages (highlighted in blue). The main area is titled "Service messages" and contains a table with two rows. Each row has a "Service message" column containing text and an "Action" column with a red "Delete service message" button.

Service message	Action
Questions about corona? Call the information hotline on 815 55 015.	Delete service message
For Accenture's advice on coronavirus check out Accenture portal.	Delete service message

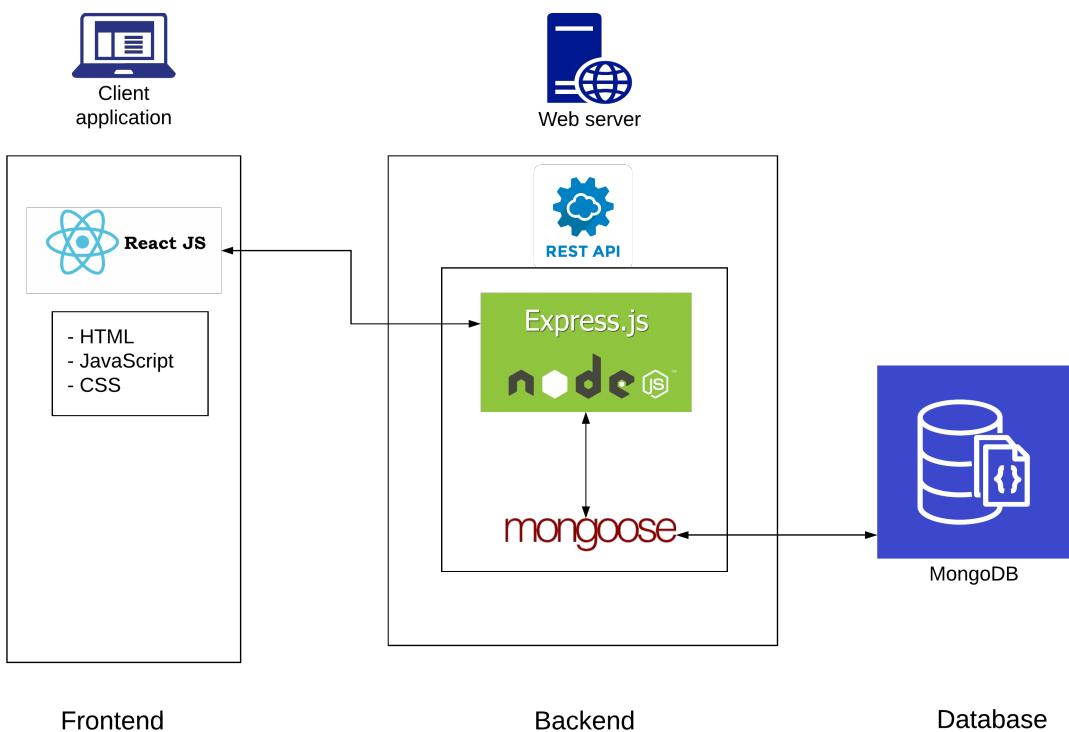
Figur 7.26: Admin driftsmeldinger.

The screenshot shows a modal dialog box titled "Create a new service message". It contains a text input field labeled "Service message" and two buttons at the bottom: "Cancel" and "Create new service message" (highlighted in blue). In the background, the "Service messages" table from Figur 7.26 is visible, showing the same two rows of data.

Figur 7.27: Admin driftsmeldinger.

Kapittel 8

Applikasjonsarkitektur



Figur 8.1: Overordnet arkitektur

8.1 Backend

Backend består av en applikasjon skrevet i Javascript. Applikasjonen kjører på Node.js og tar i bruk Express.js rammeverket. Applikasjonen eksponerer et offentlig API som vår frontend tar i

bruk. APIet kommuniserer med en MongoDB-database ved hjelp av mongoose-biblioteket. Figur 8.1 viser hvordan de ulike systemene hører sammen.

8.2 Frontend

Frontend er bygget som en mobile first webapplikasjon. Den bruker Javascript-biblioteket React med en komponentbasert frontend. React komponentene får data de har behov for ved å kalle på REST-APIet i backend. CSS rammeverket React-Bootstrap er tatt i bruk for å forenkle designprosessen.

8.3 Swagger dokumentasjon av REST API

Applikasjonens endepunkter er dokumentert med Swagger. Swagger-UI er brukt for å automatiske opprette et brukergrensesnitt med API-dokumentasjon. Dette gjør det enklere for utviklere å konsumere APIet uten å gå gjennom backend-koden.

Figur 8.2 viser en liste over noen av endepunktene. Dersom man trykker seg inn på en av disse får en opp informasjon om hvilke parametere en gyldig forespørsel må inkludere og hvordan responsen fra serveren vil se ut. En får også opp informasjon om hvilke forskjellige HTTP-feilkoder som kan forventes og hva disse kan komme av. Figur 8.3 viser hvordan dette ser ut for POST-forespørsler til /categories-endepunktet.

The screenshot shows the Swagger UI interface with the following sections:

- Categories**:
 - GET /categories/{id}**: Returns a category with the given id (blue button)
 - PUT /categories/{id}**: Update existing category (orange button)
 - GET /categories**: Returns all categories (blue button)
 - POST /categories**: Insert a new category (green button)
- Products**:
 - GET /products/{id}**: Returns a product with the given id (blue button)
 - DELETE /products/{id}**: Deletes a product with given id (red button)
 - PUT /products/{id}**: Updates an existing product with the given id (orange button)
 - GET /products**: Returns a list of all products (blue button)
 - POST /products**: Add a new product (green button)
- Auth**:
 - POST /auth**: Let user log in (blue button)
 - POST /auth/token/{JWT}**: Let user verify email (green button)
- Users** (no methods listed)

Figur 8.2: Swagger-dokumentasjon

The screenshot shows a Swagger UI interface for a RESTful API. At the top left, there's a green button labeled "POST" and a URL field containing "/categories Insert a new category". On the top right, there's a lock icon. Below the header, a message says: "Use to insert a new category. The category may contain a parent reference. Note that only one level of subcategorization allowed." Under the message, there's a section titled "Parameters" with a "No parameters" message and a "Try it out" button. A "Request body" field is marked as required and set to "application/json". The request body schema is described as: "A JSON object containing category information. Parent field is optional". It includes example values and a schema definition. In the "Responses" section, there are three entries: a 200 response with a description of "A JSON object with the new category", a media type of "application/json", and a note "No links"; a 400 response with a description of "Invalid category format or parent is not a main category.", a media type of "application/json", and a note "No links"; and a 404 response with a description of "Parent with provided parent id does not exist", a media type of "application/json", and a note "No links".

Figur 8.3: Swagger-dokumentasjon av endepunkt

8.4 Biblioteker

Npm (Node package manager) er en pakkebehandler for JavaScript-rammeverket Node.js. Gruppen har benyttet seg av flere biblioteker fra npm-registeret. Nedenfor følger en liste over disse.

8.4.1 Frontend

Pakkenavn	Versjon	Beskrivelse
@fortawesome/react-fontawesome	0.1.9	Inneholder React-komponent for å rendre SVG ikoner
@fortawesome/fontawesome-svg-core	1.2.27	Inneholder FontAwesome ikoner.
@fortawesome/free-solid-svg-icons	5.12.1	Inneholder FontAwesome ikoner.
@hapi/joi	17.1.1	Verktøy for validering av data i javascript.
axios	0.19.2	En løfte basert HTTP klient for nettleseren og Node.js.

bootstrap	4.4.1	Rammeverk for utvikling av responsive mobile-first nettsider.
holderjs	2.9.6	Bibliotek for rendring av bilder som plassholder med SVG.
jwt-decode	2.2.0	Bibliotek for å dekode JSON Web Tokens.
lodash	4.17.15	Inneholder nyttefunksjoner for vanlige programmeringsoppgaver.
moment	2.24.0	Et lettvekts bibliotek for å gjenomgå, validere, manipulere og formtere dato.
react	16.12.0	Et Javascript-bibliotek for å lage brukergrensesnitt
react-bootstrap	1.0.0-beta.16	React-utgave av bootstrap biblioteket med React komponenter istedenfor den tradisjonelle CSS-stylingen bootstrap bruker.
react-dom	16.12.0	Inneholder spesifikke DOM metoder som brukes på topp nivå i webapplikasjonen.
react-router-dom	5.1.2	DOM-bindinger for React Router.
react-scripts	3.4.0	Pakken inkluderer scripter og konfigurasjoner som benyttes av Create React App.
react-toastify	5.5.0	React-komponenter for notifikasjoner.

8.4.2 Backend

Pakkenavn	Versjon	Beskrivelse
@hapi/joi	17.1.1	Verktøy for validering av data i javascript.
bcrypt	4.0.1	Bibliotek for å hashe passord.
config	3.3.0	Bibliotek for hierarkiske konfigurasjoner til applikasjonen.
cors	2.8.5	En node.js pakke som tilbyr Connect/Express middleware som tillater CORS med forskjellige alternativer.
express	4.17.1	Web rammeverk for node.js
express-async-errors	3.1.1	En pakke som automatisk wrapper alle asynkrone kall i try/catch

fawn	2.1.5	Et promise-basert bibliotek for bruk av transaksjoner i MongoDB.
jest	25.1.0	Et rammeverk for testing i javascript. Fungerer med React men også andre rammeverk slik som Angular og Vue.
joi-objectid	3.0.1	Utvidelse av JOI for validering av Mongoose ObjectId
jsonwebtoken	8.5.1	Bibliotek for JSON Web Tokens.
mongoose	5.9.1	Et MongoDB objekt modelleringsverktøy. Støtter både promises og callbacks.
node-schedule	1.3.2	Bibliotek for jobbplanlegging i Node.js. Med dette er det mulig å plakke jobber som skal utføres ved spesifikke datoer med valgfrie gjentakelsesregler.
nodemailer	6.4.5	Bibliotek for sending av e-post via Node.js.
swagger-ui-express	4.1.3	Bibliotek for automatisk generering av UI-grensesnitt basert på swagger-dokumenter.
yamljs	0.3.0	For parsing av yaml-dokumenter. Brukt i forbindelse med swagger-dokumentasjon skrevet i yaml-format.

8.4.3 Eksterne avhengigheter

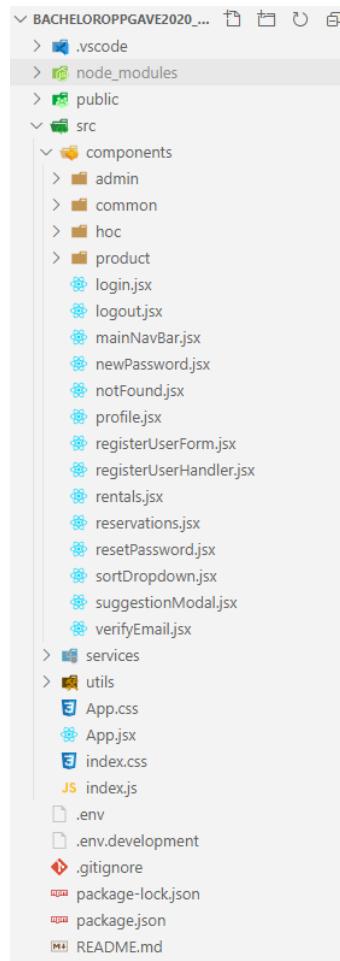
Tjenesten [Sendgrid](#) brukes til å sende e-postnotifikasjoner. SendGrid tilbyr en Simple Mail Transfer Protocol(SMTP) relay-tjeneste med mulighet for opp til 100 gratis e-postmeldinger per dag. En SMTP relay-tjeneste er en tjeneste som sender e-post fra en e-postserver til en annen [29].

8.5 Filstruktur

8.5.1 Frontend

Kildekoden for frontend-delen av webapplikasjonen ligger i mappen src. Under utviklingen av komponenter til applikasjonen ble disse lagt i mappen components. Etterhvert som en komponent ble bestående av flere underkomponenter, ble de samlet i en egen undermappe. Dette resulterer i en bedre oversikt over kodestrukturen, se figur 8.4.

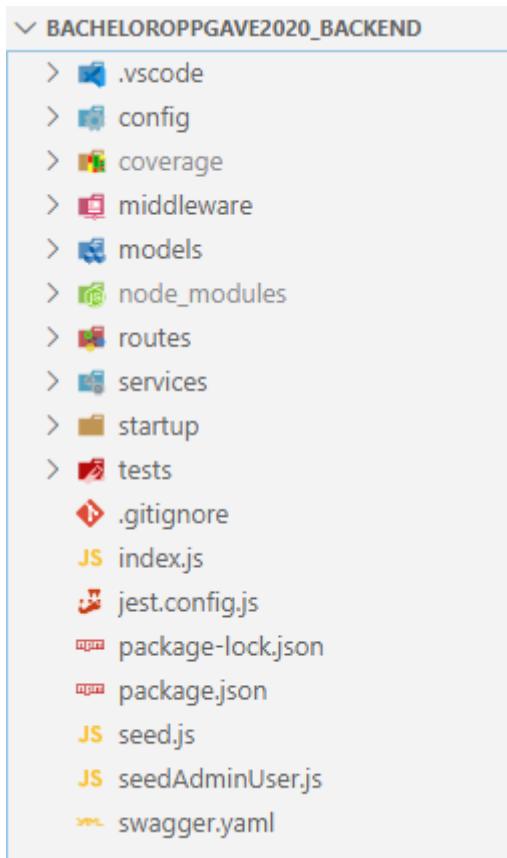
Mappen components inneholder alle react komponentene. Admin-mappen inneholder følgelig alt relatert til administratorfunksjoner, common mappen inneholder komponenter som er mer generelle og kan brukes av flere komponenter. Services-mappen inneholder metoder for kommunikasjon mot APIet.



Figur 8.4: Mappestruktur for frontend

8.5.2 Backend

I backend er filene fordelt i flere mapper som reflekterer deres hensikt. Figur 8.5 viser mappestrukturen.



Figur 8.5: Mappestruktur for backend

Tabellen under inneholder beskrivelse av mappestrukturen:

Mappe	Beskrivelse
config	Filer for konfigurasjon av environment-variabler som beskrevet i kapittel 12.3
coverage	Testrapporter generert av jest
middleware	Middleware brukt av flere routes. For eksempel for validering av autentiserings-token eller validering av request body.
models	Datamodeller for database og valideringsfunksjoner for data.
routes	Endepunktene for applikasjonen.
startup	Filer for oppstart av applikasjoner. Blant annet for tilkobling til databaser, registrering av de ulike endepunktene og oppstart av serveren.
tests	Enhets- og integrasjonstester.

8.6 Gjenbruksbare komponenter

Clean code slik som det er skrevet om i kapittel 2.2, er et viktig konsept for gruppen og har etter beste evne blitt fulgt. Et viktig punkt i Clean code er DRY prinsippet, “Don’t Repeat Yourself”. Tilsvarende er et av kjerneprinsippene i Extreme Programming “Once, and only once.”. Det følger videre at hver gang man dupliserer kode er det en ubenyttet mulighet til å abstrahere. Gruppen har med dette utviklet en del gjenbruksbare komponenter. Et eksempel på dette er filen genericForm.jsx som andre react klasser kan utvide. Koden under viser en utvidelse av denne.

```
//registerUserForm.jsx
class RegisterUserForm extends GenericForm {...}
```

Filen registerUserForm.jsx kan da benytte seg av metoder i genericForm.jsx for å generere et skjema. Figur 8.6 viser hvordan render-metoden til registerUserForm.jsx ser ut.

```
63 |   <Form>
64 |     {this.renderInputForm("firstName", "First name", "Enter first name")}
65 |     {this.renderInputForm("lastName", "Last name", "Enter last name")}
66 |     {this.renderInputForm("phone", "Phone", "Enter phone number")}
67 |     {this.renderInputForm("email", "Email", "Enter email")}
68 |     {this.renderInputForm("password", "Password", "Password", "password")}
69 |   <Link...
83 | </Form>
```

Figur 8.6: Utkast av registerUserForm.jsx sitt skjema.

Legg merke til kallet på metoden renderInputForm, eksempel:

```
{this.renderInputForm("firstName", "First name", "Enter first name")}
```

Metoden renderInputForm kan ses på figur 8.7.

Metoden er bygget for å være gjen brukbar ved at den tar imot ulike parametere. Parametrene brukes til å dynamisk aksessere variabler i state-objektet til klasse-instansen som utvider GenericForm. Se for eksempel linje 76 på figur 8.7. Her hentes variabelen this.state.data[name], hvor name er et dynamisk variabelnavn. Metoden sender disse variablene videre som props til komponenten GenericInput, som returnerer et inputfelt.

Med dette benyttes DRY prinsippet ved at metoden renderInputForm og komponenten GenericInput kun er skrevet én gang, men kalles på flere ganger. Gruppen har da abstrahert koden. Uten dette ville det vært nødvendig å skrive logikken for renderInputForm og GenericInput for hvert inputfelt i applikasjonen. Dette ville medført mye repetitiv kode som ville vært vanskelig å lese og endre på.

```
69   renderInputForm(name, label, placeholder, type = "text") {  
70     const { data, errors } = this.state;  
71  
72     return (  
73       <GenericInput  
74         type={type}  
75         name={name}  
76         value={data[name]}  
77         label={label}  
78         placeholder={placeholder}  
79         onChange={this.handleChange}  
80         error={errors[name]}  
81       />  
82     );  
83   }
```

Figur 8.7: Funksjonen renderInputForm fra genericForm.jsx.

genericForm.jsx har i tillegg to andre metoder som returnerer input-felt, renderInputSelect for input av type select og renderInputTextArea for input type textarea. Metodene brukes på tilsvarende måte som renderInputForm, men da med andre parametere.

Kapittel 9

Sikkerhet

Det er tatt flere sikkerhetshensyn i utviklingen av webapplikasjonen. Tiltakene går blant annet ut på hvordan data lagres uten at uvedkommende har tilgang, hvordan man skiller funksjonalitet basert på autoriseringsnivået til brukeren (administrator eller ikke) og hvordan man logger inn en bruker på en sikker måte. Disse hensynene blir drøftet i dette kapittelet.

9.1 Identifisering, autentisering og autorisering

Dette delkapittelet tar for seg hvordan det er lagt til rette for identifisering, autentisering og autorisering ved bruk av JWT i webapplikasjonen.

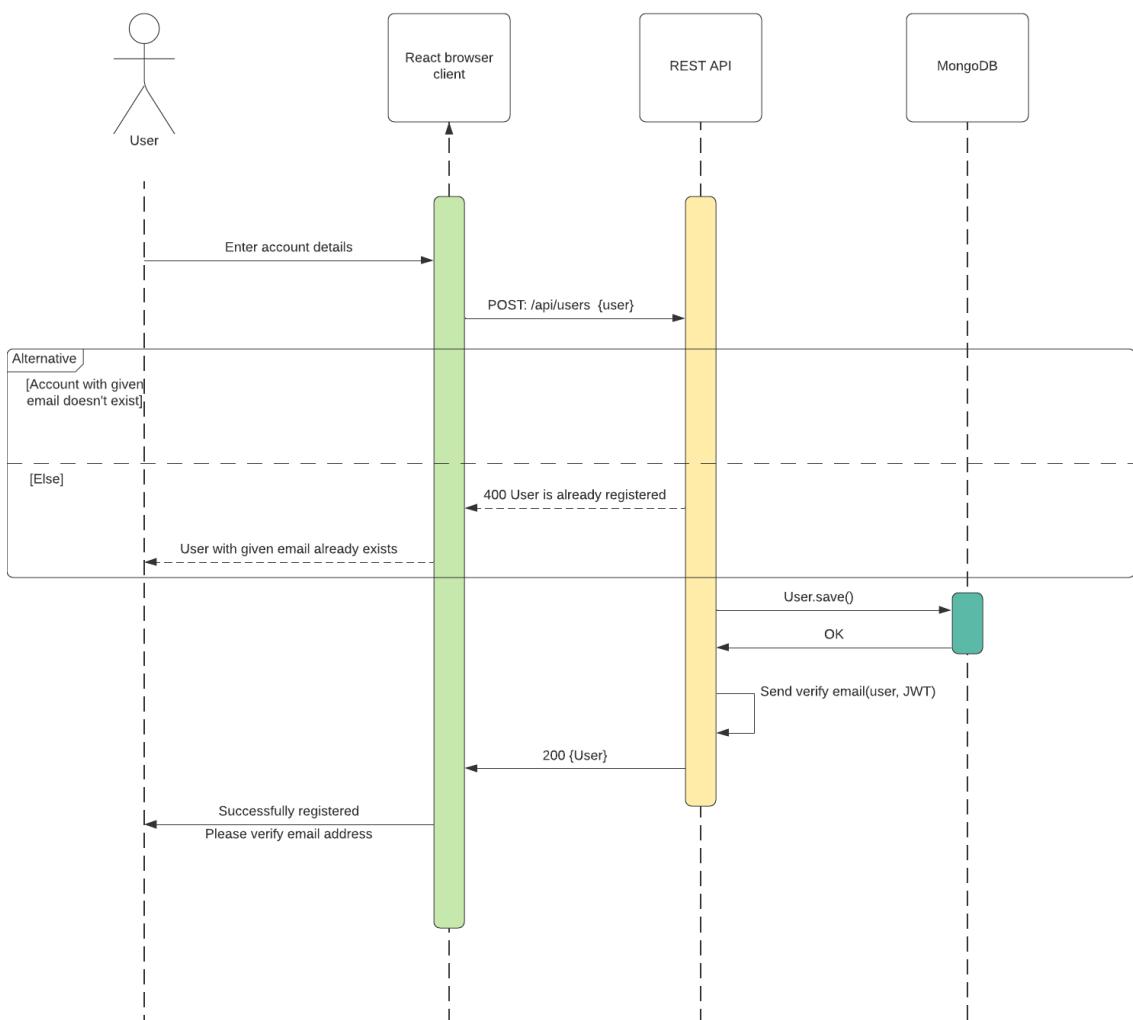
9.1.1 JSON Web Token

Webapplikasjonen bruker JSON Web Token(JWT) for identifisering, autentisering og autorisering av brukere. Et JWT er et token som inneholder informasjon om brukeren. Mer om dette i kapittel 11.6 .

Tokenet genereres i backend etter at en bruker har logget seg inn. Tokenet sendes så til klienten som har logget seg inn. Den autentiserte klienten vil sende dette tokenet sammen med alle etterfølgende forespørsler til server. Dette gjør at server slipper å lagre brukertilstand, hvilket er et av kravene til RESTful API (se kapittel 4.1.1). Tokenet identifiserer brukeren som sender forespørsel til server. JWT er ikke kryptert, bare encoded, så hvem som helst kan lese hva den inneholder, samt endre på innholdet. For å unngå forfalskning inneholder tokenet også en digital signatur som består av en hashing og kryptering. Signeringen er gjort på serversiden med en privat nøkkel og verifiseres med samme nøkkel for hver forespørsel fra klienten.

9.1.2 Registrering

Ut ifra kravspesifikasjonen er det kun Accenture-ansatte som skal ha tilgang til webapplikasjonen. For å innfri dette kravet tillater backend kun registrering av brukere som har en gyldig @accenture e-postadresse. Når en bruker registrerer seg er kontoen satt til inaktiv. En inaktiv bruker kan ikke logge seg inn. En inaktiv bruker kan kun bli aktiv dersom brukeren verifiserer e-postadressen sin. Dette gjøres ved å klikke på en link som er sendt til @accenture e-posten deres. Figur 9.1 viser et sekvensdiagram som beskriver registreringsflyten.



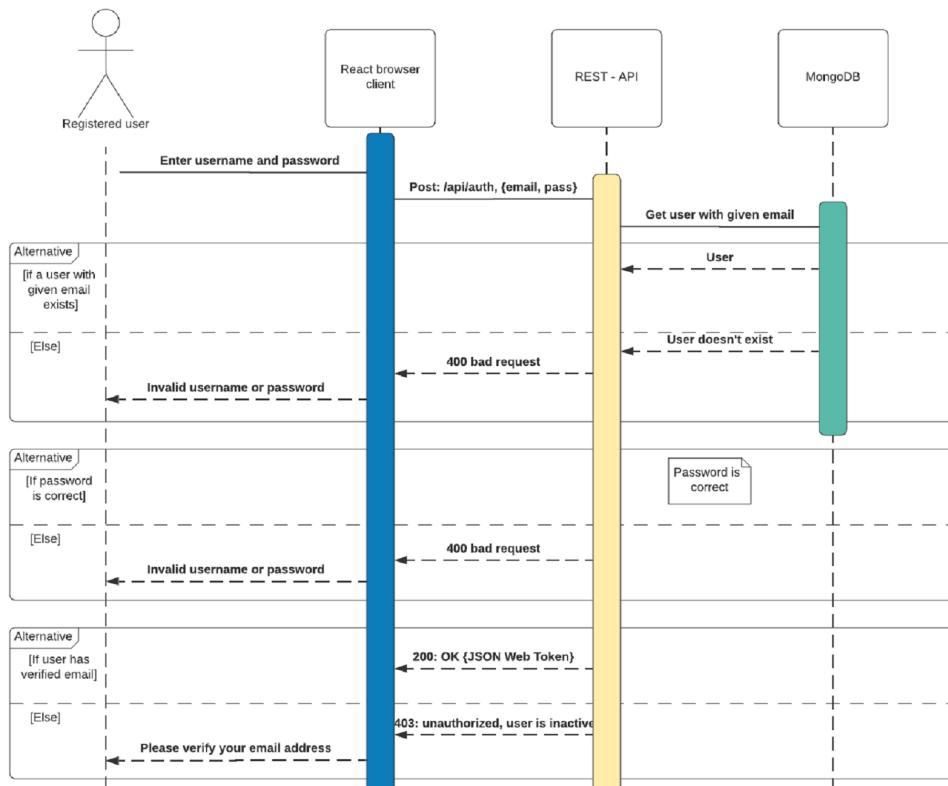
Figur 9.1: Sekvensdiagram for registrering

9.1.3 Innlogging

For å kunne motta et JWT-token fra server må brukeren ha verifisert e-posten sin og deretter logget seg inn med et gyldig brukernavn og passord. Klienten mottar et token fra server som den

lagrer i local storage [17]. Local storage er en liten lokal database som alle nettlesere har tilgang til.

Figur 9.2 er et sekvensdiagram som illustrerer hvordan en bruker logger seg inn og mottar en JWT fra server.



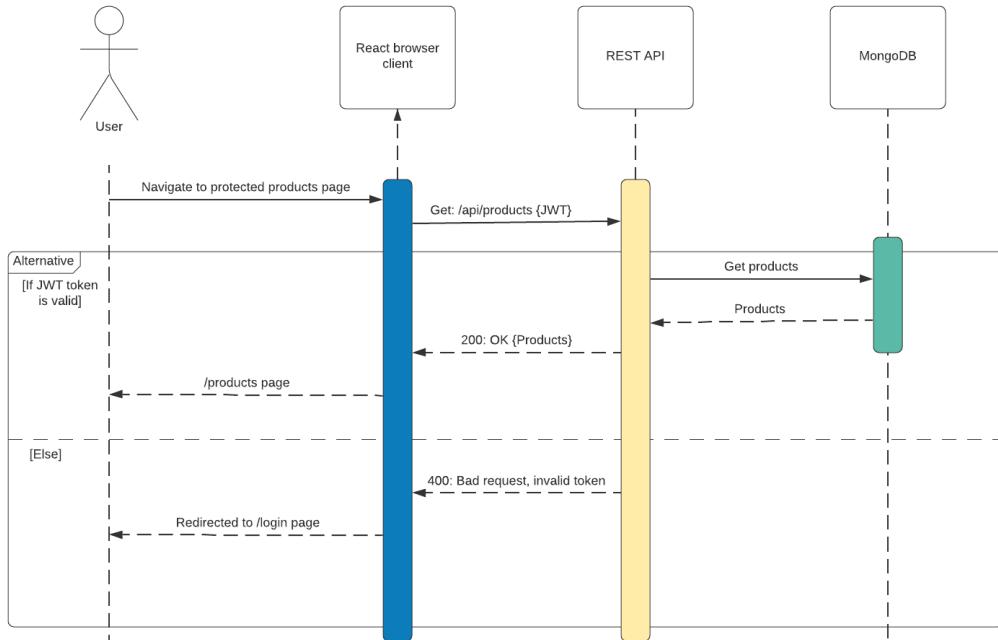
Figur 9.2: Log in sekvensdiagram

9.1.4 Aksessering av en beskyttet ressurs

For å få tilgang til en beskyttet ressurs, som for eksempel data relatert til produktsiden, må brukeren være logget inn. APIet responderer med data hvis det er sendt med et gyldig JWT i forespørselen. Figur 9.3 beskriver sekvensen når en bruker prøver å aksessere den beskyttede ressursen /products.

9.1.5 Administratorfunksjonalitet

JWT inneholder det boolske feltet isAdmin som angir om en bruker er administrator eller ikke. En administrator har høyere autorisasjonsnivå enn andre brukere og har blant annet rettigheter til å legge til nye produkter. Når backend mottar en forespørsel som krever administratorrettig-



Figur 9.3: Aksessere en beskyttet ressurs

heter, sjekkes det om JWT tokenet sitt isAdmin felt er “true”. HTTP-status 403 returneres med meldingen “Forbidden” dersom brukeren ikke er autorisert.

9.2 Lagring av passord

Passord blir lagret i databasen for å kunne muliggjøre autentisering. Som forklart i kapittel 5.3.2 om teknologivalg, bruker vi hashingalgoritmen bcrypt for å beskytte passordene i databasen. Bcrypt både hasher og salter passordet, slik at dersom en angriper får tilgang til databasen, så kan ikke passordene hentes ut, selv med en regnbuetabell. I tillegg til hashing og salting kjøres bcrypt med en parameter som bestemmer hvor treg algoritmen blir. Denne tregheten gjør at brute-force angrep blir beregningsmessig umulig.

9.3 Passordkompleksitet

For å garantere at passordene som lages av brukere er sikre så brukes valideringsbiblioteket [Joinpassword-complexity](#). National institute of science and technology(NIST) anbefaler bruk av så langt passord som mulig(8-64) symboler [9]. Vi valgte det nedre grensen av NIST sin passordlengde, 8 symboler, for å gjøre applikasjonen både brukervennlig og sikker, da sikkerhet kan komme på bekostning av brukervennlighet. Vi håndhever også en regel hvor hvert passord må inneholde minst

en stor og en listen bokstav, i tillegg til et tall.

9.4 Validering

Prinsippet om lagdeling av sikkerhetstiltak, også kjent som sikkerhet i dybden, diskuteres i kapittel 4.4.3. Vi implementerer dette prinsippet i valideringen vår. Valideringen foregår i tre omganger: Den første er på klientensiden, den andre er mot APIet og den siste er mot databasen. Mer om dette i kapittel 10.1.

Kapittel 10

Validering

10.1 Validering

Webapplikasjonen bruker en rekke valideringer, alt fra når bruker registerer seg, logger inn eller skal låne produkt(er). Enkelte valideringer er for sikkerhet slik at kun verifiserte ansatte skal kunne låne, mens andre er for å gi brukeren tilbakemelding på feil ved registrering og innlogging. Validering foregår både i frontend og backend. I backend validerer vi mot input til APIet og input til MongoDB via mongoose.

10.1.1 Frontend

Form-validering i frontend gjøres med tanke på brukervennlighet. Når det tastes inn ugyldig input blir brukeren informert om dette for å gjøre registrering og innlogging enklere. Hvis validering utelukkene hadde skjedd i backend hadde det ført til en tregere applikasjon, da man må ta veien helt til serveren og tilbake for å få tilbakemelding om input er gyldig.

Figur 10.1 viser et eksempel på hva bruker kan få som feilmelding ved registrering. Hvis alle feltene oppfyller kravene, blir “Create user” knappen aktiv og brukeren kan sende inn skjemaet. Hvis det viser seg at brukeren allerede eksisterer, informeres brukeren om dette. Ellers vises det en melding om vellykket registrering og bruker mottar e-post for å verifisere e-postadressen. Når brukeren prøver å logge inn men skriver feil brukernavn eller e-post blir bruker informert om at det er ugyldige innloggingsdetaljer.

10.1.2 Backend

Selv om vi validerer i frontend betyr ikke det at ugyldig data ikke kan sendes til APIet. Man kan kalle på APIet direkte ved bruk av f.eks [Postman](#). Derfor validerer vi også forespørsler til

Register here

First name

Not a valid name format

Last name

Not a valid name format

Phone

Must be a valid phone number

Email

Not a valid accenture mail

Password

"Password" length must be at least 8 characters long

[Create user](#)

Figur 10.1: Validering av brukerinput ved registrering av bruker

APIet i backend. Validering i backend blir hovedsaklig foretatt ved bruk av biblioteket [Joi](#). Joi er det samme valideringsbiblioteket som brukes i frontend. Dersom input er ugyldig returnerer vi HTTP status 400 inkludert mer utfyllende informasjon om hva som gikk galt med forespørselen. En respons-melding kan for eksempel være "Could not create user because email address is required".

Validering av data som skal inn til databasen valideres av mongoose. Dette sørger for at dataene som lagres alltid har et predefinert format med eventuelle påkrevde felt. Dersom ny backend-kode skrives og det ikke tas hensyn til validering, vil dette fortsatt fanges opp av mongoose. Dette følger prinsippet om lagdelt sikkerhet.

Kapittel 11

Sentrale datastrukturer

Data som applikasjonen håndterer lagres i en MongoDB database. Databasen består av følgende kolleksjoner: Products, Categories, Users, Rentals, Service Messages og Suggestions. Deres oppbygning og livssyklus blir beskrevet i dette kapittelet.

11.1 Rentals

Rentals-kolleksjonen inneholder utlånsdata. Det er et embedded JSON dokument som består av brukeren som utførte lånet, produktet som blir utlånt og data om selve lånet. Styrken ved å bruke embedded dokuments er at man kun trenger en spørring for å få informasjon om utlånet, brukeren som utførte utlånet og produktet som ble utlånt. I en tradisjonell SQL-database hadde man trengt tre spørninger (i form av joins) for å hente inn samme data.

Rentals har i tillegg til produkt- og brukerfeltet også felt som inneholder annen informasjon om lånet. Feltene er som følger:

- dateOut - Datoen og tiden på døgnet produktet ble utlånt
- dateReturned - Datoen og tiden på døgnet produktet ble returnert
- pickUpInstructions - Informasjon om hvor produktet skal hentes
- returnInstructions - Informasjon om hvor produktet skal leveres tilbake
- dateToReturn - Datoen produktet skal returneres
- remarks - Informasjon om problem med produktet

Disse feltene blir satt i løpet av livssyklusen til et utlån. Hvilke felt som er satt bestemmer i hvilket

stadie et utlån er i. Et utlån kan i løpet av sin livssyklus være i fire ulike stadier. Stadiene blir beskrevet under:

Brukere har forespurt et utlån

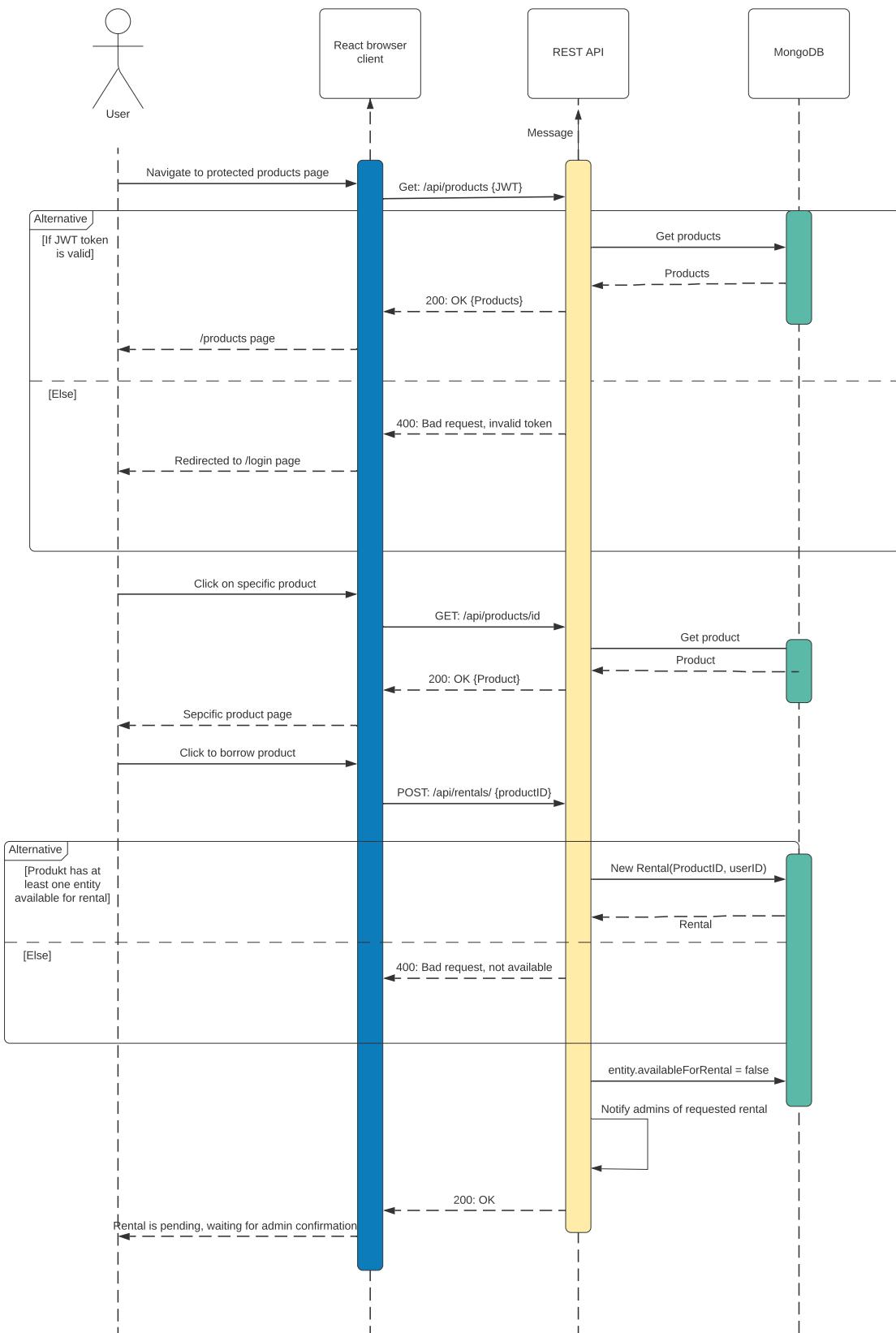
Det første stadiet forekommer når en bruker har forespurt et utlån av et produkt. Da lagres brukeren som utførte forespørselen og produktet det gjelder i utlånskolleksjonen slik vi kan se i JSON-objektet under. I tillegg til dette settes confirmedReturned til false. Sekvensdiagrammet som beskriver hendelsesforløpet når en bruker forespør et utlån vises på figur 11.1. Sekvensdiagrammene til alle stadiene i utlånslivssyklusen er å finne i vedlegg J

```
// Kolleksjon: Rentals
{
  _id: "5e9db2c57fa9697a141ebef",
  user: {
    _id: "5e9db2b19ee7de10a8be67bd",
    name: "Markus Hellestveit",
  },
  product: {
    _id: "5e9db2b19ee7de10a8be67a6",
    name: "Playstation 4"
    entity: {
      _id: "5e9db2b19ee7de10a8be67a8"
      identifier: "PS4_1"
    }
  }
  confirmedReturned: false
}
```

Admin har godkjent utlånet

Etter at en bruker har forespurt et utlån av et produkt, må bruker vente på at en administrator godkjenner utlånet. Ved godkjenning legges det til utelevering- og henteinstruksjoner. Det kan også spesifiseres hvor lenge utlånet skal være, hvor standard er syv dager dersom feltet ikke blir satt. dateToReturn settes til datoën produktet skal leveres tilbake. DateOut-feltet blir også satt og markerer at lånet er startet.

JSON objektet under beskriver rental dokumentet i dette stadiet av livssyklussen.



Figur 11.1: Sekvensdiagram for låneforespørsel

```
// Kolleksjon: Rentals
{
  _id: "5e9db2c57fa9697a141ebef",
  user: {
    _id: "5e9db2b19ee7de10a8be67bd",
    name: "Markus Hellestveit",
  },
  product: {
    _id: "5e9db2b19ee7de10a8be67a6",
    name: "Playstation 4"
    entity: {
      _id: "5e9db2b19ee7de10a8be67a8"
      identifier: "PS4_1"
    }
  }
  confirmedReturned: false
  dateOut: 2020-04-22T06:36:50.876+00:00
  dateToReturn: 2020-04-29T06:36:50.876+00:00
  pickUpInstructions: "At office"
  returnInstructions: "In cafeteria"
}
}
```

Brukeren har levert tilbake produktet

Når brukeren leverer tilbake det produktet, settes dateReturned feltet. Brukeren kan også legge til valgfrie bemerkelser dersom det er problemer eller feil med produktet. Disse havner under remarks-feltet.

```
// Kolleksjon: Rentals
{
  _id: "5e9db2c57fa9697a141ebef",
  user: {
    _id: "5e9db2b19ee7de10a8be67bd",
    name: "Markus Hellestveit",
  },
  product: {
    _id: "5e9db2b19ee7de10a8be67a6",
    name: "Playstation 4"
    entity: {
      _id: "5e9db2b19ee7de10a8be67a8"
      identifier: "PS4_1"
    }
  }
  confirmedReturned: true
  dateOut: 2020-04-22T06:36:50.876+00:00
  dateToReturn: 2020-04-29T06:36:50.876+00:00
  remarks: "Product was damaged during delivery"
  returnInstructions: "In cafeteria"
}
}
```

```

        name: "Playstation 4"
        entity: {
            _id: "5e9db2b19ee7de10a8be67a8"
            identifier: "PS4_1"
        }
    }

    confirmedReturned: false,
    dateOut: 2020-04-22T06:36:50.876+00:00,
    dateToReturn: 2020-04-29T06:36:50.876+00:00,
    pickUpInstructions: "At office",
    returnInstructions: "In cafeteria",
    dateReturned: 2020-04-22T06:45:00.867+00:00,
    remarks: "Scartches at the bottom of the console."
}

```

Admin har godkjent retur av produkt

Dette er det siste stadiet i et utlåns livssyklus. Her bekrefter en administrator at produktet er blitt levert tilbake og confirmedReturned-feltet blir satt til "true".

```

// Kolleksjon: Rentals
{
    _id: "5e9db2c57fa9697a141ebefe",
    user: {
        _id: "5e9db2b19ee7de10a8be67bd",
        name: "Markus Hellestveit",
    },
    product: {
        _id: "5e9db2b19ee7de10a8be67a6",
        name: "Playstation 4"
        entity: {
            _id: "5e9db2b19ee7de10a8be67a8"
            identifier: "PS4_1"
        }
    }
    confirmedReturned: true,
    dateOut: 2020-04-22T06:36:50.876+00:00,
    dateToReturn: 2020-04-29T06:36:50.876+00:00,
}

```

```

    pickUpInstructions: "At office",
    returnInstructions: "In cafeteria",
    dateReturned: 2020-04-22T06:45:00.867+00:00,
    remarks: "Scartches at the bottom of the console."
}
```

11.2 Users

Kolleksjonen Users inneholder informasjon om de registrerte brukerne til applikasjonen. JavaScript-objektet under beskriver oppbygningen av et user-dokument.

```
// Kolleksjon: Users
{
  _id: "5e9fe36ea5302a5aa412c237",
  isActive: true
  isAdmin: false
  firstName: "Markus",
  lastName: "Hellestveit",
  email: "markus@gmail.com",
  password: "$2b$10$.0WFec/wvMdmKdYHbZIcQ.BJQYgkEudbCgH5TUVuHe0i.foj1c90a",
  phone: "42817581"
}
```

Feltet isAdmin bestemmer om bruker er administrator. IsActive-feltet avgjør om brukeren er aktivert. Ved opprettelse av bruker er denne satt til false. Ved verifisering av e-postadresse blir den satt til true. Passordet som er lagret er en hashet og saltet utgave av det opprinnelige passordet. E-postadressen lagres alltid med små bokstaver.

11.3 Product

Produktkolleksjonen innholder informasjon om produktene en bruker kan låne.

```
// Kolleksjon: Products
{
  _id: "5e9fe36da5302a5aa412c226",
  numberOfLoans: 48,
  name: "Nintendo Switch",
  category: {
    _id: "5e9fe36da5302a5aa412c21d"
```

```

    "name": "switch"
  },
  entities: [
    {
      _id: "5e9fe36da5302a5aa412c228",
      identifier: "Switch_1",
      remarks: "Has scratches on the bottom",
      availableForRental: true
    },
    {
      _id: "5e9fe26da4812a5aa412c982",
      identifier: "Switch_2",
      remarks: "",
      availableForRental: false
    }
  ]
}

```

JavaScript-objektet ovenfor representerer et product-dokument og inneholder et array med entiteter, som er utlånbare enheter av produktet. I dette tilfellet finnes det to Nintendo Switch konsoller. AvailableForRental-feltet i hver entitet bestemmer om entiteten kan lånes ut. Identifier-feltet identifiserer det fysiske produktet (identifieren kan f.eks. være et firesifret nummer som limes på produktet) og hjelper administratoren til å skille mellom entiteter.

NumberOfLoans angir hvor mange ganger det overordnede produktet har blitt lånt ut og inkrementeres for hvert utlån av en entitet.

11.4 Service Message

```
// Kolleksjon: Service Message
{
  _id: "5e9fe36da5302a5aa412c226",
  serviceMessage: "Due to vacation, we can't deliver any products next week."
}
```

Denne kolleksjonen inneholder statusmeldinger fra administrator som vises på toppen av webapplikasjonen.

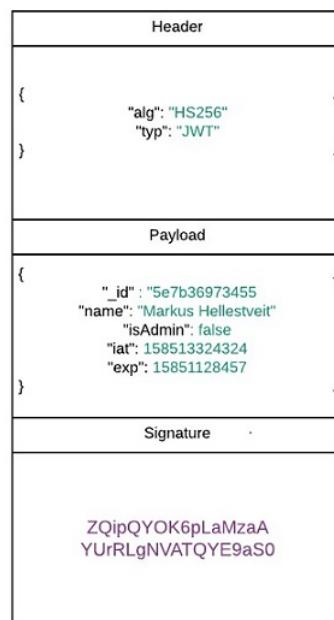
11.5 Suggestion

```
// Kolleksjon: Suggestions
{
  _id: "5e9fe36da5302a5aa412c226",
  Suggestion: "Buy Valve Index"
  user: {
    _id: "5eafcac406805b5b40f5c9f4"
    "name": "Josh Johnson"
  }
}
```

Suggestion-kolleksjonen består av produktforslag brukere sender inn. Dokumentet inneholder forslaget under feltet suggestion og et bruker-objekt som representerer brukeren som kom med forslaget

11.6 Autentiseringstoken

Webapplikasjonen bruker JWT for identifisering, autentisering og autorisering av brukere. Se [9.1.1](#) for informasjon om hvordan JWT fungerer. Figur [11.2](#) beskriver oppbygningen av tokenet applikasjonen bruker for autentisering og autorisering. Tokenet inneholder informasjon om den autentiserte brukeren med bruker-ID, navnet på brukeren og isAdmin-variabelen. Feltet "iat", exp-feltet definerer når tokenet ikke lenger er gyldig [\[20\]](#).



Figur 11.2: JWT struktur

Kapittel 12

Serverkonfigurasjon

DigitalOcean er et selskap som tilbyr forskjellige skytjenester. Gruppen fikk utdelt en virtuell maskin (VM) hos DigitalOcean til hosting av webapplikasjonen. VMen kjører Ubuntu versjon 18.04. Dette kapittelet gir en oversikt over konfigurasjonen av serveren.

12.1 Tilgang til server

For å få tilgang til VMen benyttes SSH med et RSA-nøkkelpar. For å koble til root-brukeren på serveren via SSH med en nøkkel **privKey** benyttes følgende kommando:

```
ssh -i privKey root@167.71.8.22
```

12.2 Installasjon av programvare

Installasjon av følgende programvare er nødvendig for å kjøre webapplikasjonen:

1. Node.js 12.16.x
2. MongoDB 4.2.x

Når disse er installert må mappen **/data/db** opprettes. Her vil MongoDB lagre dataene. Kommandoen under oppretter mappen:

```
mkdir -p /data/db
```

12.3 Environment-variabler

For konfigurasjon av applikasjonen i forskjellige miljøer benytter applikasjonen environment-variabler. Dette gjør det mulig å benytte forskjellige konfigurasjoner i utviklingsmiljø og produksjonsmiljø. F.eks ønsker vi ikke å benytte Accenture e-post i utviklingsmiljøet, og skrur av denne regelen ved å benytte en variabel for dette. Nedenfor følger en beskrivelse av variablene appen tar i bruk.

Variabelnavn	Beskrivelse	Default-verdi
borrowMyTech_frontEndBaseUrl	Setter URL til frontend. Benyttes i sending av e-poster for å opprette trykkbar lenke ved f.eks resetting av passord. Brukes også til å sette CORS-policy for API-et.	http://localhost:3000
NODE_ENV	Gir informasjon til Node om hvilket miljø koden kjøres i. Settes til production i produksjonsmiljø.	development
DB_URI	Setter URI til databasen som backend skal benytte. Denne kan endres dersom det skal benyttes cloudtjenester for hosting av databasen.	<code>mongodb://localhost:27017/loan-app</code>
jwtPrivateKey	Privatnøkkel for signering av JWT token. Denne er viktig å sette for å unngå forfalskning av JWT.	development
requiresAuth	Angir om autentisering kreves for tilgang til endepunkter i backend. Kan brukes for å skru av autentisering under utvikling og testing.	true
requiresAccentureEmail	Settes for å kun tillate oppretting av brukere med Accenture e-post.	Avhenger av NODE_ENV. false i development, true i production.

borrowMyTech_mailServer_host	Host for mailserveren. Benyttes av nodemailer-pakken for å sende e-post.	smtp.sendgrid.net
borrowMyTech_mailServer_user	Brukernavn for mailserveren. Benyttes av nodemailer-pakken for å sende e-post.	apiKey
borrowMyTech_mailServer_pass	Passord for mailserveren. Benyttes av nodemailer-pakken for å sende e-post.	
REACT_APP_BACKEND_URL	Benyttes av frontend for forespørsler til backend.	http://localhost: 3900/api i development-miljø
PORT	Benyttes av Node for å definere porten applikasjonen skal lytte til. Benyttes av både frontend og backend. Må derfor overstyres når frontend og backend kjøres separat på samme maskin.	Dersom variabelen ikke blir satt er standardverdi 3000 i frontend og 3900 i backend.

De minimale variablene som må settes på serveren for å kjøre applikasjonen er:

- NODE_ENV
- PORT
- REACT_APP_BACKEND_URL
- borrowMyTech_frontEndBaseUrl
- borrowMyTech_jwtPrivateKey
- borrowMyTech_mailServer_pass (API-nøkkel fra sendgrid.com. Andre e-posttjenester kan benyttes ved å sette host og user-variablene i tillegg.)

Variablene er satt i **.bashrc** filen i hjem-mappen på serveren. Fungerende eksempel på .bashrc-filen:

```
#Environment variables
export NODE_ENV="production"
```

```
export PORT=3900
export REACT_APP_BACKEND_URL="http://167.71.8.22:3900/api"
export borrowMyTech_frontEndBaseUrl="http://167.71.8.22"
export borrowMyTech_jwtPrivateKey="superSecretServerKey"
export borrowMyTechmailServerpass="EXAMPLEKEY"
```

Etter at variablene er satt i `.bashrc` vil de være tilgjengelig i alle påfølgende bash-vinduer som startes.

12.4 Kjøring av applikasjonen

For å kjøre applikasjonen på VM har gruppen valgt å la frontend og backend kjøre hver for seg. Det er likevel verdt å nevne at express også kan brukes til å servere statisk innhold, slik at disse kunne blitt kjørt i en og samme express-applikasjon dersom dette er ønskelig.

12.4.1 Frontend

Forutsatt at de minimale environment-variablene ovenfor er satt, kan frontend nå kjøres ved å installere avhengigheter, bygge prosjektet, installere npm-pakken serve og benytte denne for å kjøre applikasjonen med lytting etter HTTP-forespørslar på port 80. Dette gjøres med følgende kommandoer:

```
#Forutsetter at nåværende mappe er rotmappen til frontend.

npm install
npm run build
npm i -g serve
serve -p 80 -s build &
```

12.4.2 Backend

For å opprette den første admin-brukeren, er det laget en seed-fil i backend med navn **seedAdminUser.js**. Denne oppretter en admin-bruker med brukernavn **admin@admin.com** og passord **defaultAccentureAdmin**. Denne kjøres før første gang applikasjonen skal startes og benyttes til å lage en admin-bruker. Deretter kan denne brukes til å forfremme en annen admin, før den så slettes. For å sette inn denne brukeren benyttes kommandoen:

```
#Forutsetter at nåværende mappe er rotmappen til backend.

node ./seedAdminUser.js
```

Backend vil benytte seg av **PORT**-variabelen nevnt ovenfor for å finne ut av hvilken port den skal lytte til. For å kjøre applikasjonen benyttes kommandoen:

#Forutsetter at nåværende mappe er rotmappen til backend.

```
npm install
```

```
node index.js &
```

Kapittel 13

API Dokumentasjon

Vi har dokumentert applikasjonens endepunkter i RESTful APIet ved bruk av swagger.io.

13.1 Motivasjon for API dokumentasjon

“You can have the best, functional product, but no one will use it if they don’t know how to.”[40]
Et godt dokumentert API gjør det enkelt for utviklere som ønsker å konsumere APIet. En frontend-utvikler som ikke har mye erfaring med backend APIet slipper å analysere kode for å finne ut hva slags forespørsler og responser et API tar og gir. I tillegg får utviklere som ikke har vært med fra starten av og som ønsker å videreutvikle en applikasjon stor gevinst av dokumentasjonen.

13.2 Swagger.io

Swagger definerer regler for et format som beskriver REST-API. Formatet er leselig for både mennesker og maskiner. Swaggerdokumentasjon kan generes automatisk ved bruk av annotering rett i koden, eller i en egen YAML fil. Gruppen har brukt en YAML-fil.

13.3 API Dokumentasjonen

The screenshot shows the Swagger UI interface for a RESTful API. At the top, there is a header with a dropdown for 'Servers' set to 'http://localhost:3900/api' and a green 'Authorize' button with a lock icon.

The main area is organized into sections:

- Categories** (blue header):
 - GET /categories/{id}**: Returns a category with the given id (blue background)
 - PUT /categories/{id}**: Update existing category (orange background)
 - GET /categories**: Returns all categories (blue background)
 - POST /categories**: Insert a new category (green background)
- Products** (blue header):
 - GET /products/{id}**: Returns a product with the given id (blue background)
 - DELETE /products/{id}**: Deletes a product with given id (red background)
 - PUT /products/{id}**: Updates an existing product with the given id (orange background)
 - GET /products**: Returns a list of all products (blue background)
 - POST /products**: Add a new product (green background)
- Auth** (blue header):
 - POST /auth/resetpassword**
 - POST /auth**: Let user log in (blue background)
 - POST /auth/token/{JWT}**: Let user verify email (green background)
- Users** (blue header):
 - PATCH /users/me**: updates logged in user (green background)

Figur 13.1: Swagger-dokumentasjon

Dokumentasjonen skrevet i swagger kan nå vises i et grafisk web-grensesnitt ved å bruke [Swagger-UI](#). Figur 13.1 viser hvordan dette ser ut for noen av endepunktene. Endepunktene kan klikkes på for å få mer informasjon. Figur 13.2 viser informasjonen man får når man klikker på et endepunkt. Under **Request body** på figuren ser man at dersom man ønsker å lage en ny bruker må man sende et JSON object med firstName, lastName, email, password og phone.

Man ser også under **Responses** at dersom man sender en gyldig forespørsel så får man en HTTP 200-respons med data om brukeren som ble generert. Det hashede passordet blir ikke returnert med responsen av sikkerhetshensyn.

Swagger kan også brukes til testing av endepunkter ved å trykke på "Try it out"-knappen som er synlig i figur 13.2. Her kan en ta utgangspunkt i det predefinerte eksempelet og redigere på dette før det sendes. Responsen vises direkte i brukergrensesnittet. Dette gjør det svært enkelt å teste forespørsler under utvikling.

Fullverdig Swagger-dokumentasjon for prosjektet finnes i vedlegg G og kan aksesseres på localhost:3900/api når backend kjøres på lokal maskin.

The screenshot shows a Swagger UI interface for a POST request to the endpoint `/users`. The title bar indicates the method is `POST` and the URL is `/users Add a new user`. A lock icon is present in the top right corner.

Parameters: No parameters.

Request body (required): application/json

A JSON object containing user information is shown in the request body editor:

```
{
  "firstName": "Josh",
  "lastName": "Johnson",
  "email": "Josh_Johnson@gmail.com",
  "password": "PasswordOrJosh_Johnson",
  "phone": "48927401"
}
```

Responses:

Code	Description	Links
200	Return the new user with its corresponding ID	No links
400	Invalid email or password, or user is already registered	No links

For the 200 response, the media type is set to `application/json`, which controls the Accept header. An example value is provided in the schema editor:

```
{
  "_id": "5e6106c3b3817f3a88be0860",
  "firstName": "Josh",
  "lastName": "Johnson",
  "email": "Josh_Johnson@gmail.com",
  "phone": "48927401"
}
```

Figur 13.2: Swagger-dokumentasjon

13.4 Swagger-autentisering

Da store deler av applikasjonen krever et autentiserings-token for å fungere, må denne også sendes med alle forespørsler fra swagger. For å gjøre dette kan en logge inn ved å sende en POST-forespørsel til /auth-endepunktet. Denne vil respondere med et JWT-token. Øverst på figur 13.1 kan en trykke på “Authorize”-knappen og lime inn tokenet. Denne setter HTTP-headeren “x-auth-token” for alle etterfølgende forespørsler swagger sender.

Kapittel 14

Testdokumentasjon

Gruppen har under prosjektet hatt hovedfokus på testing av backend. Dette vil sørge for at backend fungerer som tiltenkt, at databasen holdes i konsistent form til enhver tid og at alle endepunkter som skal ha begrenset tilgang faktisk er sikret. Gruppen hadde satt seg som mål å ha minst 75% testdekning av backend. Testing er utført i [Jest](#), et rammeverk for testing i Javascript.

Når det kommer til frontend har denne vært i stadig endring under prosjektet, noe som har gjort det utfordrende å skrive automatiserte tester. Derfor har testing i frontend blitt gjort manuelt. Det har også blitt utført en kundeakseptansetest, i form av brukertest, av hele applikasjonen.

14.1 Testverktøy

Vi bruker bibliotekene Jest og Supertest for å automatisere testing.

14.1.1 Jest

Jest er et rammeverk som tilbyr tjenester for både enhets- og integrasjonstester. Jest tilbyr paralleliserte tester, hvor hver test kjører i sin egen tråd [22]. Jest kan automatisk generere en dekningsrapport som viser hvor mye av koden som er testet. Denne rapporten gjør det enkelt å få et overblikk over hva som gjenstår å testes uten å måtte analysere kode.

14.1.2 Supertest

Supertest er et Javascript bibliotek som lar oss automatisk sende HTTP-forespørsler til APIet. Biblioteket tilbyr et API som abstraherer bort lavnivåkoden forbundet med API-kall, noe som gjør integrasjonstestene våre renere [32].

14.2 Testing

Vi har testet alle endepunktene våres grundig, ved bruk av både integrasjons og enhetstesting. Testene dekker tilnærmet 100% av linjene i APIet, som vist i figur 14.2. Denne dekningsrapporten generes automatisk av Jest. Av tiden som gikk til utvikling i backend, tok testene lengst tid å skrive. Likevel var dette hensiktsmessig, da vi oppdaget flere feil i logikken da vi kjørte testene. En annen fordel med grundig kodedekning kom når vi refaktorerte koden. For hver refaktorering kunne vi kjøre testene for å bekrefte at koden fortsatt ga forventet resultat.

File		Statements	Branches	Functions	Lines
Bacheloroppgave2020_Backend	[Green]	100%	9/9	100%	100%
Bacheloroppgave2020_Backend/middleware	[Green]	89.22%	91/102	68.63% 35/51	91.67% 11/12
Bacheloroppgave2020_Backend/models	[Green]	98.7%	76/77	50% 2/4	100% 12/12
Bacheloroppgave2020_Backend/routes	[Green]	91.74%	322/351	80% 72/90	90.24% 37/41
Bacheloroppgave2020_Backend/services	[Green]	100%	7/7	50% 1/2	100% 1/1
Bacheloroppgave2020_Backend/startup	[Green]	87.5%	42/48	37.5% 3/8	75% 6/8

Figur 14.1: Testdekningsrapport av alle moduler i backend

14.2.1 Enhetstester

Mesteparten av koden i backend er kode for REST-APIet. Av den grunn har vi stor overvekt av integrasjonstester over enhetstester. Integrasjonstester er bedre egnet til å teste et API på grunn av de eksterne avhengighetene det kan ha. For å bruke enhetstester på endepunktene må man mocke database-, request- og response-objektet. Dette vil føre til mye kode som gjør testkoden vanskeligere å lese og videreutvikle. Enhetstestene tester hovedsakelig metoder og funksjoner, klasser og grensesnitt

Figur 14.2 viser hvordan vi enhetstester modulen validateObjectId. ValidateObjectId er et middleware som verifiserer at en ID som sendes som parametert til APIet er en gyldig ObjectID brukt av MongoDB/mongoose. I dette tilfellet setter vi en ugyldig ID i parameteren og møller en request, response og next object med alle de feltene som er krevd av validateObjectId. Testen sjekker at metoden returnerer en HTTP 400-feil dersom IDen ikke er gyldig.

14.2.2 Integrasjonstester

Til integrasjonstesting tar vi i bruk supertest for å kalle på APIet. I integrasjonstestene ser vi på informasjonsflyten mellom de ulike delene til applikasjonen. Det testes at APIet mener eller leser fra databasen hvis det blir gitt gyldig forespørsel og at den returner feilkode dersom forespørselen er ugyldig. Hvert eneste endepunkt har blitt testet. Gruppen hadde som mål å oppnå mer enn 70% testdekning av endepunktene våres, men endte opp med tilnærmet lik 100% dekning. I alle

```

26  it("should return 400 if ObjectId is invalid", () => [
27    const invalidObjectId = "WIOAJRIOJWA";
28
29    const req = {
30      params: {
31        id: invalidObjectId
32      }
33    };
34
35    var res = new MockExpressResponse();
36    res.status = jest.fn(() => {
37      return {
38        send: jest.fn()
39      };
40    });
41
42    next = jest.fn();
43
44    validateObjectId(req, res, next);
45    expect(res.status).toHaveBeenCalledWith(400);
46  ]);

```

Figur 14.2: Enhetstest av metoden validateObjectId

endepunkter som krever autentisering og autorisering testes det for at det ikke er mulig å sende forepørsler uten å være autentisert, noe som er viktig for å ivareta applikasjonens sikkerhet.

En testmetodikk vi har brukt er å definere en funksjon kalt exec med et gyldig API-kall. For hver etterfølgende test modifiserer vi på en av variablene i exec-funksjonen benytter og holder de andre konstante. Modifiseringen gjør API-kallet ugyldig og dette skal testen oppdage. På denne måte blir koden lettere å lese og vedlikeholde da vi enkapsulerer API-kall-logikken i en metode.

Vi har skrevet totalt 200 enhets- og integrasjonstester. Resultatene fra disse testene finnes i vedlegg I.

14.2.3 Systemtester

I utførelsen av systemtester har gruppen tatt i bruk manuell testing. Ettersom ny funksjonalitet har blitt implementert, har minst en annen person gått gjennom koden (code review i pull requests). Vi har også testet funksjonaliteten ved å kjøre backend og frontend og verifisert at ting fungerer som det skal.

14.2.4 Akseptansetester

Det har blitt utført en ekstern kundeakseptansetest i form av brukertester hvor veilederne våres hos Accenture er kundene som tester programvaren. Vi har laget et spørreskjema med kvantitative

og kvalitative spørsmål for å samle inn informasjon om brukervenneligheten til webapplikasjonen.

Figur 14.3 viser to spørsmål i akseptansetesten om brukervenneligheten i "My borrows" siden. Figuren inneholder et kvantitativt og et kvalitatittivt spørsmål. Hele brukertesten med tilbakemeldinger finnes i vedlegg H.

The figure shows a user test interface. At the top, there is a question: "Hvor brukervennlig var 'My borrows' siden? *". Below it is a horizontal scale from 1 to 10, with "Vanskelig" on the left and "Brukervennlig" on the right. The scale has 10 circles, each corresponding to a number from 1 to 10. Below the scale is a section titled "Feedback" containing the text "Lang svartekst".

Figur 14.3: To spørsmål i brukertesten

14.3 Testdekningsrapporter

En testdekningsrapport er generert og viser over 75% testdekning, som er målet gruppen hadde satt seg. Følgende figurer viser testdekningen av alle modulene i backend:

File		Statements	Branches	Functions	Lines
auth.js	[Green bar]	100%	33/33	100%	8/8
categories.js	[Green bar]	100%	34/34	87.5%	7/8
docs.js	[Green bar]	81.82%	9/11	100%	0/0
products.js	[Green bar]	100%	32/32	100%	6/6
rentals.js	[Green bar]	98.8%	82/83	96.15%	25/26
users.js	[Green bar]	100%	52/52	100%	12/12

Figur 14.4: Testing av API endepunkter

File ▾		Statements	Branches	Functions	Lines
admin.js	<div style="width: 87.5%;"><div style="width: 100px; height: 10px; background-color: #2e3436;"></div></div>	87.5%	7/8	75% 3/4	100% 1/1 100% 6/6
auth.js	<div style="width: 88.89%;"><div style="width: 100px; height: 10px; background-color: #2e3436;"></div></div>	88.89%	16/18	75% 6/8	100% 1/1 93.33% 14/15
error.js	<div style="width: 100%;"><div style="width: 100px; height: 10px; background-color: #2e3436;"></div></div>	100%	4/4	50% 1/2	100% 1/1 100% 3/3
requestLogger.js	<div style="width: 60%;"><div style="width: 100px; height: 10px; background-color: #f0e68c;"></div></div>	60%	3/5	50% 1/2	100% 1/1 50% 2/4
validateCategory.js	<div style="width: 79.17%;"><div style="width: 100px; height: 10px; background-color: #f0e68c;"></div></div>	79.17%	19/24	52.38% 11/21	50% 1/2 81.82% 18/22
validateObjectId.js	<div style="width: 100%;"><div style="width: 100px; height: 10px; background-color: #2e3436;"></div></div>	100%	5/5	100% 2/2	100% 1/1 100% 5/5
validateProduct.js	<div style="width: 100%;"><div style="width: 100px; height: 10px; background-color: #2e3436;"></div></div>	100%	11/11	100% 4/4	100% 1/1 100% 9/9
validateRental.js	<div style="width: 85.71%;"><div style="width: 100px; height: 10px; background-color: #2e3436;"></div></div>	85.71%	6/7	50% 1/2	100% 1/1 100% 6/6
validateUser.js	<div style="width: 100%;"><div style="width: 100px; height: 10px; background-color: #2e3436;"></div></div>	100%	7/7	100% 2/2	100% 1/1 100% 6/6

Figur 14.5: Testdekningsrapport av middleware

File ▾		Statements	Branches	Functions	Lines
category.js	<div style="width: 100%;"><div style="width: 100px; height: 10px; background-color: #2e3436;"></div></div>	100%	9/9	100% 0/0	100% 1/1 100% 9/9
product.js	<div style="width: 100%;"><div style="width: 100px; height: 10px; background-color: #2e3436;"></div></div>	100%	10/10	100% 0/0	100% 1/1 100% 10/10
rental.js	<div style="width: 95.65%;"><div style="width: 100px; height: 10px; background-color: #2e3436;"></div></div>	95.65%	22/23	50% 2/4	100% 4/4 100% 22/22
serviceMessage.js	<div style="width: 100%;"><div style="width: 100px; height: 10px; background-color: #2e3436;"></div></div>	100%	8/8	100% 0/0	100% 1/1 100% 8/8
suggestion.js	<div style="width: 100%;"><div style="width: 100px; height: 10px; background-color: #2e3436;"></div></div>	100%	8/8	100% 0/0	100% 1/1 100% 8/8
user.js	<div style="width: 100%;"><div style="width: 100px; height: 10px; background-color: #2e3436;"></div></div>	100%	19/19	100% 0/0	100% 4/4 100% 19/19

Figur 14.6: Testdekningsrapport av klassene

Del III

Avslutning

Kapittel 15

Videre utvikling

Typisk ved programvareutvikling er at et program aldri er ferdig, det må videreutvikles. Grunnen til dette er mangfoldige, det kan være endrede krav fra kunder, feil i koden, sikkerhetssårbarheter, forbedring av brukeropplevelsen, med mer. Gruppen fikk ikke utviklet alle funksjonaliteter som var tiltenkt ut ifra kravspesifikasjonen (se kapittel 16.1). I dette kapittelet diskutes funksjonalitet som kan videreutvikles og ny funksjonalitet som kan implementeres.

15.1 Videreutvikling basert på brukertest

I kapittel 6.4.5 ble det skrevet om en brukertest med tilhørende brukerundersøkelse hvor gruppen fikk konstruktive tilbakemeldinger. En gjennomgående tilbakemelding var designet, blant annet fargevalg. Til videreutvikling ville det således vært hensiktsmessig å engasjere en web-designer. Designeren kan forbedre fargevalget for å gjøre applikasjonen mer attraktiv og endre layouten for å forbedre helhetsinntrykket.

En annen tilbakemelding gruppen fikk var angående manglende feilmeldinger og informasjon noen steder i applikasjonen. F.eks får ikke administrator opp varsle om at det er ubehandlete låneforespørslar og må trykke seg inn på utlån på admin-dashboardet for å se disse. Dette kan enkelt løses ved å legge til et lite varsle med antall ubehandlete forespørslar ved siden av utlånsfanen. Når det kommer til feilmeldinger er dette ikke konsekvent over appen. Noen komponenter viser lasteikon og feilmelding dersom innlasting feiler, mens andre komponenter ikke viser noe i det hele tatt. Her gjenstår det litt arbeid for å gjøre applikasjonen mer konsistent. Det er laget gjenbrukbare komponenter for loadingikon og feilmelding, slik at disse kun trengs å implementeres der det mangler.

Videre ble det gitt kommentarer på strenge valideringsregler på visse steder hvor dette ikke er nødvendig. F.eks er det lagt inn minstekrav på 6 tegn på innsending av utstyrsforslag, noe som

ikke er nødvendig. Ekstra mellomrom på slutten av noen felt gir også feilmelding. En gjennomgang av valideringsregler over hele applikasjonen kan derfor være hensiktsmessig for å minke frustrasjon hos brukerne.

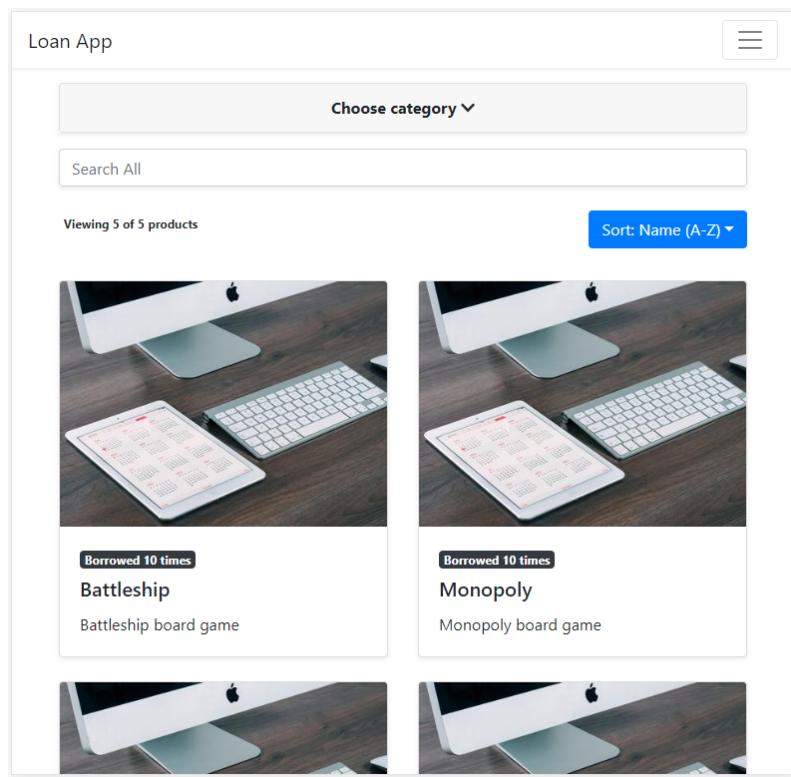
15.2 HTTPS til fordel for HTTP

Webapplikasjonen bruker for tiden Hyper Text Transfer Protocol(HTTP) for å overføre data. Problem med dette er at data sendes i klartekst mellom klient og server. Når en bruker registrerer seg, sendes brukernavnet og passordet med klartekst til serveren. Dette kan være sårbart for et Man in the middle-angrep, da det er først på serveren at passordet blir hashet og saltet. En ondsinnet aktør kan for eksempel bruke wireshark til å lytte til Wi-Fi trafikk og plukke opp passordet når en bruker registrerer seg.

Som mottiltak til dette bør man benytte HTTPS. Denne protokollen bruker en kombinasjon av assymetrisk og symmetrisk kyptering for å kryptere all kommunikasjon mellom klient og server. For å ta i bruk HTTPS trenger man et sertifikat [43].

15.3 Produktbilder

Muligheten for å legge til produktbilder i applikasjonen ble ikke implementert. Det brukes for tiden kun et tilfeldig bilde fra [picsum](#) for alle produktene (se figur 15.1). På produktsiden til admin-dashbordet bør administratoren kunne endre på bildet til et produkt. Det bør også være mulig å legge til et bilde samtidig som man legger til et nytt produkt. Denne funksjonaliteten vil være med på å forbedre brukeropplevelsen ved at det blir lettere å finne fram til produktet man er ute etter.



Figur 15.1: Validering av brukerinput i registrering av bruker

15.4 Notifikasjoner på telefon

For tiden er e-post den eneste måten å sende beskjed til brukere på. En mulig forbedring vil være å gi beskjeder via SMS. Telefonnummer blir allerede lagret med brukeren i databasen, slik at det kun trengs å koble på et API for sending av SMS.

15.5 Mobilapplikasjon

Som diskutert i delkapittel 4.1 muliggjør REST-APIet en løs kobling mellom klienter. Dette innebærer at man ikke er låst til en klientimplementasjon. Derfor er en naturlig videreutvikling å lage en mobilapplikasjon som tar i bruk APIet. Applikasjonen er allerede utviklet for å være mobilvennlig og skalerer passende til en mobil, men en mobilapplikasjon vil tillate å ta bruk flere funksjonaliteter som smarttelefoner byr på, blant annet push-notifikasjoner.

15.6 CAPTCHA

Det er foreløpig ikke implementert **CAPTCHA** i webapplikasjonen. CAPTCHA står for Completely Automated Public Turing Test To Tell Computers and Humans Apart. Det har som formål å beskytte nettsiden mot automatiske forespørsler gjort av datamaskiner. Dette gjøres ved å generere

og gradere en test som mennesker klarer å bestå, men ikke maskiner [7]. Typisk er slike tester at du skal lese et forvrengt ord og så skrive ordet i en tekstboks for å komme videre på nettsiden.

Det kan være fordelaktig å implementere dette på innloggingssiden ved flere mislykkede innloggingsforsøk for å forhindre brute force-angrep, samt på siden for brukerregistrering for å unngå at applikasjonen blir spammed med falske brukerprofiler.

15.7 Innloggingsforsøk og brukeraktivitet

Det ble ikke satt krav til begrensning eller logging av innloggingsforsøk av produkteier. Det er derfor mulig å forsøke et uendelig antall passordkombinasjoner slik applikasjonen står i dag. Gruppen foreslår at det implementeres en begrensning på tre innloggingsforsøk før en konto sperres og passord må resettes. Dette kan implementeres ved å lage et nytt felt i brukerkolleksjonen som teller antall mislykkede forsøk, og inkrementere denne i endepunktet for autentisering. Eventuelt kan det implementeres en tidssperr, slik at det tar lengre tid for hvert påfølgende, mislykkede forsøk.

En annen mulig løsning ville vært å ta i bruk Google sin nyeste utgave av CAPTCHA, reCAPTCHA v3. reCAPTCHA v3 returnerer en poengsum som genereres ut i fra brukerens interaksjon med nettsiden. Basert på denne poengsummen er det mulig å si noe om sannsynligheten for at interaksjonen er gjort av en maskin, for så å ta preventative tiltak basert på dette.

Tofaktorautentisering kan også implementeres flere steder, enten ved at bruker får sms for å bekrefte registrering av bruker men også for å få en kode ved innlogging eller kode for å resette passordet.

15.8 Dynamisk utlånsvarighet

Etter diskusjon med produkteier om dynamisk utlånsvarighet kom vi fram til at det ikke var en kritisk funksjonalitet. Derfor blir hvert utlån satt til syv dager fra datoén administratoren bekrefter utlånet. For å endre dette kreves det at dateToReturn-feltet settes ved innsending av låneforspørsler (POST /rentals), slik at admin kan se dette når han behandler lånesøknaden.

Brukeren kan da velge til-dato for utlånet, eller alternativt velge antall dager/uker brukeren ønsker å lånet produktet. Enda en videreutvikling kunne vært å tillate brukeren å sette en fra-dato på utlånet sitt. Dette gjør at brukere kan reservere produkter fram i tid.

15.9 Venteliste

Funksjonalitet for venteliste var lavt prioritert og ble ikke utviklet. Dette kan implementeres ved at brukere kan sette seg på venteliste og få en mail med en gang produktet blir tilgjengelig.

15.10 Forlenge lånefrist

En nyttig funksjonalitet som kan legges til er å gjøre det mulig for en brukere å forlenge utlånet sitt. Dette bør kombineres med funksjonalitet for venteliste, slik at det ikke er mulig å forlenge fristen dersom en annen bruker ønsker å låne produktet.

Kapittel 16

Diskusjon

16.1 Kravspesifikasjon og produkt

Kravspesifikasjonen vedlagt i vedlegg E inkluderte en liste med funksjonelle og ikke-funksjonelle krav som sluttproduktet skulle innfri. De fleste kravene har blitt møtt innenfor tidsrammen. Produktet har blitt utviklet med Scrum-metodikk, hvor brukerhistoriene/kravene ble prioritert for hvert utviklingssprint sammen med produkteier fra Accenture. Prioriteringen ble gjort etter en kostnads-nytte-analyse med hensyn på den estimerte tiden utviklingen av en funksjonalitet ville ta, mot verdien funksjonaliteten gir for brukeren.

Av de 13 funksjonelle kravene for brukeren innfri sluttproduktet 11. For administratoren er alle kravene innfridd. Kravene som sluttproduktet ikke møter er:

- Brukere kan forlenge lånetid på utstyr de har lånt.
- Brukere kan sette seg på venteliste for utlån.

Sluttproduktet tilfredsstiller 9 av de 10 ikke-funksjonelle kravene til webapplikasjonen. Kravet som ikke er innfridd her er:

- Applikasjonen forhindrer gjentatte innloggingsforsøk.

En mulig løsning til dette kravet er diskutert i kapittel 15.6 om videre utvikling.

16.2 Videre diskusjon

Gruppen har fått godt utbytte av bacheloroppgaven. Ved å kombinere fagkunnskaper fra forskjellige emner vi har hatt på universitetet, har vi utviklet en fullverdig webapplikasjon for første gang i

løpet av studiet. Det har vært en erfearingsrik prosess å ta et produkt fra idéfasen til en app som kan bli tatt i bruk av Accenture.

Vi har valgt å bruke en teknologi-stakk som ingen på gruppen hadde tidligere erfaring med. Med dette har vi fått bredere erfaring enn tidligere. Vi har sett hvordan kunnskaper innen andre språk og teknologier er nyttefulle når man lærer nye. Med bredere erfaring innen ulike teknologier er gruppen bedre rustet til å velge den teknologien som er best egnet for neste oppgave man skal løse.

Stakken vi har brukt har gitt oss fordelen med å kun måtte forholde oss til JavaScript, noe som har gjort utviklingen enklere. Ulempen er at det gikk mye tid til opplæring av de nye teknologiene. Det ble satt av rundt en måned, men dette tok lengre tid for noen av gruppemedlemmene grunnet jobbintervjuprosesser. En konsekvens av dette har vært at gruppen ikke fikk implementert all funksjonaliteten vi hadde sett for oss til å begynne med. Forslag til videreutvikling av appen basert på hva som ikke ble fullført er diskutert i kapittel 15.

Ved å benytte Javascript med Node.js i backend, har gruppen også erfart forskjellen mellom et statisk språk som Java og et dynamisk språk som Javascript. Et dynamisk språk kompileres ikke, noe som gjør at feil ofte ikke blir oppdaget før ved kjøretid. Ved å slippe kompilering går kodingen raskere, mens det går mer tid til å feilsøke og rette på programmet, da variabeltyper ikke blir sjekket før ved kjøretid.

Da Accenture stengte kontorene sine grunnet Covid-19, gikk gruppen over til å arbeide hjemmefra. Vi forstatte Scrum-praksisen ved å ha daglige stand up-møter og møter med veiledere over nett. Disse daglige møtene ble raskt en av de få kommunikasjonskildene medlemmene hadde i løpet av arbeidsdagen. Dette har ført til mer inkonsistent arbeid enn tidligere. Det har blant annet ikke vært like enkelt å holde en konsekvent kodestruktur til tross for at vi sjekket hverandres arbeid gjennom pull requests. Ved å ha retrospektiv-møter mot slutten av sprintene adresserte vi noen av problemene som oppstod med den endrede arbeidsituasjonen. Det ble blant annet tatt tiltak ved å innføre et sekundært møte i løpet av arbeidsdagen, noe som økte motivasjonen i gruppen. Gruppen har også sett nytten av å bruke skjermdeling via [Teamviewer](#) for parprogrammering, og vi ser tydelig forskjell på kodekvaliteten fra moduler som er utviklet under parprogrammering, kontra moduler som vi har utviklet individuelt.

Et klart forbedringspotensiale for applikasjonen vår er kodekvaliteten til React-komponentene i frontend. Flere komponenter ble skrevet med ulikt abstraksjonsnivå ved at komponenter ble satt sammen av både lavnivå- og høynivåkode. Dette har gjort koden vanskeligere å lese. Figur 16.1 viser en komponent som kun er satt sammen av andre komponenter. Ved første øyekast kan en se at komponenten består av en filtrerings-, søkings-, produktvisnings-, paginerings- og en modalkomponent. Denne komponenten er svært ryddig. Figur 16.2 viser en komponent som er satt sammen av både en høynivå TableHeader-komponent og lavnivå HTML-element (tbody). I tillegg til å gjøre

komponenten unødvendig lang, må HTML-koden analyseres for å finne ut hva den faktisk gjør.

```
152 |     return (
153 |       <>
154 |         <FilterContainerV2
155 |           items={categories}
156 |           selectedItem={selectedCategory}
157 |           onItemSelected={this.handleCategorySelect}
158 |         />
159 |         <SearchBar
160 |           placeHolder={this.getSearchPlaceHolder()}
161 |           onChange={this.handleSearch}
162 |         />
163 |         <ProductView
164 |           isLoading={isLoading}
165 |           error={error}
166 |           products={products}
167 |           productCount={this.state.products.length}
168 |           sortProperties={this.state.sortProperties}
169 |           selectedSortProperty={this.state.selectedSortProperty}
170 |           onSortSelect={this.handleSortSelect}
171 |         />
172 |         <Pagination
173 |           className="mt-3"
174 |           itemsCount={this.state.products.length}
175 |           pageSize={pageSize}
176 |           currentPage={currentPage}
177 |           onPageChange={this.handlePageChange}
178 |         />
179 |
180 |         <SuggestionModal />
181 |       </>
182 |     );
183   }
184 }
```

Figur 16.1: Eksempel på en ryddig React-komponent

I retrospektiv kunne gruppen hatt nytte av å ha større mangfold av teknisk interesse og erfaring. De fleste på gruppen hadde erfaring innen backend og ønsket å jobbe med dette. Vi ser at det hadde vært hensiktsmessig å ha minst ett medlem som spesialiserte seg i frontend-utvikling, brukerinteraksjon og design. Dette gjenspeiles i tilbakemeldingene fra brukertestene, hvor designet fikk lavest score.

```
92
93     return (
94       <>
95       {processedReturns && processedReturns.length > 0 ? (
96         <Table className="mt-3" bordered hover responsive>
97           <TableHeader
98             columns={this.columns}
99             sortColumn={sortColumn}
100            onSort={this.handleSort}
101          >/TableHeader>
102          <tbody>
103            {processedReturns.map((currReturn) => {
104              return (
105                <tr key={currReturn._id}>
106                  <td>{currReturn.user.name}</td>
107                  <td>{currReturn.product.name}</td>
108                  <td>{currReturn.product.entity.identifier}</td>
109                  <td>{currReturn.dateReturned}</td>
110                  <td>{currReturn.remarks}</td>
111                  <td>{currReturn.confirmedReturned.toString()}</td>
112
113                  {!currReturn.confirmedReturned && (
114                    <td>
115                      <Button
116                        block
117                        onClick={() => {
118                          this.setCurrentReturn(currReturn);
119                          this.handleModalButtonClicked();
120                        }}
121                      >
122                        | Confirm Return
123                      </Button>
124                    </td>
125                  )}
126                </tr>
127              );
128            )})
129          </tbody>
130        </Table>
```

Figur 16.2: Eksempel på en mindre ryddig komponent

Kapittel 17

Oppsummering

Målet med denne oppgaven har vært å utvikle en applikasjon for utleie av utstyr for Accenture sin spillgruppe. Gruppen har tatt i bruk prosessmodellen Scrum for gjennomføring av arbeidet. En backlog med funksjonaliteter som skal implementeres har blitt prioritert i samarbeid med produkteier, og gruppen har med dette utviklet og levert en webapplikasjon utviklet med React, Node.js, Express og MongoDB.

Applikasjonen tilbyr både brukerfunksjonalitet og administratorfunksjonalitet. Utleie og administrasjon av eksisterende og nytt utstyr er applikasjonens hovedområde. Appen vil kunne gi en bedre brukeropplevelse og øke aktiviteten i spillgruppen. Medlemmer kan finne og låne tilgjengelig utstyr på en enklere måte, og gruppeledere vil i mindre grad være avhengig av manuelt arbeid for å utføre spillgruppens funksjoner. Under en brukerundersøkelse foretatt av veilederne, fikk gruppen gode tilbakemeldinger på applikasjonen.

Produktet møter de fleste kravene definert i kravspesifikasjonen. Forslag til utvikling av manglende krav, samt andre anbefalninger til videre utvikling, er diskutert under kapittel 16.2. Ytterligere brukertester vil kunne sette lys på flere forbedringer som kan gjøres i applikasjonen.

Avslutningsvis kan det sies at bachelorgruppen har fått stort utbytte av oppgaven. Ved å kombinere tidligere kunnskaper og lære og ta i bruk nye rammeverk og biblioteker har det blitt utviklet enn fullverdig applikasjon hvor frontend og backend kommuniserer via et REST-API. Dette har etterhvert blitt en svært kjent arkitektur i bransjen, og vil være en verdifull erfaring å ha med videre inn i arbeidslivet.

Bibliografi

- [1] *A single protective technology means a single point of failure.* 2017. URL: <https://www.welivesecurity.com/2017/05/02/single-protective-technology-means-single-point-failure/>.
- [2] *About - Branching and Merging.* URL: <https://git-scm.com/about/branching-and-merging>.
- [3] *Acceptance testing.* 2019. URL: <http://softwaretestingfundamentals.com/acceptance-testing/>.
- [4] *API.* URL: <https://snl.no/API>.
- [5] *Backend.* URL: <https://snl.no/backend>.
- [6] *Bcrypt.* 2020. URL: <https://en.wikipedia.org/wiki/Bcrypt>.
- [7] *CAPTCHA: Telling Humans and Computers Apart Automatically.* URL: <http://www.captcha.net/>.
- [8] Kristina Chodorow. *MongoDB the definitive guide.* O'reilly, 2013.
- [9] CISA. *Security tips.* URL: <https://www.us-cert.gov/ncas/tips/ST04-002>.
- [10] E. F. Codd. «A relational model of data for large shared data banks». I: (1970).
- [11] *Cryptographic Hash Functions Explained: A Beginner's Guide.* 2018. URL: <https://komodoplatform.com/cryptographic-hash-function/>.
- [12] Database. *Knuth: Computers and Typesetting.* URL: <https://snl.no/databas>.
- [13] *Definisjon av interoperabilitet.* URL: <http://interoperability-definition.info/no/>.
- [14] *git.* URL: <https://git-scm.com/>.
- [15] *GitHub features: the right tools for the job.* URL: <https://github.com/features>.
- [16] Thomas Gramstad. *åpen kildekode.* URL: https://snl.no/pen_kildekode.
- [17] *HTML Web Storage API.* URL: https://www.w3schools.com/html/html5_webstorage.asp.

- [18] *Integration Testing: What is, Types, Top Down & Bottom Up Example.* 2020. URL: <https://www.guru99.com/integration-testing.html>.
- [19] *Introducing JSX.* URL: <https://reactjs.org/docs/introducing-jsx.html>.
- [20] *Introduction to JWT.* URL: <https://jwt.io/introduction/>.
- [21] *Introduction to mongoDB.* URL: <https://docs.mongodb.com/manual/introduction/>.
- [22] *Jest - Delightful JavaScript Testing.* 2020. URL: <https://jestjs.io/>.
- [23] *Manifesto for Agile Software Development.* URL: <https://agilemanifesto.org/>.
- [24] Robert Cecil. Martin. *Clean code: a handbook of agile software craftsmanship.* Prentice Hall, 2009.
- [25] *Principles behind the agile manifesto.* URL: <https://agilemanifesto.org/principles.html>.
- [26] *React (web framework).* 2020. URL: [https://en.wikipedia.org/wiki/React_\(web_framework\)](https://en.wikipedia.org/wiki/React_(web_framework)).
- [27] *RESTful Web Services Tutorial with Example.* URL: <https://www.guru99.com/restful-web-services.html>.
- [28] *Security: Identification, Authentication, and Authorization.* 2005. URL: <https://danielmiessler.com/blog/security-identification-authentication-and-authorization/>.
- [29] Sendgrid. *SMTP Relay Service 101 [Back to Basics].* 2018. URL: <https://sendgrid.com/blog/smtp-relay-service-basics/>.
- [30] Mark Smallcombe. *SQL vs. NoSQL: How Are They Different and What Are the Best SQL and NoSQL Database Systems?* 2019. URL: <https://www.xplenty.com/blog/the-sql-vs-nosql-difference/>.
- [31] Ian Sommerville. *Software Engineering.* Global edition. Pearson, 2015, s. 105–111. ISBN: 9783827370013.
- [32] *SuperTest Github repository.* 2020. URL: <https://github.com/visionmedia/supertest#readme>.
- [33] *System Testing.* 2020. URL: <https://www.guru99.com/system-testing.html>.
- [34] *Understanding Rainbow Table Attack.* URL: <https://www.geeksforgeeks.org/understanding-rainbow-table-attack/>.
- [35] *User Stories with Examples and Template.* URL: <https://www.atlassian.com/agile/project-management/user-stories>.
- [36] *Version Control Systems.* 2019. URL: <https://www.geeksforgeeks.org/version-control-systems/>.

- [37] *What are Web Services? Architecture, Types, Example.* 2020. URL: <https://www.guru99.com/web-service-architecture.html>.
- [38] *What does REST's Client-Server mean now?* 2012. URL: <http://byterot.blogspot.com/2012/07/what-does-rests-client-server-mean-now.html>.
- [39] *What is Agile Software Development?* 2020. URL: <https://www.agilealliance.org/agile101/>.
- [40] *What is API Documentation?* 2019. URL: <https://swagger.io/blog/api-documentation/what-is-api-documentation-and-why-it-matters/>.
- [41] *What is Automated Testing?* 2020. URL: <https://smartbear.com/learn/automated-testing/what-is-automated-testing/>.
- [42] *What is BLACK Box Testing? Techniques, Example & Types.* URL: <https://www.guru99.com/black-box-testing.html>.
- [43] *What is HTTPS.* URL: <https://www.cloudflare.com/learning/ssl/what-is-https/>.
- [44] *What is Layered Security?* URL: <https://www.ericom.com/whatis/layered-security/>.
- [45] *What is Software Testing? Introduction, Definition, Basics & Types.* URL: <https://www.guru99.com/software-testing-introduction-importance.html>.
- [46] *What is Unit Testing?* 2020. URL: <https://smartbear.com/learn/automated-testing/what-is-unit-testing/>.
- [47] *What is WHITE Box Testing? Techniques, Example, Types & Tools.* URL: <https://www.guru99.com/white-box-testing.html>.
- [48] *Why Manual Testing Is Important — Benefits Of Manual Testing.* 2019. URL: <https://medium.com/habilelabs/why-manual-testing-is-important-benefits-of-manual-testing-4cd995e01dab>.

Vedlegg

Tillegg A

Gruppekontrakt

Gruppekontrakt - Bachelorgruppe

Målsetning og gjennomføring

§1.1 Oppgavens mål er at gruppens medlemmer skal få erfaring innen smidig programvareutvikling og lære nye verktøy innen dette.

§1.2 Gruppen skal ha det gøy underveis, men det er forventet at alle medlemmer er initiativtakende og bidrar til prosjektets fremdrift.

§1.3 Kommunikasjon vil hovedsakelig foregå gjennom Slack, som gruppen har blitt enige om ved oppstart av prosjektet.

§1.4 Gruppens medlemmer følger til enhver tid arbeidsmetodikken (Scrum eller Kanban) som det er kommet til enighet om slik at alle medlemmer har oversikt over hverandres oppgaver.

§1.5 Gruppemedlemmene fører prosjektdagbok som spesifiserer hva vedkommende har jobbet med den dagen.

Arbeidstid og arbeidssted

§2.1 Medlemmene arbeider med oppgaven primært fire dager i uken.

§2.2 Arbeidstid er primært fra 09:00 - 16:00, men kan justeres etter behov.

§2.3 Arbeidssted er ved Accenture sine kontorer på Fornebu, men kan endres ved behov.

Forsentkomming og sykdom

§3.1 Medlemmene plikter å gi beskjed via avtalt kommunikasjonskanal ved forsentkomming.

§3.2 Gruppen følger arbeidsmiljølovens bestemmelser om 3 dagers egenmelding. Ved spørsmål om legitimitet, plikter vedkommende å fremskaffe legeerklæring.

Pause

§4.1 Gruppemedlemmene administrerer selv pausetid på 30 minutter som kan deles i mindre pauser.

Konflikter og problemløsing

§5.1 Gruppen diskuterer fortløpende konflikter som oppstår underveis.

§5.2 Gruppemedlemmene plikter å gi beskjed dersom noe er til hinder for utføring av vedkommendes arbeidsoppgaver.

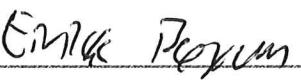
§5.3 Ved uenighet tas avgjørelser primært etter demokratisk avstemming.

§5.4 Dersom alle medlemmene er uenige kan tredjepart innblandes for rådføring. Tredjepart skal hovedsakelig være en eller flere av de tildelte veilederne.

Signaturer:


Rany Tarek Boueifm


Markus Hellestveit


Eirik Bøyum

Tillegg B

Framdriftsplan

WN 5 (27-02)
**Su
26**
**Mo
27**
**Tu
28**
**We
29**
**Th
30**
**Fr
31**
**Sa
01**
**Su
02**
**Mo
03**
**Tu
04**
**We
05**
**Th
06**
**Fr
07**
**Sa
08**
**Su
09**
**Mo
10**
**Tu
11**
**We
12**

Opplæring og kurs

27.01.2020 - 14.02.2020

TASK

✓ Kravspesifikasjon

✓ NodeJS/Express kurs

React kurs

Sprint 0 - Oppsett og ar...

17.02.2020 - 28.02.2020

TASK

Sprint 1 - Utvikling

02.03.2020 - 20.03.2020

Sprint 2 - Utvikling

23.03.2020 - 10.04.2020

Sprint 3 - Utvikling

13.04.2020 - 01.05.2020

Sprint 5 - Design focus

04.05.2020 - 10.05.2020

Presentasjon for styrin...

06.02.2020 - 12.02.2020

Dokumentasjon

27.01.2020 - 25.05.2020

Presentasjon

18.05.2020 - 07.06.2020

MARCH 2020

WN 8 (17-23)

Th
13
Fr
14

Sa
15
Su
16

Mo
17
Tu
18

We
19
Th
20

Sa
21
Su
22

WN 9 (24-01)

Mo
24
Tu
25

We
26
Th
27

Fr
28
Sa
29

WN 10 (02-08)

Su
01
Mo
02

Tu
03
We
04

Th
05
Fr
06

Sprint 0 - Oppsett og arkitektur

- ✓ DB Arkitektur
- ✓ Backend arkitektur
- ✓ Frontend arkitektur
- ✓ Skeleton project

Sprint 1 - Utvikling

WN 11 (09-15)

Sa
07

Su
08 Mo
09

Tu
10

We
11

Th
12

Fr
13

Sa
14

Su
15 Mo
16

Tu
17

We
18

Th
19

Fr
20

Sa
21

Su
22 Mo
23

Tu
24

We
25

Th
26

Fr
27

Sa
28

WN 12 (16-22)



WN 13 (23-29)

Sprint 2 - Utvikling

APRIL 2020

WN 14 (30-05)

Su
29
Mo
30

Tu
31

We
01

Th
02

Fr
03

Sa
04

Su
05

Mo
06

Tu
07

We
08

Th
09

Fr
10

Sa
11

Su
12

Mo
13

Tu
14

We
15

Th
16

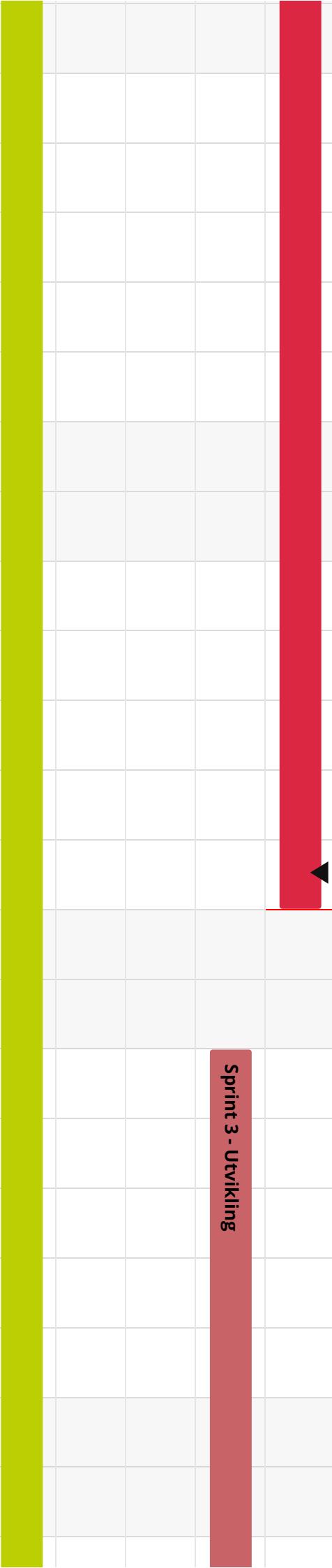
Fr
17

Sa
18

Su
19

WN 17 (20)

Sprint 3 - Utvikling



MAY 2020

WN 18 (27-03)

Mo
20

Tu
21

We
22

Th
23

Fr
24

Sa
25

Su
26

Mo
27

Tu
28

We
29

Th
30

Fr
01

Sa
02

Su
03

Mo
04

Tu
05

We
06

Th
07

Fr
08

Sa
09

Su
10

Mo
11

Tu
12

WN 19 (04-10)

Mo
04

Tu
05

We
06

Th
07

Fr
08

Sa
09

WN 20 (11-17)

Mo
11

Tu
12

Sprint 5 - Design focus



JUNE 2020

WN 23 (01-07)

We
01
Tu
02
We
03

Sa
30

Su
31
Th
29
Fr

Mo
25
Tu
26
We
27
Th
28
Fr

Sa
23

Su
21
Th
22
Fr

We
20

Tu
19
Mo
18

Sa
17
Su
16
Mo
15

We
13
Th
14

WN 21 (18-24)

Sa
16

Su
17

Mo
18

Tu
19

We
20

Th
21

Fr

Sa
23

Su
24
Mo
25

Tu
26

We
27

Th
28

Fr

Sa
30

Su
31
Mo
01

Tu
02

We
03

Presentasjon

Th
04

Fr
05

Sa
06

Tillegg C

Risikoanalyse

Risikoanalyse før tiltak

Risiko	Beskrivelse	Sannsynlighet	Konsekvens	Risikofaktor	Tiltak
1	Langvarig sykdom hos en av gruppemedlemmene slik at de andre medlemmene må ta arbeidet	1	5	5	Alle gruppemedlemmene har oversikt over hele stacken
2	Kortvarig sykdom hos en av gruppemedlemmene	3	2	6	Alle gruppemedlemmene har oversikt over hele stacken
3	Endrede krav underveis	4	4	16	Jobbe smidig med sprinter slik at endrede krav blir enklere å håndtere
4	Tap av prosjektdata	4	5	20	Bruke GitHub for å lagre prosjekt og versjonsstyring, Bruke skytjenester for rapport og prosjektdokumenter
5	For lite tid til å fullføre produktet	3	4	12	Jobbe i sprinter hvor prioritert funksjonalitet utvikles først
6	Misforståelse mellom oppdragsgiver og bachelorgruppen fører til feil sluttprodukt	4	4	16	Brukertester og sprint reviews
7	Utfordrende å vite hvilke teknologier som burde brukes/læres til oppgaven	2	3	6	God kommunikasjon med veiledere, utvikling av kravspesifikasjon gir bedre oversikt
8	Utfordrende å lære nye teknologier	4	4	16	Opplæringsfase før start av utvikling

Risikomatrise før tiltak

Konsekvens/ Sannsynlighet	1 Ufarlig	2 En viss	3 Farlig	4 Kritisk	5 Svært kritisk
5 Svært sannsynlig					
4 Meget sannsynlig				3	4, 6, 8
3 Sannsynlig		2		5	
2 Mindre sannsynlig			7		
1 Lite sannsynlig					1

Risikoanalyse etter tiltak

Risiko	Beskrivelse	Sannsynlighet	Konsekvens	Risikofaktor	Tiltak
1	Langvarig sykdom hos en av gruppemedlemmene slik at de andre medlemmene må ta arbeidet	1	3	3	Alle gruppemedlemmene har oversikt over hele stacken
2	Kortvarig sykdom hos en av gruppemedlemmene	3	1	3	Alle gruppemedlemmene har oversikt over hele stacken
3	Endrede krav underveis	4	2	8	Jobbe smidig med sprinter slik at endrede krav blir enklere å håndtere
4	Tap av prosjektdata	1	5	5	Bruke GitHub for å lagre prosjekt og versjonsstyring, Bruke skytjenester for rapport og prosjektdokumenter
5	For lite tid til å fullføre produktet	2	2	4	Jobbe i sprinter hvor prioritert funksjonalitet utvikles først
6	Misforståelse mellom oppdragsgiver og bachelorgruppen fører til feil sluttprodukt	2	4	8	Brukertester og sprint reviews
7	Utfordrende å vite hvilke teknologier som burde brukes/læres til oppgaven	1	2	2	God kommunikasjon med veiledere, utvikling av kravspesifikasjon gir bedre oversikt
8	Utfordrende å lære nye teknologier	3	2	6	Opplæringsfase før start av utvikling

Risikomatrise etter tiltak

Konsekvens/ Sannsynlighet	1 Ufarlig	2 En viss	3 Farlig	4 Kritisk	5 Svært kritisk
5 Svært sannsynlig					
4 Meget sannsynlig					
3 Sannsynlig	2	8			
2 Mindre sannsynlig		5		6	
1 Lite sannsynlig		7	1		4

Tillegg D

Round Robin

CHALLENGE STATEMENT

Hvordan forbedre dagens lærings for admin og bruker?

FOLD TO DOTTED LINE

PROPOSED SOLUTION

Come up with an unconventional way to address the challenge.

- Lage en felles plattform (nettsted/app)
- 1) -Plattfermen må være intuitiv og enkelt å førestå.
- 2) -Gjør den lett tilgjengelig for folk i Accutur.
- 3) -Ta med galleri om nye spill etc fra bruker.
- 4) -Ha god oversikt over hva som finnes av spill os hvern som lærer et og til hver leser.

FOLD TO DOTTED LINE

WHY THE SOLUTION WILL FAIL

Review the proposed solution, and find a reason that it will fail.

This is your chance to be the armchair critic!

Lære design gjennom å analysere populære nettsteder.

- ~~• Hva er jobb:~~
- 1) • Vanligvis å få til uten designer
 - 2) • Hvordan skal den bli lett tilgjengelig? Accutur sender allerede ut veldig mye informasjon på mail.
→ den kan fort drabne i alt annet.
 - 3) • Leder til mer jobb for admin. Bedre nå, hvor ledergruppa bare kan bestemme selv hva som skal hjelpes.
 - 4)

FOLD TO DOTTED LINE

1. Vi kan lære oss design gjennom bøker / Udemy / lære av andre nettsider
2. Lage en mobilapp med pop-up notifications
3. Lage enkelt grensesnitt for admin slik at han/hun enkelt kan prosesere ønsker



CHALLENGE STATEMENT

Lage en app som er lett å
overta

PROPOSED SOLUTION

Come up with an unconventional way to address the challenge.

FOLD TO DOTTED LINE

- 1 - Test-driven development
- 2 - Følge/lagre en kodestandard
- 3 - Bruke gode variabelnavn
- 4 - Skrive dokumentasjon underveis
- 5 - Lage modeller før utvikling
- 6 - Bruke eksisterende Pakker/biblioteker
- 7 - Bruke Github og committe ofte

FOLD TO DOTTED LINE

WHY THE SOLUTION WILL FAIL

Review the proposed solution, and find a reason that it will fail.

This is your chance to be the armchair critic!

- 1 - For mye jobb / test
- 2 - Feil Kodestandard
- 3 - Hva er gode variabelnavn?
- 4 - Mye ekstra jobb
- 5 - Unedvendig teknikk
- 6 - Lite kontroll over scenar i bibliotekene og deres standard
- 7 - Det kommer til å gå salt. Feil Kode blir oversett etc...

FOLD TO DOTTED LINE

FINAL CONCEPT

Review the critique. Then, quickly generate an idea that resolves the issues raised.

Test-driven development går bare kjøye litt sent, -
et alternativ er å konsekvent skrive til hørende
tester når man legger til ny funksjonalitet.
Vi vil bruke gode variabelnavn som er forståelige
for en utenforstående. Konsekvent på bruk av
norsk eller engelsk. Vi burde lage en plan for
pår og hvordan vi skal lage systemdokumentasjon.
Github kan være et godt verktøy.



LUMA INSTITUTE

CHALLENGE STATEMENT

Hvor kan lage en app som er lett å overta?

PROPOSED SOLUTION

Come up with an unconventional way to address the challenge.

1. • Tenk gjennom arkitekturen før man begynner å kode.
2. • Bruk Pull Requests - la andre godkjenne koden din før du merger.
3. • Brak selvforskelende variablene.
4. • Lag støtbebyg systemdokumentasjon.
5. • Tester. Luk ut enkle feil med enhetstester.

FOLD TO DOTTED LINE

WHY THE SOLUTION WILL FAIL

Review the proposed solution, and find a reason that it will fail.

This is your chance to be the armchair critic!

1. IKKE bruk for mye tid på modellering og arkitektur i starten. Krav endrer seg konstantlig
2. +
3. + Ikke bruk kommentarer for å kompensere for dårlig kode/Mangling
4. Se punkt 1.
5. Etter enhetstester og integreringstester. Kunstige T-D-D, siden JS er svært dynamic

FOLD TO DOTTED LINE

FINAL CONCEPT

Review the critique. Then, quickly generate an idea that resolves the issues raised.

1. Lag en overordnet arkitektur før man koder.
2. Bruk pull requests og feature branching.
3. Bruk gode variablene fremfor kommentarer.
4. Lag dokumentasjon underveis, ikke på slutten.
5. Skriv enhetstester der mulig og integrasjontester der enhetstester er vanskelig å fø til.

FOLD TO DOTTED LINE



LUMA INSTITUTE

CHALLENGE STATEMENT

Skape gode brukeropplevelser

PROPOSED SOLUTION

Come up with an unconventional way to address the challenge.

- 1 Samle brukerstatistikk for å forbedre brukeropplevelsen over tid

- 1 - Designe dummy templates og forbered deg med brukere av applikasjonen.
- 2 - Lese bøker om UI, UX og DE
- 3 - Sette opp en common lowest denominator Bestemoren din f.eks
- 4 - Bruke SPA for raskere client routing
- 5 - Hente inspirasjon fra populære webvirksomheter
- 6 - Udemy courses.

WHY THE SOLUTION WILL FAIL

Review the proposed solution, and find a reason that it will fail.

This is your chance to be the armchair critic!

- 2: Orker ikke lese bøker - Tid!
- 3: Bestemoren min skal ikke bruke dette.
- 4: Hvis det ikke går annet å dele link til utstyr suger dette!
- 5: Don't be a copy-cat!
- 7: Statistikken er kjedelig...

FINAL CONCEPT

Review the critique. Then, quickly generate an idea that resolves the issues raised.

2. tutorial
3. Test mot faktiske brukere.
4. Implementerer bedre lesvinkler
5. Gi bruker undersøkelse
7. Samle kun essensiell statistikk
2. 1.
- 6.

Tillegg E

Kravspesifikasjon

Kravspesifikasjon - Webapplikasjon for spillgruppen

Rany Tarek Bouorm

Markus Hellestveit

Eirik Bøyum

14. Februar 2020

Forord

Kravspesifikasjonen beskriver de kravene oppdragsgiver setter til prosjektet. Dette omfatter både funksjonelle krav, hva systemet skal gjøre, og ikke funksjonelle krav, hvordan systemet skal implementere de funksjonelle kravene. Kravene er utarbeidet i samarbeid med oppdragsgiver og fungerer som en kontrakt mellom partene.

Kravspesifikasjonen inkluderer bakgrunnen for oppgaven, beskrivelse av systemet som skal utvikles samt krav til systemet oppdelt i funksjonelle og ikke-funksjonelle krav.

Innhold

1 Innledning	4
1.1 Om Accenture	4
1.2 Om gruppen	4
1.3 Veiledere	5
1.3.1 Veileder fra OsloMet	5
1.3.2 Veildere og produkteier fra Accenture	5
1.3.3 Kontaktpersoner fra Accenture	5
2 Bakgrunn for oppgaven	7
3 Systembeskrivelse	8
4 Krav	9
4.1 Funksjonelle krav	9
4.1.1 Funksjonelle krav for bruker	9
4.1.2 Funksjonelle krav for administrator	10
4.2 Ikke-funksjonelle krav	10

Kapittel 1

Innledning

1.1 Om Accenture

Accenture er et stort internasjonalt selskap som tilbyr tjenester innen strategi, teknologi, konsultasjon, operasjon, digitalisering og sikkerhet. Det har 492 000 ansatte på verdensbasis i tilsammen 52 land fordelt på 200 byer. I Norge har Accenture 1100 ansatte og har hovedkontor på Fornebu. Tonje Sandberg pr Januar 2020 er administrerende direktør i Norge.

1.2 Om gruppen

Gruppen består av tre studenter fra OsloMet:

- Rany er 25 år gammel og går dataingeniør-studiet. Han har også en ingeniørgrad innen maskin fra tidligere som han bringer med seg kunnskaper fra. Han har hovedsaklig IT-kunnskaper innen HTML, CSS, Javascript, Java og PHP.
- Eirik har flere interesser og har vært innom både økonomi, politikk og nanoteknologi. Til tross for godt læringsutbytte innenfor disse emnene har han alltid vendt tilbake til teknologi og brenner for IT. Han er alltid interessert i å tilordne seg den nyeste kunnskapen innenfor området og jobber for tiden med et prosjekt som innebærer maskinlæring.
- Markus er 25 år gammel og går dataingeniør-studiet. Han har erfaring med desktop applikasjonsutvikling i Java og C#, i tillegg til språk som Javascript, PHP, HTML, CSS.

1.3 Veiledere

Gruppen har fått tildelt en veileder fra OsloMet og to veiledere fra Accenture, hvorav en av disse også er leder av spillgruppen og produkteier for prosjektet.

1.3.1 Veileder fra Oslomet

Qian Meng

- **Tittel:** førsteamannusis
- **Epost:** qianmeng@oslomet.no
- **Telefon:** +4741393020

1.3.2 Veildere og produkteier fra Accenture

Adam Asskali - Produkteier

- **Tittel:** Software Engineer
- **Epost:** adam.asskali@accenture.com
- **Telefon:** 47449187

Mira Lilleholt Vik

- **Tittel:** Application Development Analyst
- **Epost:** mira.lilleholt.vik@accenture.com
- **Telefon:** 97466436

1.3.3 Kontaktpersoner fra Accenture

Daniel Meinecke

- **Tittel:** Functional Solution Designer
- **Epost:** daniel.meinecke@accenture.com
- **Telefon:** 97672525

Kjell Olav Dale

- **Tittel:** Tech Arch Delivery Specialist

- **Epost:** kjell.olav.dale@accenture.com

- **Telefon:** 95947459

Marius Torsrud

- **Tittel:** Systems Analyst

- **Epost:** marius.torsrud@accenture.com

- **Telefon:** 91854004

Kapittel 2

Bakgrunn for oppgaven

Accenture har idag en interessegruppe kalt “spillgruppa” med ca. 50 medlemmer. Spillgruppa holder arrangementer og leier ut spillutstyr som konsoller og brettspill til ansatte i Accenture.

I dag har spillgruppa én leder og to underledere. Arrangementer avtales i Microsoft Teams, og uteleie av utstyr håndteres manuelt av gruppens leder, f.eks ved bruk av notat-app på telefonen.

Ledere av spillgruppen opplever mye manuelt arbeid i forbindelse med håndtering av utstyr, og ønsker å utvikle en web-basert løsning for å automatisere arbeidet.

Kapittel 3

Systembeskrivelse

Det skal utviklens en webapplikasjon hvor brukere kan låne ut utstyr til spill-evenenter. Brukeren skal kunne søke seg frem og ha oversikt over tilgjengelig spill samt sette seg på venteliste dersom spillet allerede er uteid. En administrator skal kunne godkjenne disse forespørslene i webapplikasjonen. Videre skal administrator ha administrative funksjoner slik som oversikt over inventar og deres tilstand og oversikt over utlån. For å benytte seg av webapplikasjonene må administrator og brukere logge seg på. Det skal implementeres rollebasert tilgangsstyring for administratorer, slik at noen kan ha flere rettigheter enn andre.

Kapittel 4

Krav

Kravene til et system er en beskrivelse av tjenestene et system tilbyr. Kravene blir vanligvis utformet i samarbeid med kunden og utviklere.

Systemkrav blir ofte klassifisert som enten funksjonelle eller ikke-funksjonelle krav.

4.1 Funksjonelle krav

Funksjonelle krav beskriver tjenestene systemet skal tilby, hvordan systemet skal oppføre seg i bestemte situasjoner og hvordan det skal reagere på system- eller bruker-interaksjon. Funksjonelle krav kan også eksplisitt definere hva et system **ikke** skal gjøre [1].

4.1.1 Funksjonelle krav for bruker

- Webapplikasjonen skal la brukere med tilgang til Accenture-epost registrere seg på siden.
- Registrerte brukere skal kunne logge seg inn.
- Brukere skal få oversikt over inventar når de er innlogget.
- Brukere kan søke i inventar
- Applikasjonen skal automatisk sende påminnelser om innleveringsfrist.
- Brukere kan sende forespørsel om å låne spillutstyr.
- Brukere kan se oversikt over sine lån.
- Brukere kan sette seg på venteliste for utlån.

- Brukere kan sende inn ønske om hvilket utstyr spillgruppen skal kjøpe inn.
- Brukere kan endre passord.
- Brukere kan få tilsendt epost for å opprette nytt passord.
- Brukere kan forlenge lånefrist på utstyr de har lånt
- Webapplikasjonen skal tillate rapportering fra bruker om feil/mangler på utstyr ved lån eller ved retur.

4.1.2 Funksjonelle krav for administrator

- Registrerte administratorer kan logge seg inn.
- Administratoren kan godkjenne eller avslå forspørser om utlån av utstyr.
- Administrator skal få oversikt over utlån.
- Administrator skal kunne legge til og fjerne inventar.
- Registrerte administratorer kan endre rollen til andre brukere til administrator.
- Administrator har mulighet til å utestenge registrerte brukere.
- Administrator kan se statistikk over utlånt inventar.

4.2 Ikke-funksjonelle krav

Ikke-funksjonelle krav er begrensninger på de tjenestene systemet tilbyr. Istedentfor for krav til individuelle funksjoner, retter ikke-funksjonelle krav seg til helheten i systemet. Eksempler på ikke-funksjonelle krav er krav som omhandler hastighet, sikkerhet og pålitelighet. [1]

- Webapplikasjonen skal ha et intuitivt brukergrensesnitt.
- Kun ansatte i Accenture skal ha tilgang til å registrere seg i systemet.
- Applikasjonen forhindrer gjentatte innloggingsforsøk.
- Flere brukere skal kunne bruke webapplikasjonen samtidig.
- Flere administratorer skal kunne bruke webapplikasjonen samtidig.
- Passordet må bestå av store og små bokstaver samt tall.
- Webapplikasjonen skal fungere på PC og mobil, uavhengig av platform.

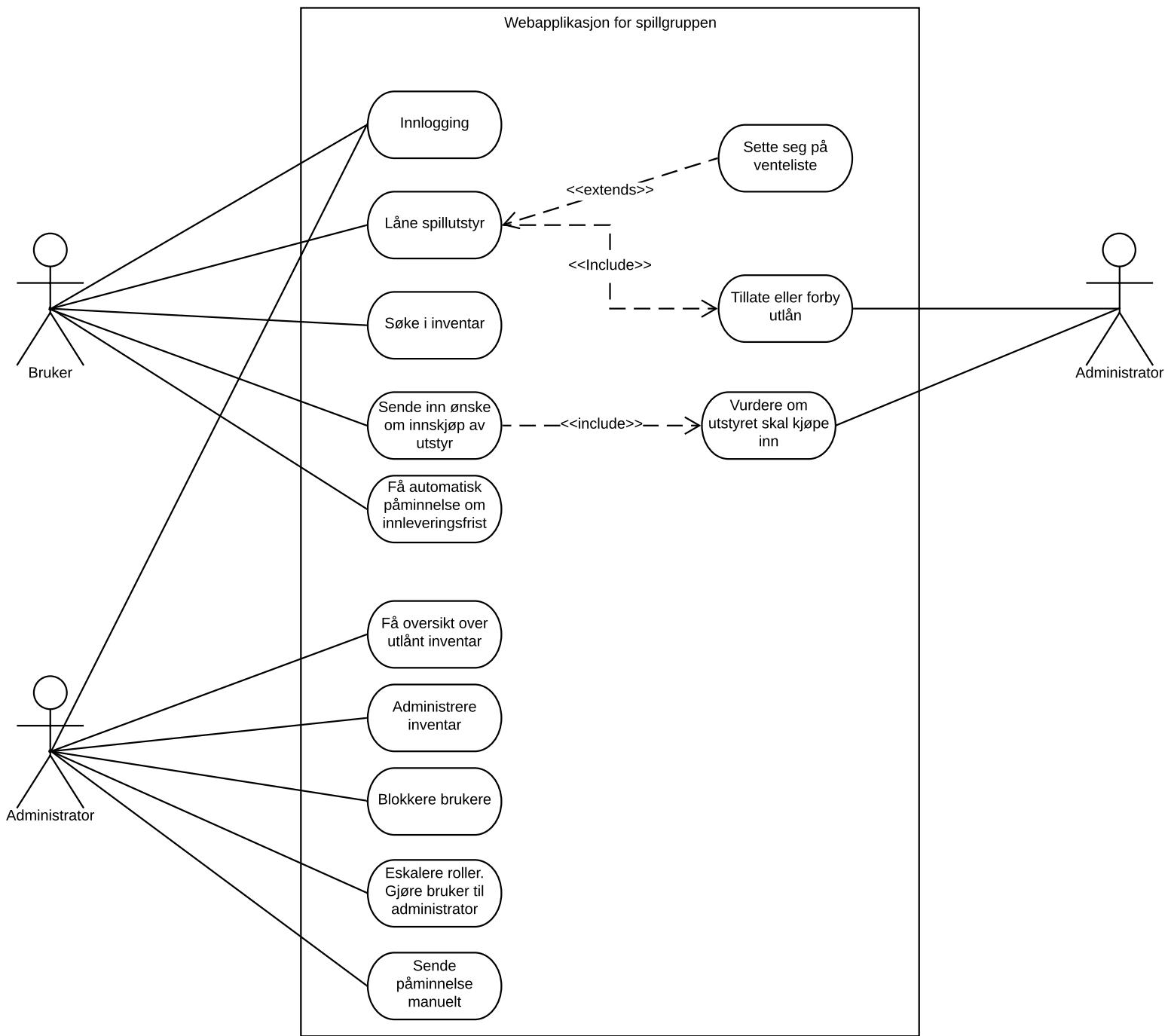
- Applikasjonen skal følge gjeldende personvernlover (GDPR).
- Kodebasen til applikasjonen skal skrives på engelsk.
- Dokumentasjon til RESTful-API skal skrives på engelsk.

Bibliografi

- [1] Ian Sommerville. *Software Engineering*. Global edition. Pearson, 2015, s. 105–111. ISBN: 9783827370013.

Tillegg F

Use-case diagram



Tillegg G

Swagger dokumentasjon



Loan-app API

1.0.0

OAS3

Servers

<http://localhost:3900/api> ▾

Authorize



Categories



GET

/categories/{id} Returns a category with the given id



Use to get a category by id

Parameters

Try it out

Name

Description

id * required

string
(path)

Valid mongoose Object Id

5e539c042f9afe16e09ca244

Responses

Code

Description

Links

Code	Description	Links
200	Returns a category	No links
	<p>Media type</p> <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">application/json</div> ▼ <p>Controls Accept header.</p>	
	Example Value Schema <pre>{ "_id": "5e539c042f9afe16e09ca244", "name": "Game Consoles", "parent": { "_id": "5e539c042f9afe16e09ca244", "name": "Gaming" } }</pre>	
400	Invalid ID. The ID must be a valid MongoDB ID.	No links
404	Category with given ID not found	No links

PUT /categories/{id} Update existing category 🔒

Used to update an existing category.

Parameters [Try it out](#)

Name	Description
id * required string (path)	Valid mongoose Object Id 5e539c042f9afe16e09ca244

Request body required application/json ▼

A JSON object containing category information. **Parent field is optional**

[Example Value](#) | [Schema](#)

```
{  
  "name": "Game Consoles",  
  "parent": {  
    "_id": "5e539c042f9afe16e09ca244"  
  }  
}
```

Responses

Code	Description	Links
200	Category successfully updated	<i>No links</i>
	Media type	
	<div style="border: 1px solid #ccc; padding: 2px;">application/json</div> ▾	
	Controls Accept header.	
	Example Value Schema	
	<pre>{ "_id": "5e539c042f9afe16e09ca244", "name": "Game Consoles", "parent": { "_id": "5e539c042f9afe16e09ca244", "name": "Gaming" } }</pre>	
400	Invalid category in request body.	<i>No links</i>
404	The category does not exist.	<i>No links</i>

GET

/categories Returns all categories



Use to get all categories

Parameters

[Try it out](#)

No parameters

Responses

Code	Description	Links
200	A JSON array with all categories	No links

Media type

application/json



Controls Accept header.

[Example Value](#) | [Schema](#)

```
[  
  {  
    "_id": "5e539c042f9afe16e09ca244",  
    "name": "Game Consoles",  
    "parent": {  
      "_id": "5e539c042f9afe16e09ca244",  
      "name": "Gaming"  
    }  
  }  
]
```

POST

/categories Insert a new category



Use to insert a new category. The category may contain a parent reference. Note that only one level of subcategorization allowed.

Parameters

[Try it out](#)

No parameters

Request body required

application/json



A JSON object containing category information. **Parent field is optional**

[Example Value](#) | [Schema](#)

```
{  
  "name": "Game Consoles",  
  "parent": {
```

```
        }  
    }  
  
    "_id": "5e539c042f9afe16e09ca244"
```

Responses

Code	Description	Links
------	-------------	-------

200	A JSON object with the new category	<i>No links</i>
-----	-------------------------------------	-----------------

Media type

application/json



Controls Accept header.

[Example Value](#) | [Schema](#)

```
{  
    "_id": "5e539c042f9afe16e09ca244",  
    "name": "Game Consoles",  
    "parent": {  
        "_id": "5e539c042f9afe16e09ca244",  
        "name": "Gaming"  
    }  
}
```

400	Invalid category format or parent is not a main category.	<i>No links</i>
-----	---	-----------------

404	Parent with provided parent id does not exist	<i>No links</i>
-----	---	-----------------

Products



GET

/products/{id} Returns a product with the given id



Returns a specific product

Parameters

Try it out

Name	Description
------	-------------

id * required

string
(path)

Valid mongoose Object Id

5e539c042f9afe16e09ca244

Responses

Code	Description	Links
------	-------------	-------

200

A JSON object representing the product

No links

Media type

application/json



Controls Accept header.

[Example Value](#) | [Schema](#)

```
{
  "_id": "5e539c042f9afe16e09ca244",
  "name": "Playstation 4",
  "category": {
    "_id": "5e539c042f9afe16e09ca244",
    "name": "Gaming Consoles"
  },
  "entities": [
    {
      "identifier": "PS4_1",
      "availableForRental": false,
      "remarks": "Scratches"
    }
  ],
  "numberOfLoans": 10,
  "description": "A Playstation 4 console",
  "details": [
    {
      "displayName": "Maximum players",
      "value": "4"
    }
  ]
}
```

400

Invalid ID. The ID must be a valid MongoDB ID.

No links

Code	Description	Links
404	The product with the specified ID was not found	No links

DELETE /products/{id} Deletes a product with given id 

Use to delete a product by id

Parameters Try it out

Name	Description
id * required string (path)	Valid mongoose Object Id 5e539c042f9afe16e09ca244

Responses

Code	Description	Links
------	-------------	-------

Code	Description	Links
200	Returns a JSON object representing the deleted product	No links
	<p>Media type</p> <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">application/json</div> ▼ <p>Controls Accept header.</p>	
	Example Value Schema	
	<pre>{ "_id": "5e539c042f9afe16e09ca244", "name": "Playstation 4", "category": { "_id": "5e539c042f9afe16e09ca244", "name": "Gaming Consoles" }, "entities": [{ "identifier": "PS4_1", "availableForRental": false, "remarks": "Scratches" }], "numberOfLoans": 10, "description": "A Playstation 4 console", "details": [{ "displayName": "Maximum players", "value": "4" }] }</pre>	
400	Invalid ID. The ID must be a valid MongoDB ID.	No links
404	Product was not found	No links

PUT
/products/{id} Updates an existing product with the given id
🔒

Update an existing product

Parameters
Try it out

Name	Description

Name	Description
------	-------------

id * required

string
(path)

Valid mongoose Object Id

5e539c042f9afe16e09ca244

Request body required

application/json



A JSON object containing product information

[Example Value](#) | [Schema](#)

```
{  
  "name": "Playstation 4",  
  "category": {  
    "_id": "5e539c042f9afe16e09ca244"  
  },  
  "entities": [  
    {  
      "identifier": "PS4_1",  
      "availableForRental": false,  
      "remarks": "Scratches"  
    }  
  ],  
  "numberOfLoans": 10,  
  "description": "A Playstation 4 console",  
  "details": [  
    {  
      "displayName": "Maximum players",  
      "value": "4"  
    }  
  ]  
}
```

Responses

Code	Description	Links
------	-------------	-------

Code	Description	Links
200	Returns the updated product	No links
	<p>Media type</p> <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">application/json</div> <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>{ "_id": "5e539c042f9afe16e09ca244", "name": "Playstation 4", "category": { "_id": "5e539c042f9afe16e09ca244", "name": "Gaming Consoles" }, "entities": [{ "identifier": "PS4_1", "availableForRental": false, "remarks": "Scratches" }], "numberOfLoans": 10, "description": "A Playstation 4 console", "details": [{ "displayName": "Maximum players", "value": "4" }] }</pre>	
400	Invalid product in request body	No links
404	Product with given ID was not found	No links

GET /products Returns a list of all products 

Use to request all products

Parameters [Try it out](#)

No parameters

Responses

Code	Description	Links
200	A JSON array of products	No links

Media type

application/json



Controls Accept header.

[Example Value](#) | [Schema](#)

```
[  
  {  
    "_id": "5e539c042f9afe16e09ca244",  
    "name": "Playstation 4",  
    "category": {  
      "_id": "5e539c042f9afe16e09ca244",  
      "name": "Gaming Consoles"  
    },  
    "entities": [  
      {  
        "identifier": "PS4_1",  
        "availableForRental": false,  
        "remarks": "Scratches"  
      }  
    ],  
    "numberOfLoans": 10,  
    "description": "A Playstation 4 console",  
    "details": [  
      {  
        "displayName": "Maximum players",  
        "value": "4"  
      }  
    ]  
  }  
]
```

POST

/products Add a new product



Use to add a new product

Parameters

[Try it out](#)

No parameters

Request body required

application/json



A JSON object containing product information

[Example Value](#) | [Schema](#)

```
{  
  "name": "Playstation 4",  
  "category": {  
    "_id": "5e539c042f9afe16e09ca244"  
  },  
  "entities": [  
    {  
      "identifier": "PS4_1",  
      "availableForRental": false,  
      "remarks": "Scratches"  
    }  
  ],  
  "numberOfLoans": 10,  
  "description": "A Playstation 4 console",  
  "details": [  
    {  
      "displayName": "Maximum players",  
      "value": "4"  
    }  
  ]  
}
```

Responses

Code	Description	Links
------	-------------	-------

Code	Description	Links
200	Returns the new product with corresponding ID Media type <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">application/json</div> ▼ Controls Accept header. Example Value Schema <pre>{ "_id": "5e539c042f9afe16e09ca244", "name": "Playstation 4", "category": { "_id": "5e539c042f9afe16e09ca244", "name": "Gaming Consoles" }, "entities": [{ "identifier": "PS4_1", "availableForRental": false, "remarks": "Scratches" }], "numberOfLoans": 10, "description": "A Playstation 4 console", "details": [{ "displayName": "Maximum players", "value": "4" }] }</pre>	No links

Auth



POST	/auth/resetpassword	🔒
Use to request a password reset link to email		
Parameters		Try it out
No parameters		

Request body

application/json ▾

[Example Value](#) | Schema

```
{  
  "email": "John@email.com"  
}
```

Responses

Code	Description	Links
200	Email has been sent if the user was found in database.	No links
400	Email missing in request body	No links

POST

/auth Let user log in



Use to log in a new user

Parameters

[Try it out](#)

No parameters

Request body required

application/json ▾

A JSON object containing user email and password

[Example Value](#) | Schema

```
{  
  "email": "Josh_Johnson@gmail.com",  
  "password": "PasswordForJosh_Johnson"  
}
```

Responses

Code	Description	Links
200	Returns the JSON web token of the user	No links

Media type

text/plain

Controls Accept header.

[Example Value](#) | [Schema](#)

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI1ZTZjYjE3NWUwYjNiZDM2NGNjMmZmYjgiLCJuYW1lIjoiZmlyc3ROYW11MSBsYXN0TmFtZTdEiLCJpc0FkbWluIjpmYWxzZSwiaWF0IjoxNTg0MzYxOTg4fQ.FsUbFWHD9AwI7UfxGwDUKtVtvwVfNg7Mo7vac(ARG64)
```

POST

/auth/token/{JWT} Let user verify email



Activate users account

Parameters

[Try it out](#)

Name	Description
------	-------------

JWT * required

string
(path)

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQi
```

Responses

Code	Description	Links
------	-------------	-------

Code**Description****Links**

200

Account activated

*No links***400**
Users

Invalid token

*No links***PATCH**`/users/me` updates logged in user

Use to updated the logged in user

Parameters**Try it out**

No parameters

Request body required`application/json`

A JSON object containing user info to be updated

[Example Value](#) | [Schema](#)

```
{  
  "firstName": "Josh",  
  "lastName": "Johnson",  
  "phone": "48927401"  
}
```

Responses**Code****Description****Links**

Code	Description	Links
200	User updated	No links
	<p>Media type</p> <p>application/json ▾</p> <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>{ "_id": "5e6106c3b3817f3a88be0860", "firstName": "Josh", "lastName": "Johnson", "phone": "48927401" }</pre>	

GET /users/{id} Gets a user with the given ID 

Returns the specific user

Parameters [Try it out](#)

Name	Description
id * required string (path)	Valid mongoose Object Id <input type="text" value="5e539c042f9afe16e09ca244"/>

Code	Description	Links

Code	Description	Links
200	A JSON object representing the user	No links
	<p>Media type</p> <div style="border: 2px solid green; padding: 2px; display: inline-block;">application/json</div> ▼ <p>Controls Accept header.</p>	
	Example Value Schema <pre>{ "_id": "5e6106c3b3817f3a88be0860", "firstName": "Josh", "lastName": "Johnson", "email": "Josh_Johnson@gmail.com", "password": "\$2b\$10\$cQyTw0/9XqjSwCOK3lFgevZ1AYVSCWr6VJYzgha2zBCWk3X9uUkC", "phone": "48927401" }</pre>	
400	Invalid ID. The ID must be a valid MongoDB ID.	No links
404	The user with the specified ID was not found	No links

DELETE /users/{id} Deletes a user with given id 

Use to delete a user by id

Parameters [Try it out](#)

Name	Description
id * required string (path)	Valid mongoose Object Id <input type="text" value="5e539c042f9afe16e09ca244"/>

Responses

Code	Description	Links
200	Returns a JSON object representing the deleted User	No links
	<p>Media type</p> <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">application/json</div> ▼ <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>{ "_id": "5e6106c3b3817f3a88be0860", "firstName": "Josh", "lastName": "Johnson", "email": "Josh_Johnson@gmail.com", "password": "\$2b\$10\$wCQyTw0/9XqjSwCOK3lFgevZ1AYVSCWr6VJYzgha2zBCWk3X9uUkC", "phone": "48927401" }</pre>	
400	Invalid ID. The ID must be a valid MongoDB ID.	No links
404	User was not found	No links

PUT

/users/{id} Updates an existing User with the given id



Update an existing product

Parameters**Try it out****Name****Description****id** * requiredstring
(path)

Valid mongoose Object Id

5e539c042f9afe16e09ca244

Request body required

application/json



A JSON object containing user

[Example Value](#) | [Schema](#)

```
{  
  "firstName": "Josh",  
  "lastName": "Johnson",  
  "email": "Josh_Johnson@gmail.com",  
  "password": "PasswordForJosh_Johnson",  
  "phone": "48927401"  
}
```

Responses

Code	Description	Links
200	Returns the updated User	<i>No links</i>
	Media type	
	<div style="border: 1px solid #ccc; padding: 2px 10px; display: inline-block;">application/json</div> ▾	
	Controls Accept header.	
	Example Value Schema	
	<pre>{ "_id": "5e6106c3b3817f3a88be0860", "firstName": "Josh", "lastName": "Johnson", "email": "Josh_Johnson@gmail.com", "password": "\$2b\$10\$/wCQyTw0/9XqjSwCOK3lFgevZlAYVSCWr6VJYzgha2zBCWk3X9uUkC", "phone": "48927401" }</pre>	
400	Invalid User in request body	<i>No links</i>
404	User with given ID was not found	<i>No links</i>

GET

/users Return list of all Users



User to add a new User

Parameters

[Cancel](#)

No parameters

[Execute](#)

[Clear](#)

Responses

Curl

```
curl -X GET  
"http://localhost:3900/  
api/users" -H "accept:  
application/json"
```

Request URL

<http://localhost:3900/api/users>

Server response

Code Details

401 Error: Unauthorized

Undocumented

Response body

Access denied. No token provided.

[Download](#)

Response headers

```
access-control-allow-origin: http://localhost:3000  
connection: keep-alive  
content-length: 33  
content-type: text/html; charset=utf-8  
date: Fri, 08 May 2020 10:02:38 GMT  
etag: W/"21-Bk1Us52z+uiNr4xD+HZFkaoeUOI"  
vary: Origin  
x-powered-by: Express
```

Responses

Code Description

Links

Code	Description	Links
200	<p>Return all the registered users.</p> <p>Media type</p> <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">application/json</div> <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>{ "_id": "5e6106c3b3817f3a88be0860", "firstName": "Josh", "lastName": "Johnson", "email": "Josh_Johnson@gmail.com", "password": "\$2b\$10\$wCQyTw0/9XqjSwCOK3lFgevZlAYVSCWr6VJYzgha2zBCWk3X9uUkC", "phone": "48927401" }</pre>	No links

POST /users Add a new user

Use to add a new user

Parameters [Try it out](#)

No parameters

Request body required application/json

A JSON object containing user

[Example Value](#) | [Schema](#)

```
{
  "firstName": "Josh",
  "lastName": "Johnson",
  "email": "Josh_Johnson@gmail.com",
  "password": "PasswordForJosh_Johnson",
  "phone": "48927401"
}
```

Responses

Code	Description	Links
200	Return the new user with its corresponding ID	No links
	<p>Media type</p> <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;"> application/json </div> <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>{ "_id": "5e6106c3b3817f3a88be0860", "firstName": "Josh", "lastName": "Johnson", "email": "Josh_Johnson@gmail.com", "password": "\$2b\$10\$wCQyTw0/9XqjSwCOK3lFgevZlAYVSCWr6VJYzgha2zBCWk3X9uUkC", "phone": "48927401" }</pre>	

400	Invalid email, invalid password, user already registered, first name missing, last name missing, phone missing.	No links
-----	---	----------

GET

/users/{userId}/rentals Returns an array of rentals that is rented by the given userId



Returns an array of rentals

Parameters**Try it out****Name****id** * requiredstring
(path)

Valid mongoose Object Id

5e539c042f9afe16e09ca244

requested * requiredboolean
(query)

If true, returns only requested/unprocessed rentals

--

Responses

Code	Description	Links
200	A JSON object representing the product	No links

Media type

application/json

Controls Accept header.

[Example Value](#) | [Schema](#)

```
[  
  {  
    "_id": "5e6106c3b3817f3a88be0860",  
    "user": {  
      "_id": "5e6106c3b3817f3a88be0860",  
      "name": "John Johnson"  
    },  
    "product": {  
      "_id": "5e6106c3b3817f3a88be0860",  
      "name": "Playstation 4",  
      "entity": {  
        "_id": "5e6106c3b3817f3a88be0860",  
        "identifier": "PS4_1",  
        "remarks": "Scratches"  
      }  
    },  
    "dateOut": "2020-01-01T17:32:28.000Z",  
    "dateReturned": "2020-01-10T17:32:28.000Z",  
    "confirmedReturned": true,  
    "pickupInstructions": "Pick up at the mall. Call 99887766 when  
there",  
    "returnInstructions": "Call me at 99887766 to schedule return",  
    "remarks": "Power cable was damaged under use"  
  },  
  {  
    "_id": "5e6106c3b3817f3a88be0860",  
    "user": {  
      "id": "5e6106c3b3817f3a88be0860",  
      "name": "John Johnson"  
    },  
    "product": {  
      "_id": "5e6106c3b3817f3a88be0860",  
      "name": "Playstation 4",  
      "entity": {  
        "_id": "5e6106c3b3817f3a88be0860",  
        "identifier": "PS4_1",  
        "remarks": "Scratches"  
      }  
    },  
    "dateOut": "2020-01-01T17:32:28.000Z",  
    "dateReturned": "2020-01-10T17:32:28.000Z",  
    "confirmedReturned": true,  
    "pickupInstructions": "Pick up at the mall. Call 99887766 when  
there",  
    "returnInstructions": "Call me at 99887766 to schedule return",  
    "remarks": "Power cable was damaged under use"  
  }]
```

POST

/users/newPassword/{token} Changes user password



Use to change user password

Parameters

[Try it out](#)

Name

Description

Name	Description	
token * required string (path)	JWT token for password reset	
	token - JWT token for password reset	
Request body required	application/json	
Example Value Schema		
{ "password": "string" }		
Responses		
Code	Description	Links
200	Password was reset successfully	No links
400	Missing password in request body!	No links
401	Token expired or invalid	No links

Rentals



DELETE	/rentals/{id} Delete rental	
Delete rental with given ID		

Parameters

[Try it out](#)

Name	Description
------	-------------

id * requiredstring
(path)

Valid mongoose Object Id

5e539c042f9afe16e09ca244

Request body required

application/json



Whether to notify user of rental rejection and the reason for rejection

[Example Value](#) | [Schema](#)

```
{  
  "notifyUser": true,  
  "deleteMessage": "Can't find it."  
}
```

Responses

Code**Description****Links**

Code	Description	Links
200	Renturn deleted rental. Media type application/json ▾ Controls Accept header. Example Value Schema	No links

PATCH /rentals/{id} Confirms a rental (Admin functionality) 

Used to confirm a rental and send an email to the user connected to the rental. This sets the properties pickupInstructions, returnInstructions and dateOut.

Parameters	Try it out
Name Description <hr/> id * required string <i>(path)</i>	
Request body required <hr/> application/json ▾	

[Example Value](#) | [Schema](#)

```
{  
  "pickupInstructions": "Pick up at Ole Brumms vei 1 between 16:00 - 21:00. Call me on  
  99887766.",  
  "returnInstructions": "Return at same spot."  
}
```

Responses

Code	Description	Links
200	OK	<i>No links</i>
	<p>Media type</p> <div><p>application/json ▾</p><p>Controls Accept header.</p></div> <p>Example Value Schema</p> <pre>[{ "_id": "5e6106c3b3817f3a88be0860", "user": { "_id": "5e6106c3b3817f3a88be0860", "name": "John Johnson" }, "product": { "_id": "5e6106c3b3817f3a88be0860", "name": "Playstation 4", "entity": { "_id": "5e6106c3b3817f3a88be0860", "identifier": "PS4_1" } }, "confirmedReturned": false, "pickupInstructions": "Pick up at the mall. Call 99887766 when there", "returnInstructions": "Call me at 99887766 to schedule return", "dateOut": "2020-01-01T17:32:28.000Z" }]</pre>	
400	Invalid request body. Should contain pickupInstructions and returnInstructions	<i>No links</i>
404	Rental with given ID not found	<i>No links</i>

Code	Description	Links
500	ID of user in rental body does not exist in user collection in database.	No links

GET /rentals Get rentals 

Get all rentals or only requested/unprocessed rentals

Parameters Try it out

Name	Description
requested * required boolean (query)	If true, returns only requested/unprocessed rentals <input type="button" value="--"/>

Responses

Code	Description	Links
------	-------------	-------

Code	Description	Links
200	Returns all rentals. Media type <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">application/json</div> Controls Accept header.	No links

[Example Value](#) | [Schema](#)

```
[  
  {  
    "_id": "5e6106c3b3817f3a88be0860",  
    "user": {  
      "_id": "5e6106c3b3817f3a88be0860",  
      "name": "John Johnson"  
    },  
    "product": {  
      "_id": "5e6106c3b3817f3a88be0860",  
      "name": "Playstation 4",  
      "entity": {  
        "_id": "5e6106c3b3817f3a88be0860",  
        "identifier": "PS4_1"  
      }  
    },  
    "confirmedReturned": false,  
    "pickupInstructions": "Pick up at the mall. Call 99887766 when there",  
    "returnInstructions": "Call me at 99887766 to schedule return",  
    "dateOut": "2020-01-01T17:32:28.000Z"  
  },  
  {  
    "_id": "5e6106c3b3817f3a88be0860",  
    "user": {  
      "_id": "5e6106c3b3817f3a88be0860",  
      "name": "John Johnson"  
    },  
    "product": {
```

Code	Description	Links
202	Returns requested rentals only.	No links

Media type

application/json ▾

[Example Value](#) | [Schema](#)

```
[{"_id": "5e6106c3b3817f3a88be0860", "user": {"_id": "5e6106c3b3817f3a88be0860", "name": "John Johnson"}, "product": {"_id": "5e6106c3b3817f3a88be0860", "name": "Playstation 4", "entity": {"_id": "5e6106c3b3817f3a88be0860", "identifier": "PS4_1"}}, "confirmedReturned": false}]
```

POST /rentals Create a new rental 

Let user rent an item

Parameters 

No parameters

Request body required 

A JSON object containing the productId

[Example Value](#) | [Schema](#)

```
{ "productId": "5e539c042f9afe16e09ca244" }
```

Responses

Code	Description	Links
200	Returns the new rental	<i>No links</i>

Media type

application/json

Controls Accept header.

[Example Value](#) | [Schema](#)

```
{  
  "_id": "5e6106c3b3817f3a88be0860",  
  "user": {  
    "_id": "5e6106c3b3817f3a88be0860",  
    "name": "John Johnson"  
  },  
  "product": {  
    "_id": "5e6106c3b3817f3a88be0860",  
    "name": "Playstation 4",  
    "entity": {  
      "_id": "5e6106c3b3817f3a88be0860",  
      "identifier": "PS4_1",  
      "remarks": "Scratches"  
    }  
  },  
  "confirmedReturned": false  
}
```

Suggestions

GET

/suggestions Get Suggestions



Get all suggestions

Parameters

[Try it out](#)

No parameters

Responses

Code	Description	Links
200	Returns all suggestions	No links

Media type

application/json
 ▼

Controls Accept header.

[Example Value](#) | [Schema](#)

```
[{"_id": "5e6106c3b3817f3a88be0860", "suggestion": "Get Playstation 6", "user": {"_id": null, "name": "5e6106c3b3817f3a88be0860"}}]
```

DELETE /suggestions Delete suggestion 🔒

Delete a suggestion with the given ID

Parameters [Try it out](#)

Name	Description
id * required string (path)	Valid mongoose Object Id 5e539c042f9afe16e09ca244

Responses

Code	Description	Links
------	-------------	-------

Code	Description	Links
200	Return deleted suggestion	No links
	<p>Media type</p> <div><p>application/json</p><p>▼</p></div> <p>Controls Accept header.</p>	
	<p>Example Value Schema</p> <pre>{ "_id": "5e6106c3b3817f3a88be0860", "suggestion": "Get Playstation 6", "user": { "_id": null, "name": "5e6106c3b3817f3a88be0860" } }</pre>	
400	Invalid mongoose ID	No links
404	The suggestion with the given ID doesn't exist	No links

POST /suggestions Create a new suggestion 

Create a new suggestion using the given request body

Parameters 

No parameters

Request body required 

A JSON object containing the suggestion

Example Value | Schema

```
{  
  "suggestion": "Buy playstation 6"  
}
```

Responses

Code	Description	Links
200	Returns the created suggestion	No links

Media type

application/json



Controls Accept header.

[Example Value](#) | [Schema](#)

```
{  
  "_id": "5e6106c3b3817f3a88be0860",  
  "suggestion": "Get Playstation 6",  
  "user": {  
    "_id": null,  
    "name": "5e6106c3b3817f3a88be0860"  
  }  
}
```

400

No links

The suggestion in the request body is invalid

Service-Messages



GET

/service-messages Get service messages



Get all service messages

Parameters

[Try it out](#)

No parameters

Responses

Code	Description	Links
------	-------------	-------

Code	Description	Links
200	OK	No links

Media type

application/json

Controls Accept header.

[Example Value](#) | [Schema](#)

```
[
  {
    "_id": "5e6106c3b3817f3a88be0860",
    "serviceMessage": "The site will be down Sunday 23/06 due to maintenance"
  }
]
```

POST
/service-messages
Create a new service message
🔒

Create a new service message with the given request body

Parameters
[Try it out](#)

No parameters

Request body required
application/json

A JSON object containing the serviceMessage

[Example Value](#) | [Schema](#)

```
{
  "serviceMessage": "The site will be down friday 23/06 due to maintenance"
}
```

Responses

Code
Description
Links

Code	Description	Links
200	OK	No links
	<p>Media type</p> <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">application/json</div> <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>[{ "_id": "5e6106c3b3817f3a88be0860", "serviceMessage": "The site will be down Sunday 23/06 due to maintenance" }]</pre>	

DELETE	/service-messages/{id} Delete service message									
Delete a service message with the given ID										
Parameters		Try it out								
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 5px;">Name</th><th style="text-align: left; padding: 5px;">Description</th></tr> </thead> <tbody> <tr> <td style="padding: 5px;">id * required</td><td style="padding: 5px;"></td></tr> <tr> <td style="padding: 5px;">string (path)</td><td style="padding: 5px;">Valid mongoose Object Id</td></tr> <tr> <td colspan="2" style="text-align: left; padding: 5px;"><input type="text" value="5e539c042f9afe16e09ca244"/></td></tr> </tbody> </table>			Name	Description	id * required		string (path)	Valid mongoose Object Id	<input type="text" value="5e539c042f9afe16e09ca244"/>	
Name	Description									
id * required										
string (path)	Valid mongoose Object Id									
<input type="text" value="5e539c042f9afe16e09ca244"/>										
Responses										
Code	Description	Links								

Code	Description	Links
200	Return delete service message	No links
	<p>Media type</p> <div style="border: 2px solid #00AEEF; padding: 2px; display: inline-block;">application/json</div> ▼ <p>Controls Accept header.</p>	
	Example Value Schema <pre>{ "_id": "5e6106c3b3817f3a88be0860", "serviceMessage": "The site will be down Sunday 23/06 due to maintenance" }</pre>	
400	Invalid ID. The ID must be a valid MongoDB ID.	No links
404	Service message was not found	No links

Returns



GET
/rentals/returns Get returns
Lock

Get unprocessed returns or all returns

Parameters
Try it out

Name	Description
processed boolean (query)	---

Responses

Code	Description	Links
200	Returns an array of all returns Media type <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">application/json</div> ▼ Controls Accept header. Example Value Schema	No links

```
[  
  {  
    "_id": "5e6106c3b3817f3a88be0860",  
    "user": {  
      "_id": "5e6106c3b3817f3a88be0860",  
      "name": "John Johnson"  
    },  
    "product": {  
      "_id": "5e6106c3b3817f3a88be0860",  
      "name": "Playstation 4",  
      "entity": {  
        "_id": "5e6106c3b3817f3a88be0860",  
        "identifier": "PS4_1",  
        "remarks": "Scratches"  
      }  
    },  
    "dateOut": "2020-01-01T17:32:28.000Z",  
    "dateReturned": "2020-01-10T17:32:28.000Z",  
    "confirmedReturned": true,  
    "pickupInstructions": "Pick up at the mall. Call 99887766 when  
there",  
    "returnInstructions": "Call me at 99887766 to schedule return",  
    "remarks": "Power cable was damaged under use"  
  },  
  {  
    "_id": "5e6106c3b3817f3a88be0860",  
    "user": {  
      "id": "5e6106c3b3817f3a88be0860",  
      "name": "John Johnson"  
    },  
    "product": {  
      "_id": "5e6106c3b3817f3a88be0860",  
      "name": "Playstation 4",  
      "entity": {  
        "_id": "5e6106c3b3817f3a88be0860",  
        "identifier": "PS4_2",  
        "remarks": "Scratches"  
      }  
    },  
    "dateOut": "2020-01-01T17:32:28.000Z",  
    "dateReturned": "2020-01-10T17:32:28.000Z",  
    "confirmedReturned": true,  
    "pickupInstructions": "Pick up at the mall. Call 99887766 when  
there",  
    "returnInstructions": "Call me at 99887766 to schedule return",  
    "remarks": "Power cable was damaged under use"  
  }]
```

Code	Description	Links
202	Returns an array of only unprocessed returns	No links

Media type

application/json

Example Value | Schema

```
[{"_id": "5e6106c3b3817f3a88be0860", "user": {"_id": "5e6106c3b3817f3a88be0860", "name": "John Johnson"}, "product": {"_id": "5e6106c3b3817f3a88be0860", "name": "Playstation 4", "entity": {"_id": "5e6106c3b3817f3a88be0860", "identifier": "PS4_1", "remarks": "Scratches"}}, "dateOut": "2020-01-01T17:32:28.000Z", "dateReturned": "2020-01-10T17:32:28.000Z", "confirmedReturned": false, "pickupInstructions": "Pick up at the mall. Call 99887766 when there", "returnInstructions": "Call me at 99887766 to schedule return", "remarks": "Power cable was damaged under use"}]
```

POST /rentals/returns/{id} Requests a return. 

Use to request confirmation of a return when a user has returned an item. This sets "dateReturned" property to current date.

Parameters [Try it out](#)

Name	Description
id * required string (path)	Valid mongoose Object Id 5e539c042f9afe16e09ca244

Request body

application/json ▾

A JSON object containing remarks property.

[Example Value](#) | [Schema](#)

```
{  
  "remarks": "Got some new scratches during transport."  
}
```

Responses

Code	Description	Links
200	OK	<i>No links</i>
	<p>Media type</p> <p>application/json ▾</p> <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>{ "_id": "5e6106c3b3817f3a88be0860", "user": { "_id": "5e6106c3b3817f3a88be0860", "name": "John Johnson" }, "product": { "_id": "5e6106c3b3817f3a88be0860", "name": "Playstation 4", "entity": { "_id": "5e6106c3b3817f3a88be0860", "identifier": "PS4_1" } }, "confirmedReturned": false }</pre>	
400	User requesting return does not has the same ID as in the rental.	<i>No links</i>
404	Rental with given ID not found.	<i>No links</i>

PATCH

/rentals/returns/{id} Confirms a return



Use to confirm a return. This sets the confirmReturned field and marks the end of the rental lifecycle.

Parameters

[Try it out](#)

Name	Description
------	-------------

id * required	
----------------------	--

string (path)	Valid mongoose Object Id
------------------	--------------------------

```
5e539c042f9afe16e09ca244
```

Request body	required
--------------	----------

application/json	
------------------	--

A JSON object containing a setAvailable property to determine whether the entity is to be set as available for rental or not.

[Example Value](#) | [Schema](#)

```
{  
  "setAvailable": true  
}
```

Responses

Code	Description	Links
------	-------------	-------

Code	Description	Links
200	OK	No links
	<p>Media type</p> <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">application/json</div> <p>Controls Accept header.</p>	
	Example Value Schema <pre>{ "_id": "5e6106c3b3817f3a88be0860", "user": { "_id": "5e6106c3b3817f3a88be0860", "name": "John Johnson" }, "product": { "_id": "5e6106c3b3817f3a88be0860", "name": "Playstation 4", "entity": { "_id": "5e6106c3b3817f3a88be0860", "identifier": "PS4_1" } }, "confirmedReturned": false }</pre>	
400	Request body missing setAvailable field.	No links
404	Rental with given ID not found.	No links

Schemas

Category >

Product >

Auth >

User >

UpdatedUser >

Rental >

RequestedRental >

ConfirmedRental >

Suggestion >

UnprocessedReturn >

ServiceMessage >

Tillegg H

Brukerakseptansetest

Spørreundersøkelse for loan app

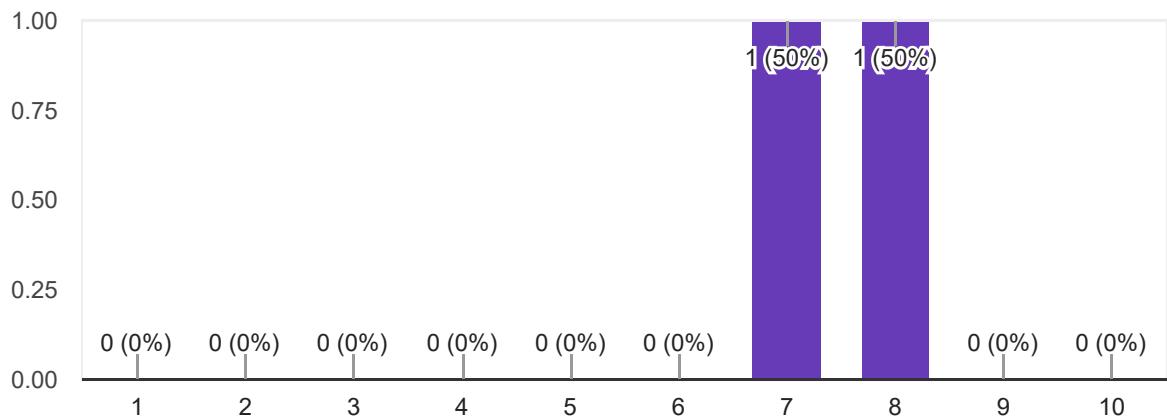
2 responses

[Publish analytics](#)

Testing av brukerprofil

Hvor enkelt var det å lage en ny bruker?

2 responses



Feedback

2 responses

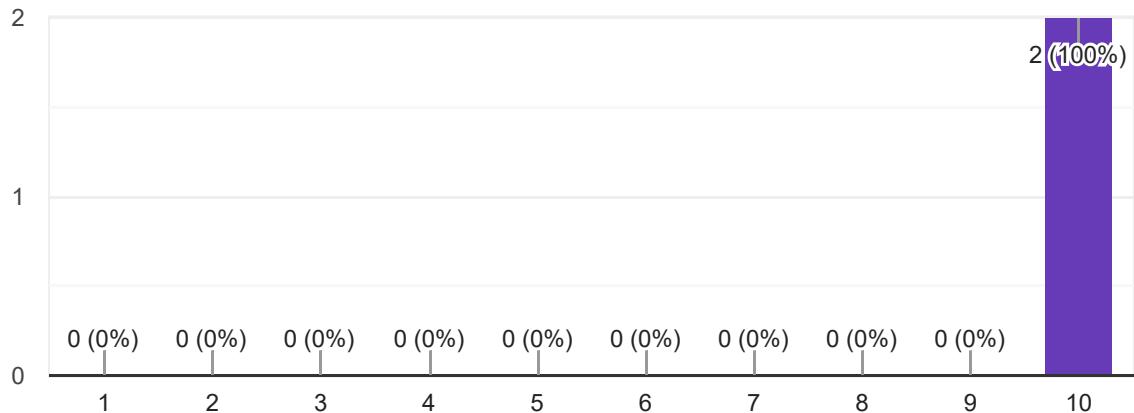
Enkelt og intuitivt å lage en ny bruker. Bekreftelsesmailen fungerte ikke på mobil, men det kan være pga Accenture-restriksjoner. Det fungerte fint på PC.

I et ordentlig https miljø så tror jeg dette vil fungere svært bra. Litt utfordringer nå med en ikke https sikret mailserver. Forbedringer som kan gjøres er at feilmeldingene med ikke valid input kan gjøres litt bedre. De forsvinner hvis man bruker backspace knappen eller hvis man fyller felter med nettleserens innebygde funksjonalitet. Også greit med en validering av når e-post er sendt, eller en eventuell feilmelding hvis utsending av e-post feiler.



Hvordan var prosessen med å logge inn?

2 responses



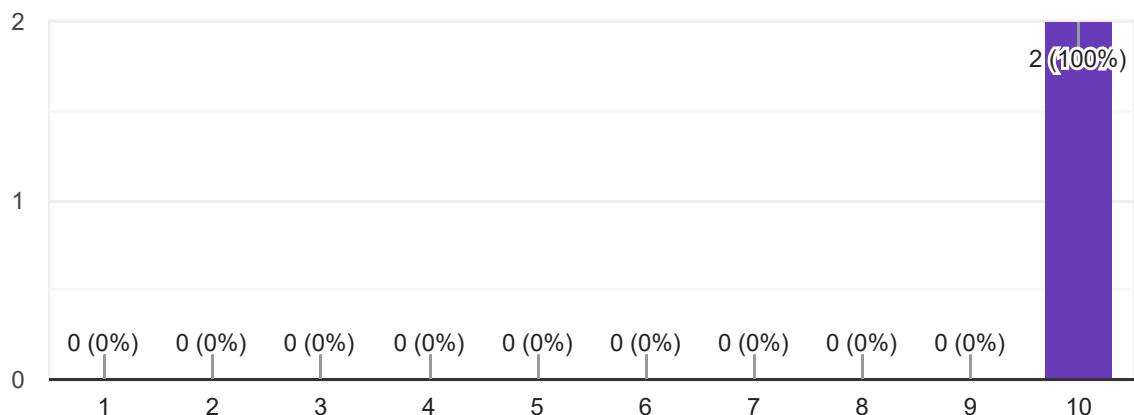
Feedback

1 response

Veldig tydelig logg inn side, ingen forvirring her.

Hvor intuitivt var det å redigere bruker informasjon?

2 responses



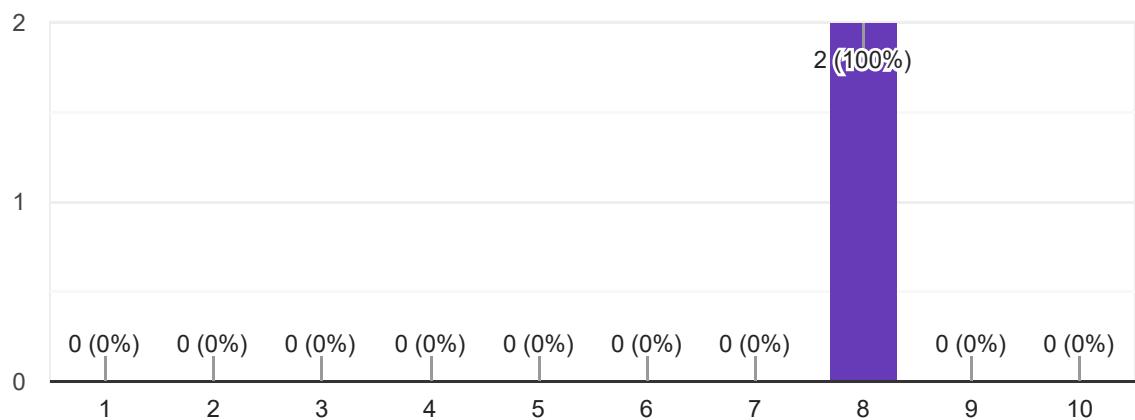
Feedback

1 response

Veldig intuitivt!

Hvordan var brukeropplevelsen med å bytte passord?

2 responses



Feedback

2 responses

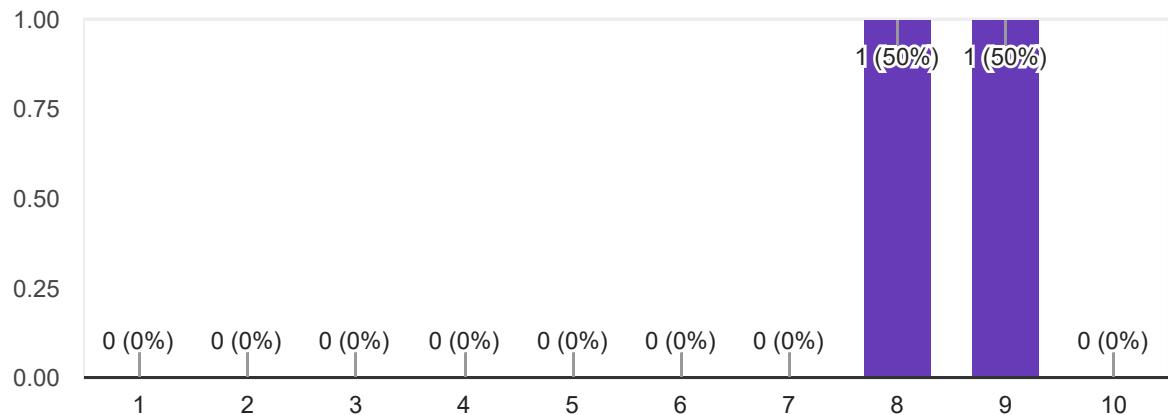
Linken viste ingen informasjon når jeg gikk inn på den med mobilen. Det kan være samme problem som over. På PC fungerte det fint og intuitivt!

Fikk det ikke til å fungere nå, men fått beskrevet funksjonaliteten og virker akkurat som forventet. Igjen tror jeg det er lurt å innføre en feilmelding når e-post utsendelse feiler.



Hva er din totale vurdering av brukerprofil og dets funksjonalitet?

2 responses



Feedback

2 responses

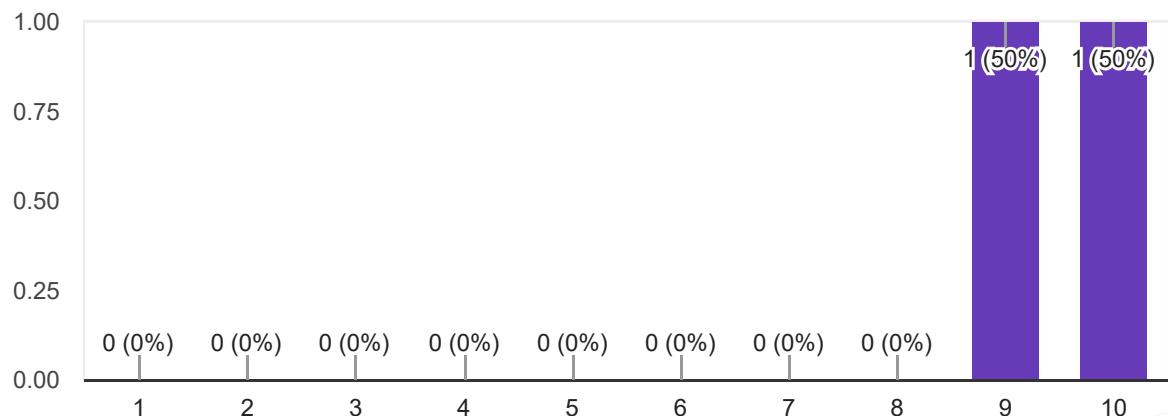
Jeg synes at det er intuitivt og godt satt opp.

Litt smårusk med valideringsmeldinger, men liker hvordan de ser ut. Må bare støtte autofyll fra nettleser. Ser også at nettsiden ikke fungerer i firefox som kunne vært en tanke til videre utvikling.

Testing av produkter og leie funksjonalitet

Hvor oversiktlig var produktsiden?

2 responses



Feedback

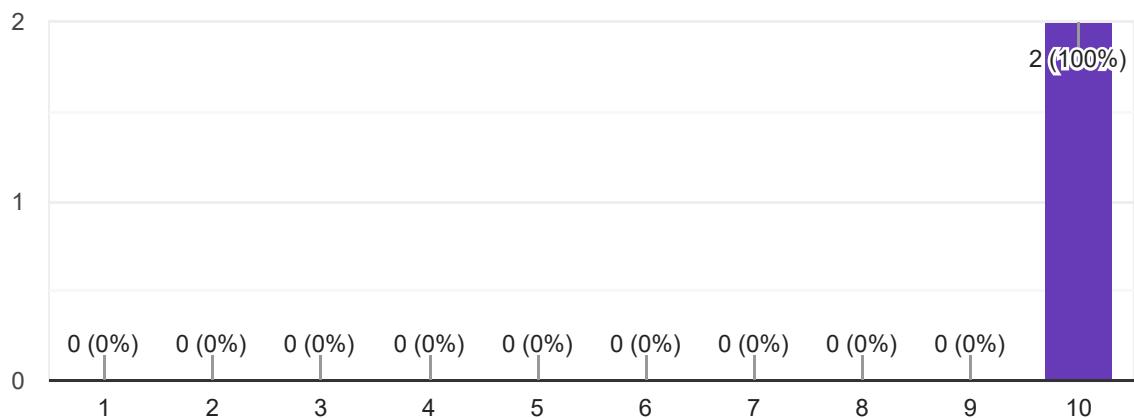
2 responses

Jeg synes produktsiden er enkel å få oversikt over.

Veldig oversiktlig! Noen fargevalg hadde nok gjort siden enda lettere å lese, ettersom nå er det mye hvitt på hvitt. En annen mindre justering jeg ville gjort er at alle product boksene skulle fått samme størrelse som den største boksen, men det er pirk. Liker veldig godt søke funksjonene og kategori skillet.

Hvor enkelt var det å finne et produkt?

2 responses



Feedback

2 responses

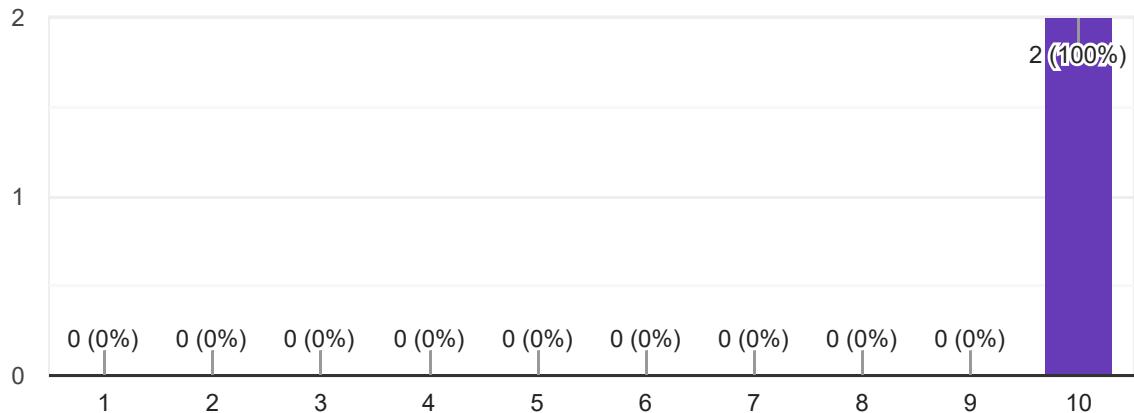
Søkefunksjonaliteten er fin. Hvis man ikke vet helt hva man ser etter kan kategoriene hjelpe en på vei.

Veldig fint å kunne lete gjennom produkter enten ved søker, sortering eller kategorier. Gjør det både lett å finne noe spesifikt, og bare titte hvis du har lyst på noe innenfor en kategori.



Hvordan var brukeropplevelsen med å leie et produkt?

2 responses



Feedback

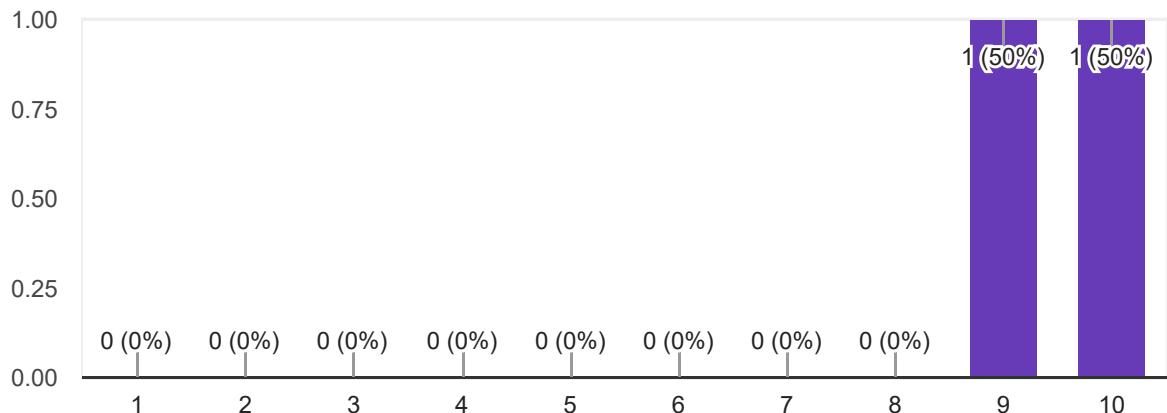
2 responses

Jeg likte at produktet jeg ville låne havnet inne på "My Borrows" med status "Pending approval".

Veldig rett fram!

Hvor brukervennlig var "My borrows" siden?

2 responses



Feedback

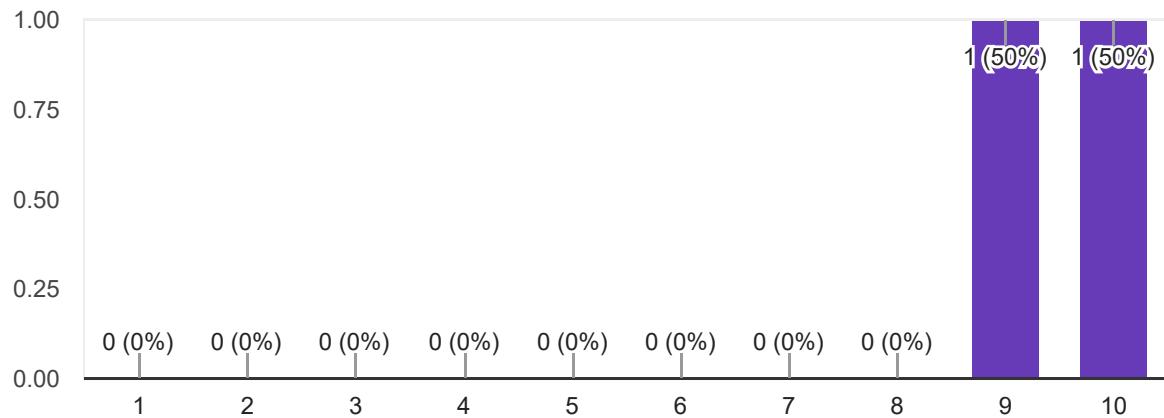
2 responses

Kort og oversiktlig.

Synes siden er veldig oversiktlig. Et veldig lite pikk er at jeg gjerne skulle hatt en knapp som er "Cancel request" eller tilsvarende istedenfor "Return" mens requesten min er under "Pending approval". Men veldig tydelig side!

Hva er din totale vurdering av produkt og leie funksjonalitet?

2 responses



Feedback

1 response

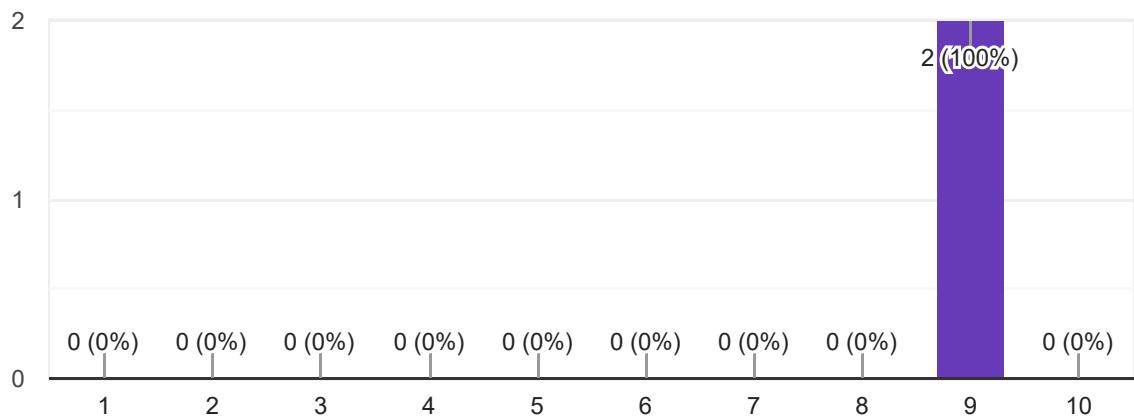
Veldig god, mesteparten av kritikken er bare små kosmetiske endringer som ville forbedret helhetsinntrykket. Veldig oversiktlige og intuitive sider.

Administrator dashboard



Hvordan likte du designet til Admin dashboard?

2 responses



Feedback

2 responses

Synes at designet er enkelt og oversiktlig. Kunne vært fint med en form for varsler på de seksjonene hvor det er ubehandlete forespørsler.

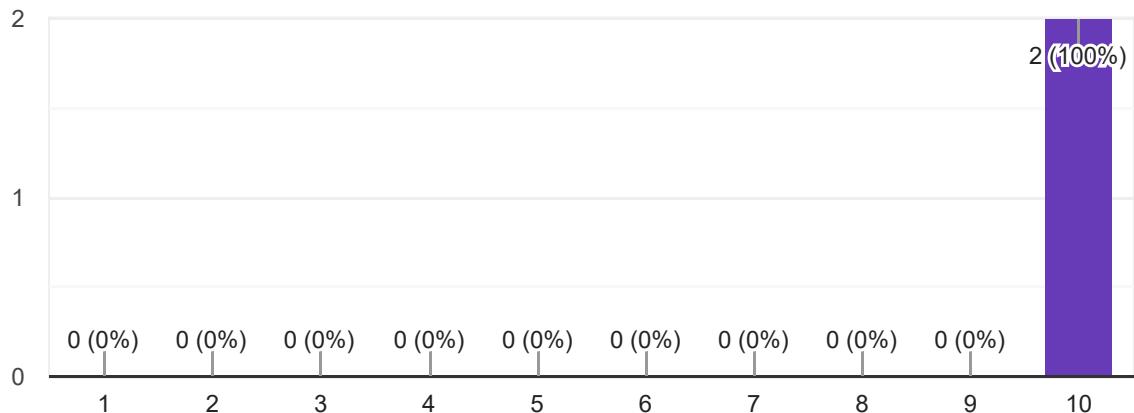
Samme som tidligere så tror jeg at noen farger/gråtoner hadde gjort dette enda bedre. Hvitt på hvitt skaper litt lite kontraster.

Liker hvordan siden ser ut ellers, tydelige og fine knapper. Tekst som kan klikkes på er tydelig, skjønner det basert på fargen. Liker det godt.



Hvor enkelt var det å navigere seg frem til ønsket administrator funksjonalitet?

2 responses



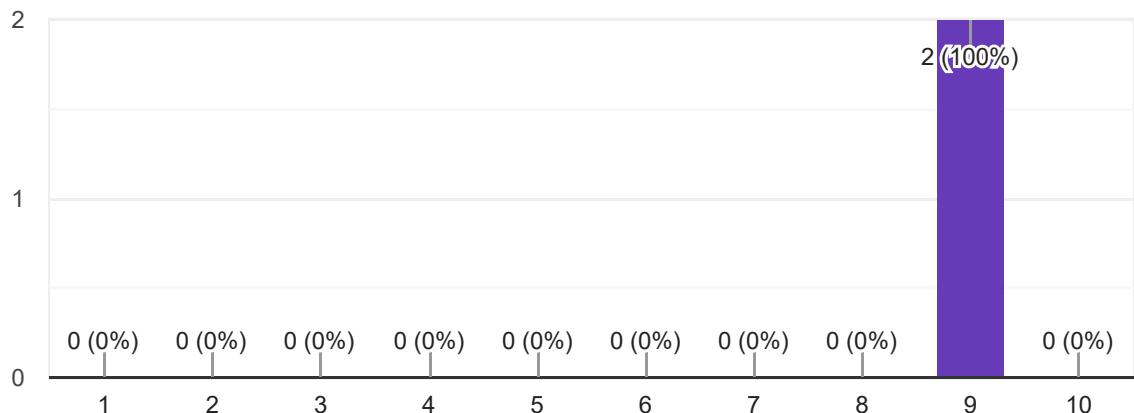
Feedback

1 response

Veldig enkelt, en topp bar blir brukt akkurat som den bør her. Eneste kommentar jeg har er å gjøre valgt side enda litt mer tydelig i top baren, typ med bold skrift. (Mener da top baren på hele siden, ikke den inne på Admin siden)

Hva er din totale vurdering av designet til admin dashboard?

2 responses



Feedback

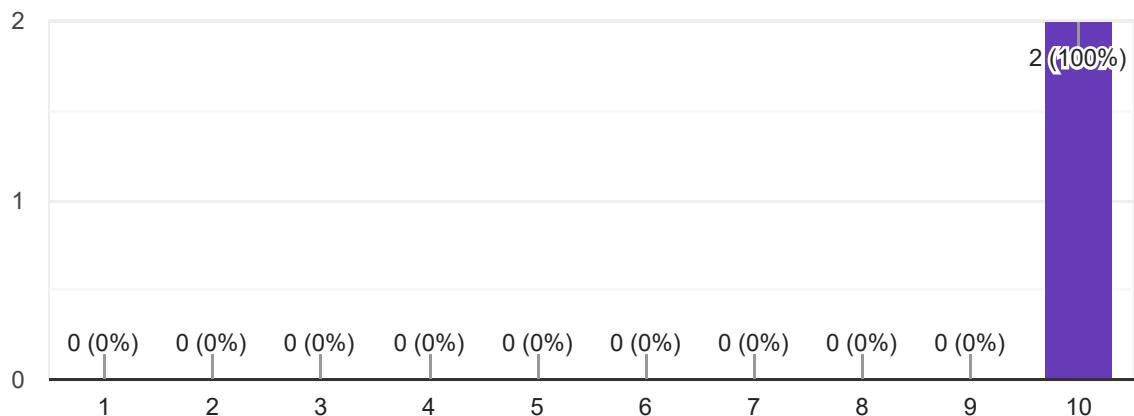
1 response

Liker det veldig godt. Enkelt å navigere, tydelige knapper. Eneste kommentarene jeg har for selve admin siden er som før fargevalg, og at ved opprettelse av nye produkter tror jeg det kan være lurt å støtte flere kategorier. For eksempel så kan man ha kategorien konsoll, men også ha kategorien Sony. Da er det fint at PS4 dukker opp for både Sony og Konsoll kategorien.

Administrering av produkter del 1

Hvor vanskelig eller enkelt er det å legge til et nytt produkt?

2 responses



Feedback

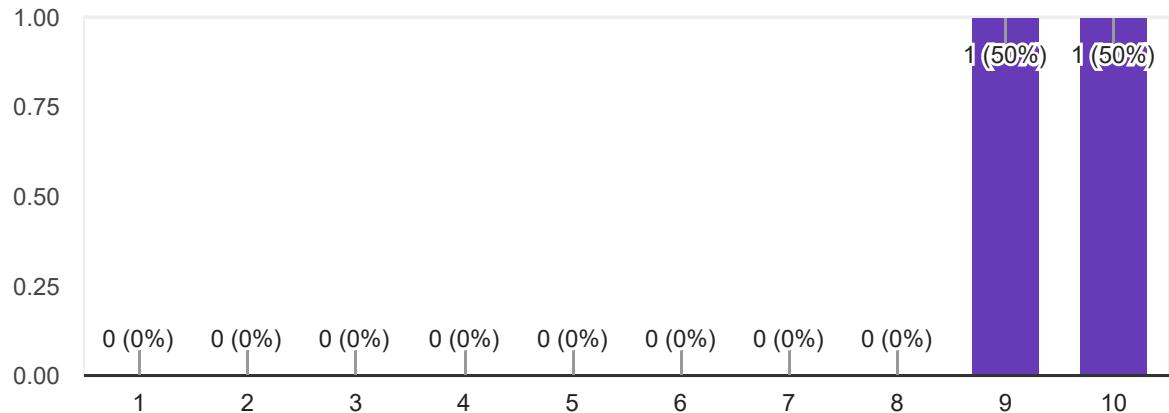
1 response

Veldig enkelt, nå kommenterer jeg ting i litt rekkefølge men skulle gjerne kunne lagt til flere kategorier.



Hvor oversiktlig og brukervennlig er "Products" menyen for administrator?

2 responses



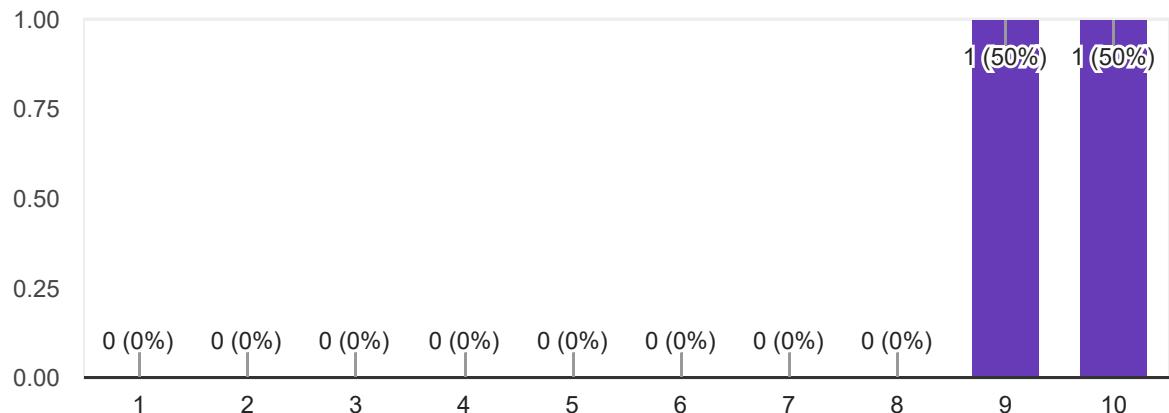
Feedback

1 response

Veldig oversiktlig og intuitivt. Kunne hatt et bedre fargeutvalg.

Hvor vanskelig eller enkelt var det å modifisere et produkt?

2 responses



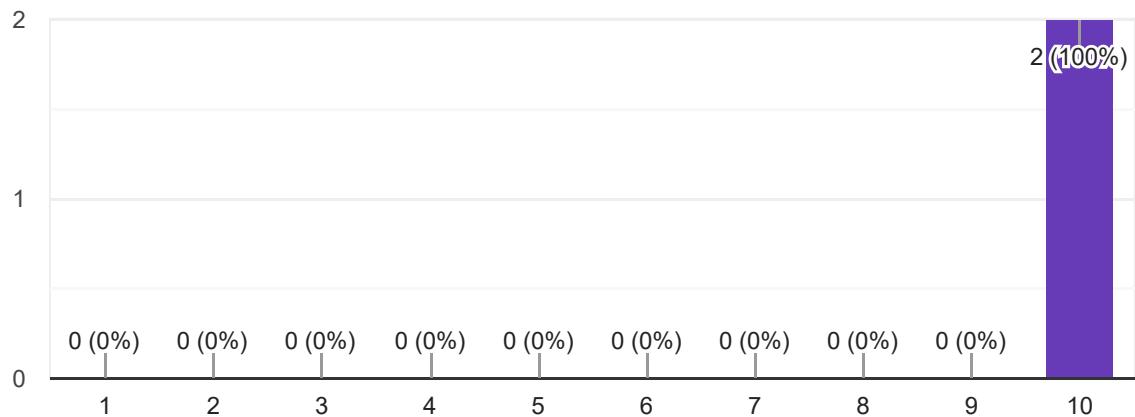
Feedback

1 response

Veldig enkelt å redigere et produkt.

Hvor vanskelig eller enkelt var det å legge til en ny entitet?

2 responses



Feedback

0 responses

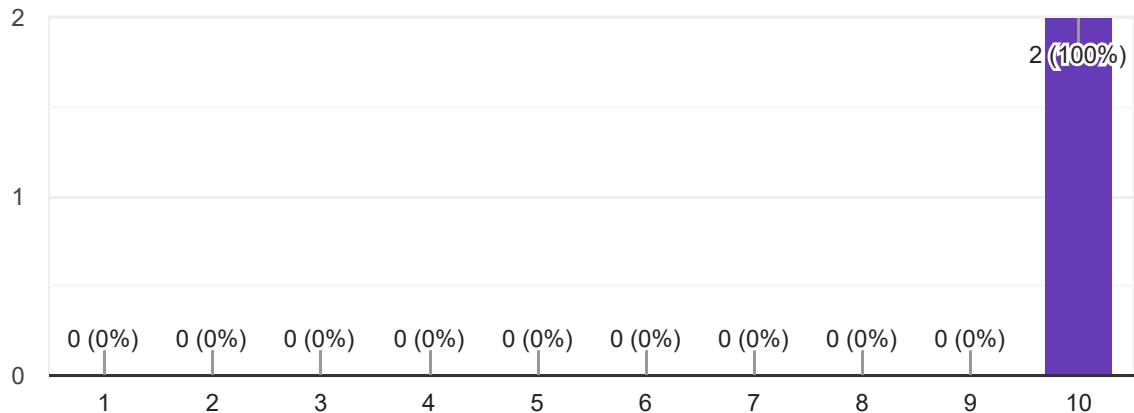
No responses yet for this question.

Administrering av kategorier



Hvor vanskelig eller enkelt var det å legge til en ny kategori?

2 responses



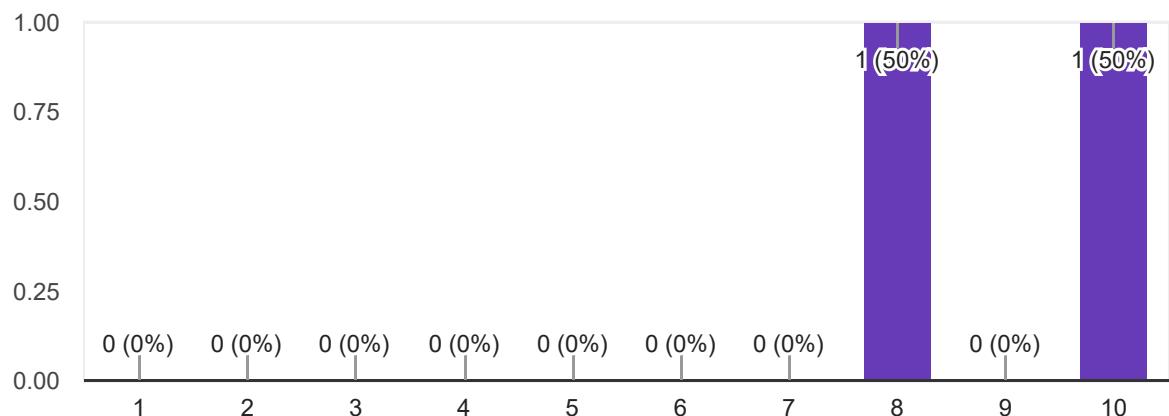
Feedback

1 response

Det var enkelt.

Hva er din totale vurdering av "Categories" funksjonaliteten?

2 responses



Feedback

2 responses

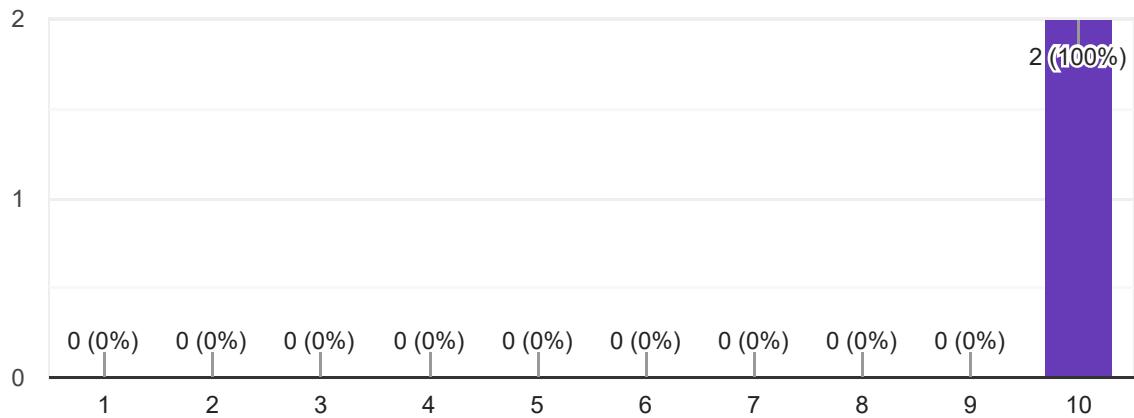
Det kunne evt. vært en beskrivende setning om at man må legge til parent-kategori før underkategori. (men det er egentlig noe man skjønner av seg selv når man prøver å legge til en kategori.)

Burde kunne gå an å endre kategorier hvis man gjør en typo eller glemmer parent category

Administrering av utleie

Hvor en vanskelig eller enkelt var det å godkjenne et utlån?

2 responses



Feedback

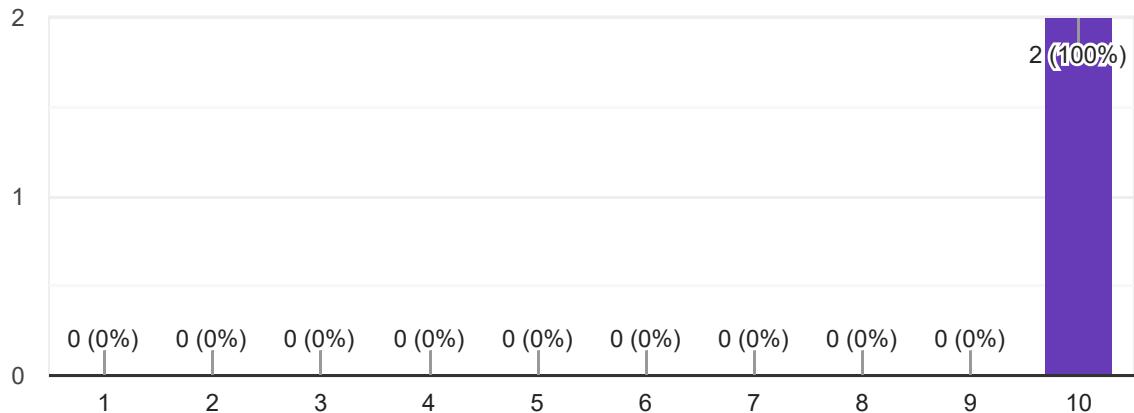
0 responses

No responses yet for this question.



Hvor oversiktlig var det å holde styr på sine utlån?

2 responses



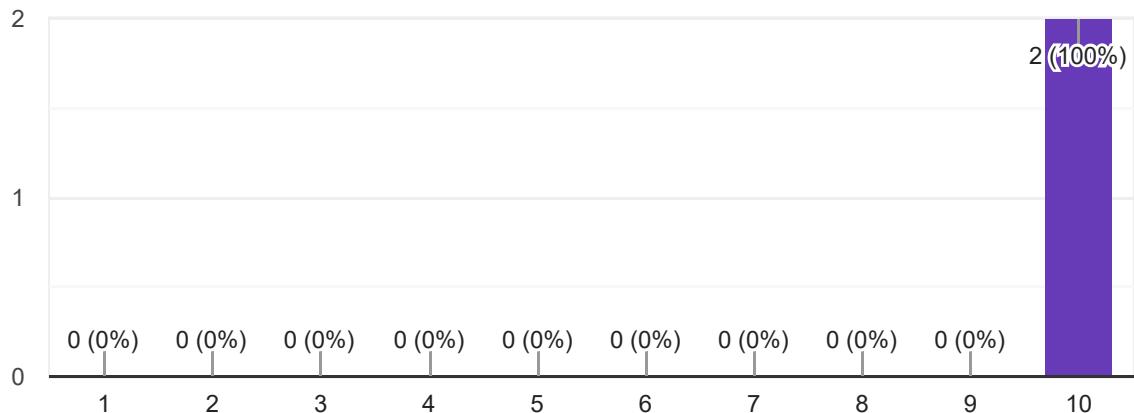
Feedback

0 responses

No responses yet for this question.

Hvor vanskelig eller enkelt var det å returnere et produkt?

2 responses



Feedback

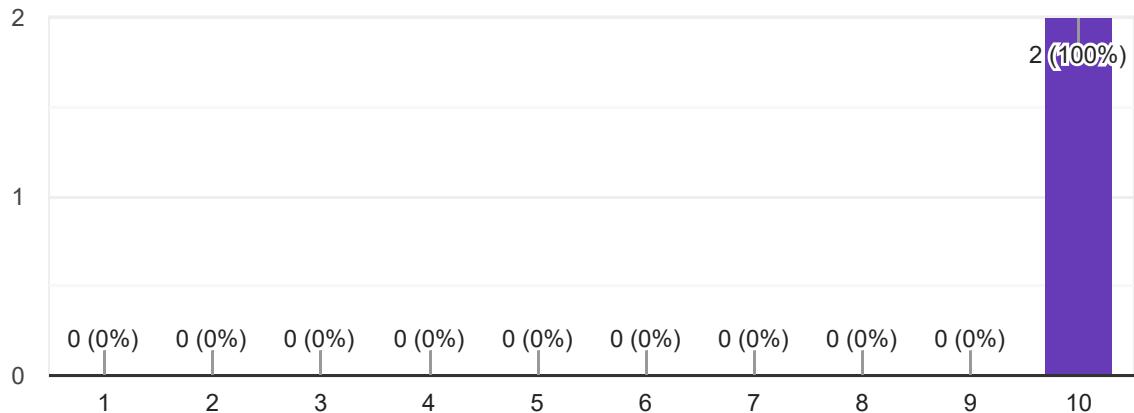
0 responses

No responses yet for this question.



Hvor vanskelig eller enkelt var det å godkjenne en retur av utleie?

2 responses



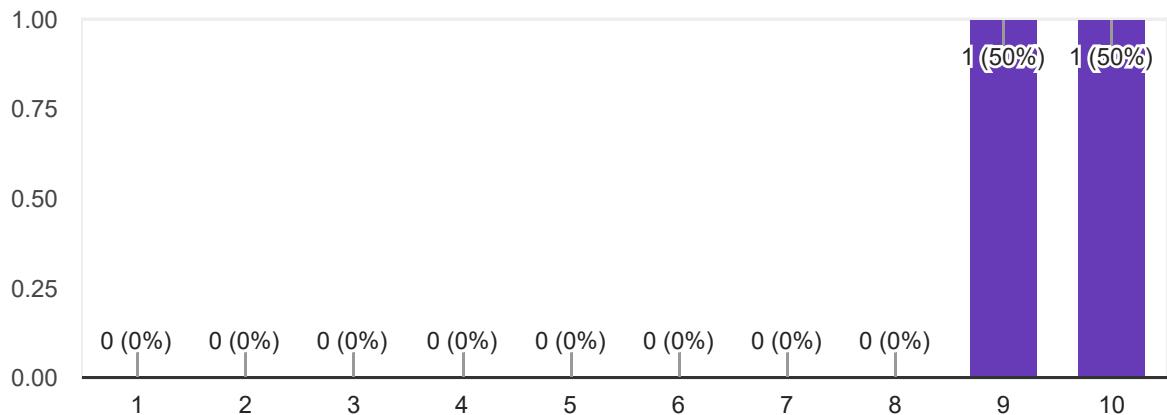
Feedback

0 responses

No responses yet for this question.

Hva er din totale vurdering av "Rentals" funksjonalitetene som administrator?

2 responses



Feedback

2 responses

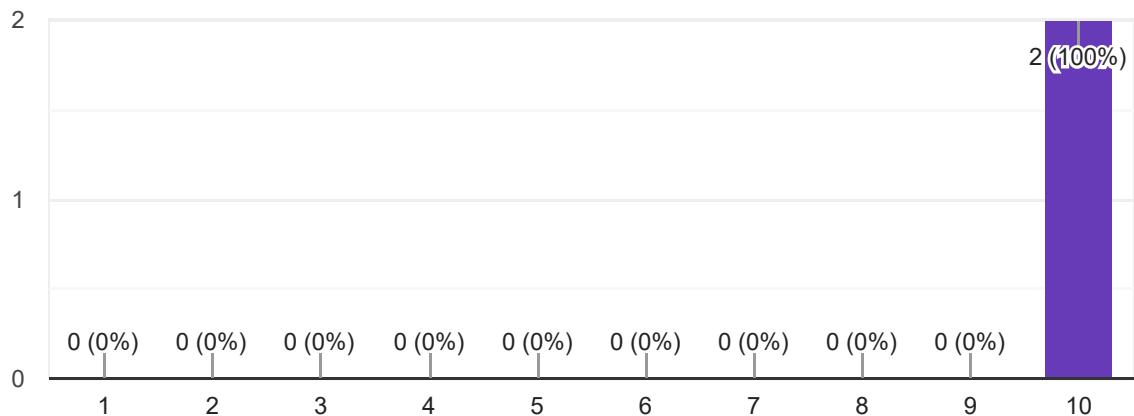
Enkelt og intuitivt.

Veldig fin og oversiktlig. Lurer på hva som skjer med informasjonen admin legger inn ved godkjenning av retur? Veldig fint at du kan filtrere ut andre renatals situasjoner enn requests.

Administrering av produkter del 2

Hvor enkelt var det å slette en entitet?

2 responses



Feedback

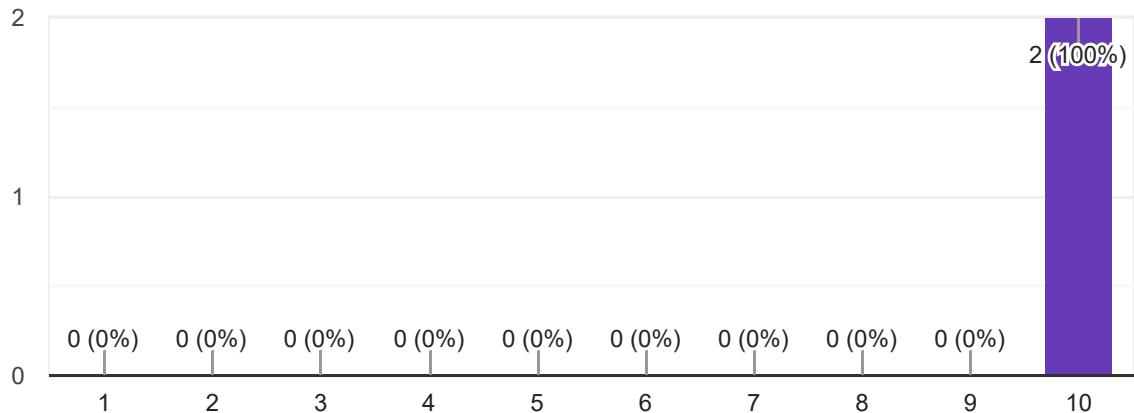
0 responses

No responses yet for this question.



Hvor enkelt var det å slette et produkt?

2 responses



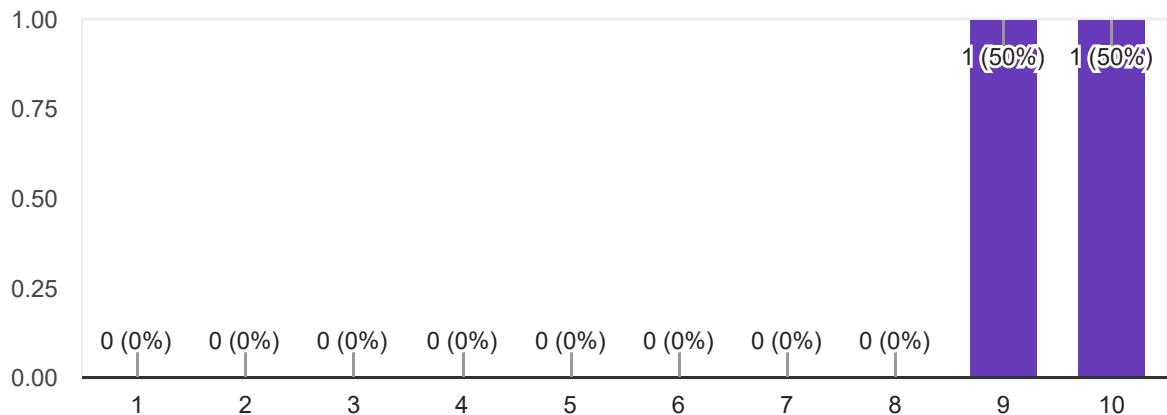
Feedback

0 responses

No responses yet for this question.

Hva er din totale opplevelse av "Products" funksjonaliteten til administrator?

2 responses



Feedback

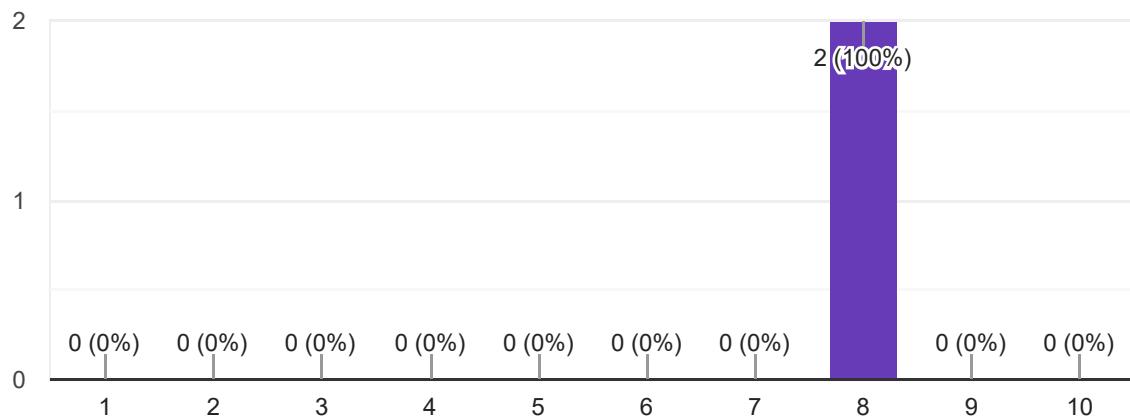
1 response

Veldig bra. Enkelt å lære, og rask i bruk. Ting er veldig intuitivt og krever minimalt med opplæring. Eneste kommentar at jeg ikke tydelig ser hva som skjer med det du skriver inn ved godkjenning av retur.

Administrering av brukere

Hvor praktisk var sorteringen av brukere?

2 responses



Feedback

2 responses

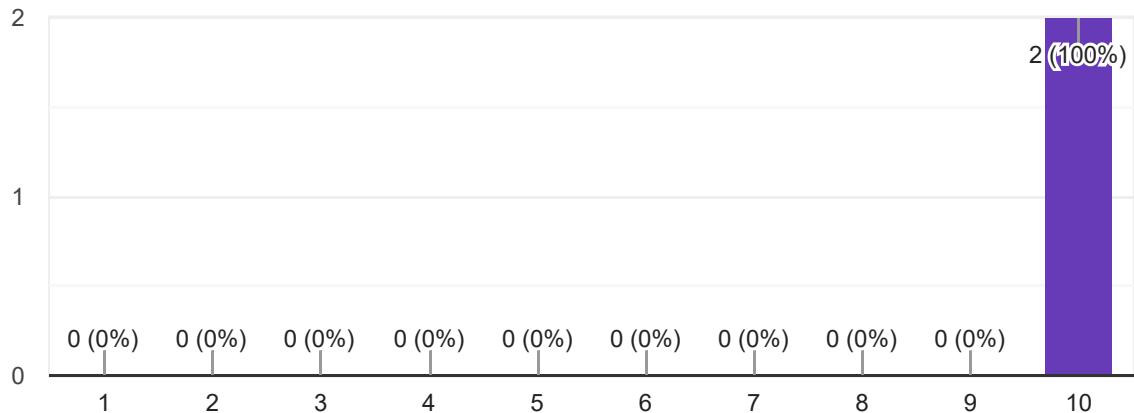
Det kunne ha vært sortering på etternavn også.

Veldig fint å kunne sortere. Skulle gjerne hatt søk funksjonalitet ettersom det er ganske mange medlemmer i gruppen.



Hvor enkelt vanskelig eller enkelt var det å endre rolle til bruker?

2 responses



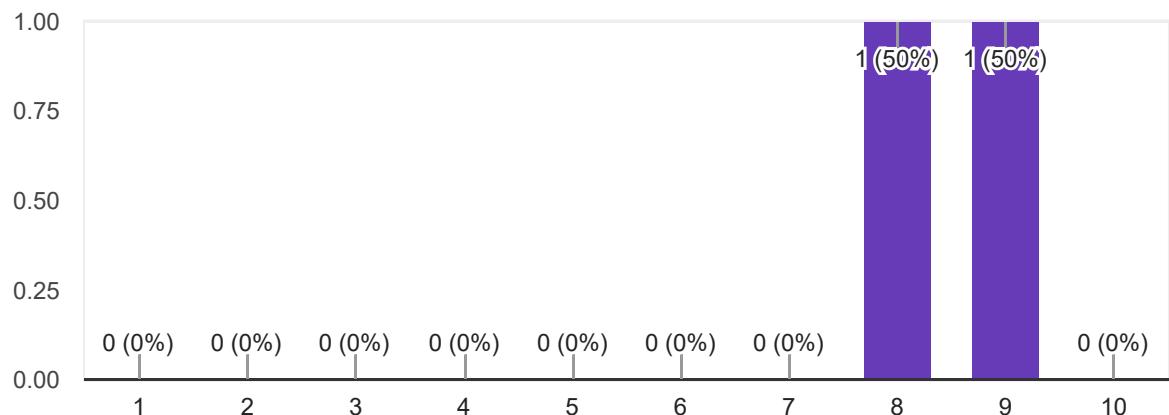
Feedback

1 response

Veldig enkelt.

Hva er din totale opplevelse av administrering av brukere?

2 responses



Feedback

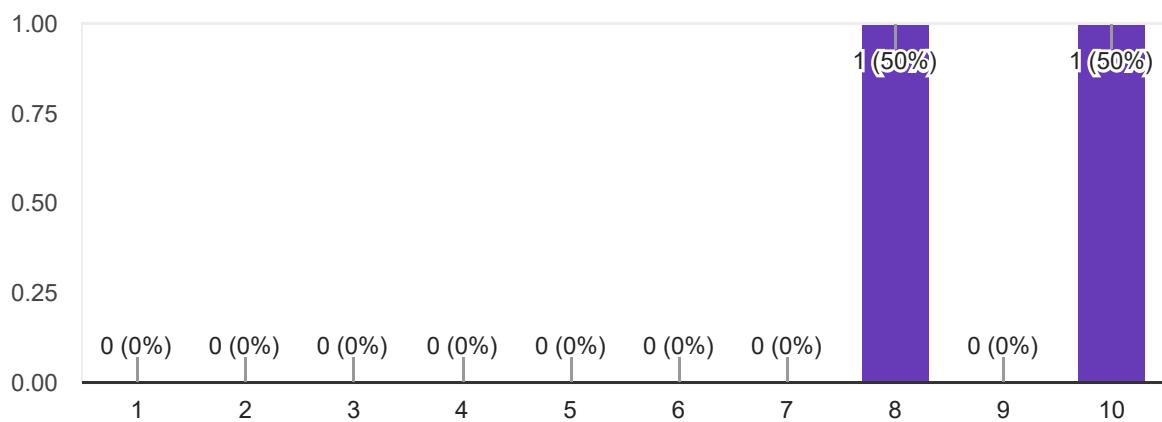
1 response

Veldig lett å bruke siden, eneste utfordringen er hvis det blir lagt til mange brukere som vi har i gaming gruppen så vil manglende søk funksjonalitet treffe.

Administrering av forslag

Hvor enkelt eller vanskelig var det å sende inn et forslag til et nytt produkt?

2 responses



Feedback

2 responses

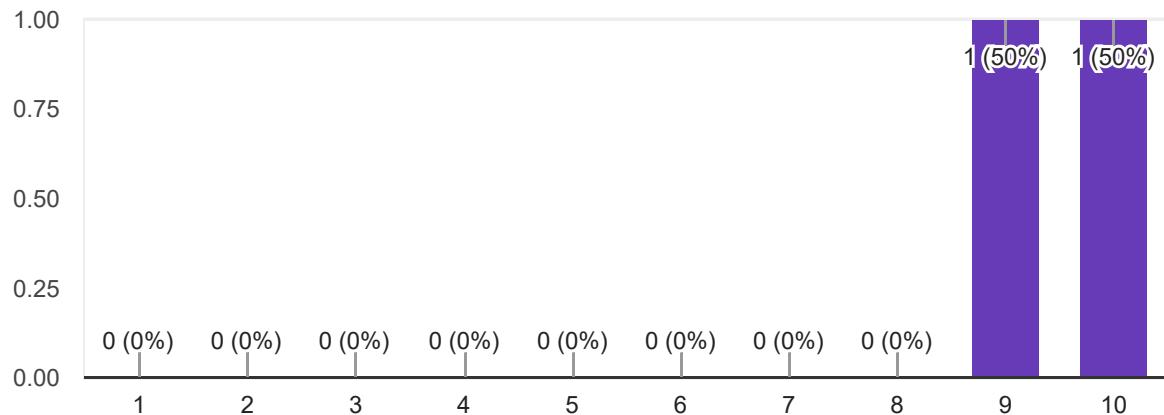
Ville legge til Risk, men det var et minimumskrav på 6 tegn.

Veldig enkelt. Eneste kritikk er at når man trykker legg til produkt så forsvinner ikke modalen. Teksten forblir også på siden selvom man krysser ut modalen også trykker add products knappen igjen.



Hvor vanskelig eller enkelt var det å slette et forslag?

2 responses



Feedback

2 responses

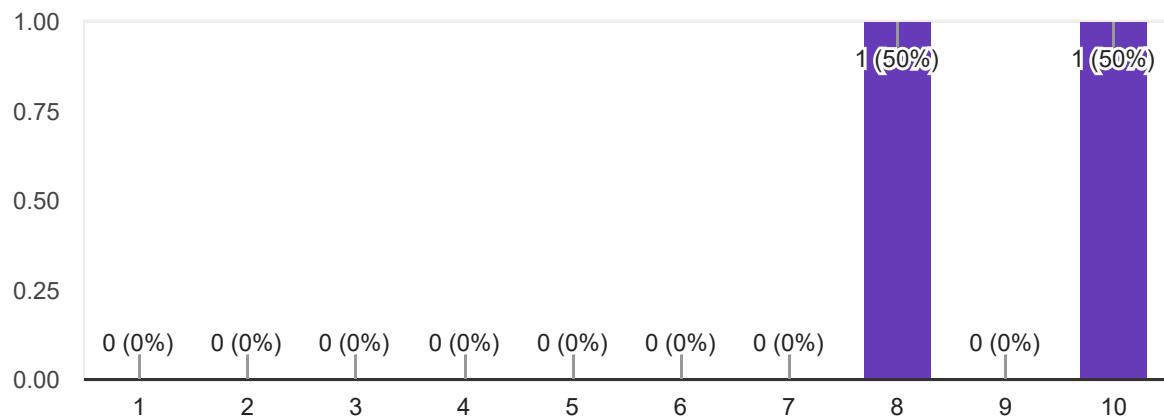
Det var enkelt.

Veldig enkelt å slette.

Småspørk er at jeg kunne likt å se antall suggestions i suggestions fanen på samme måte som products og categories.

Hvor fornøyd er du totalt av administrering av forslagsfunksjonalitet?

2 responses



Feedback

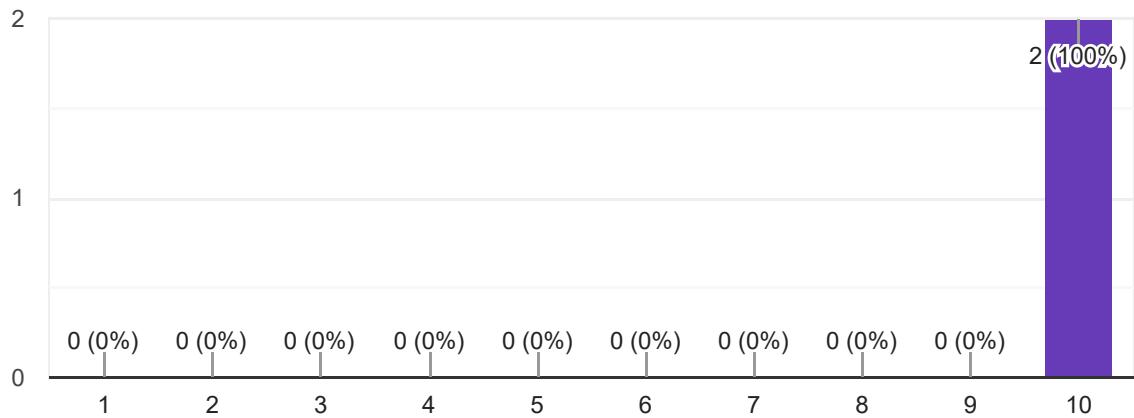
0 responses

No responses yet for this question.

Administrering av tjenestemeldinger

Hvor enkelt eller vanskelig var det å legge til en tjenestemelding?

2 responses



Feedback

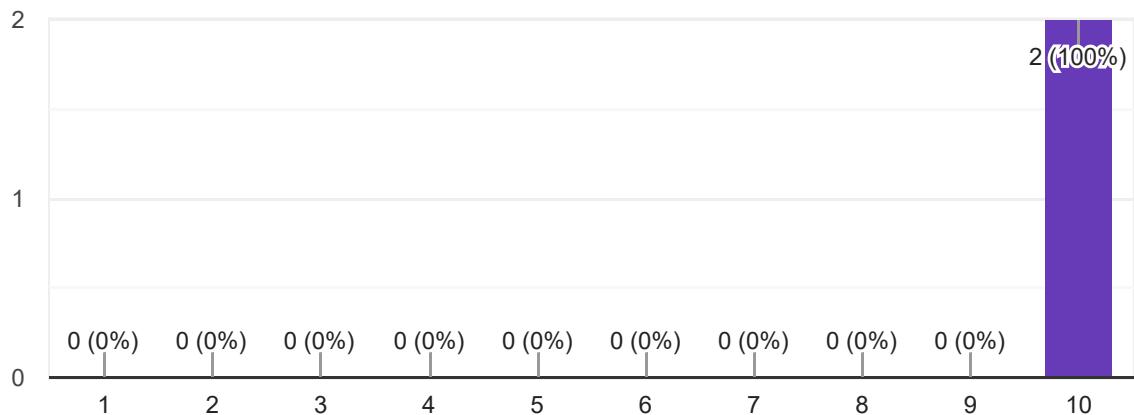
0 responses

No responses yet for this question.



Hvor intuitivt var det å krysse ut tjenestemeldingen?

2 responses



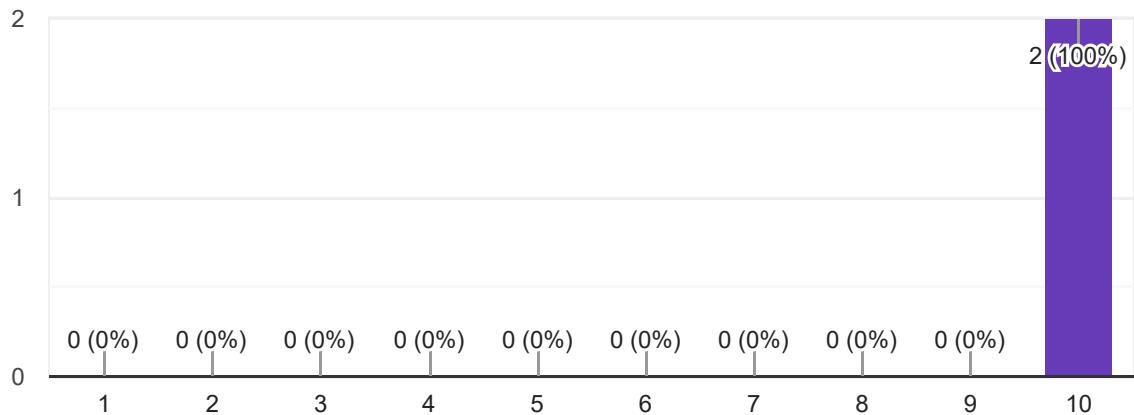
Feedback

0 responses

No responses yet for this question.

Hvor enkelt eller vanskelig var det å slette en tjenestemelding?

2 responses



Feedback

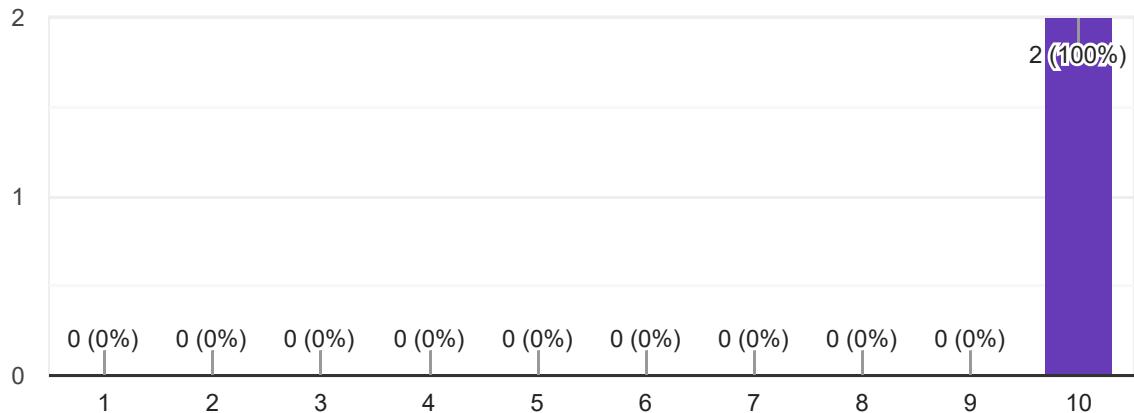
0 responses

No responses yet for this question.



Hvordan var din totale opplevelse av administrering av tjenestemeldinger?

2 responses



Feedback

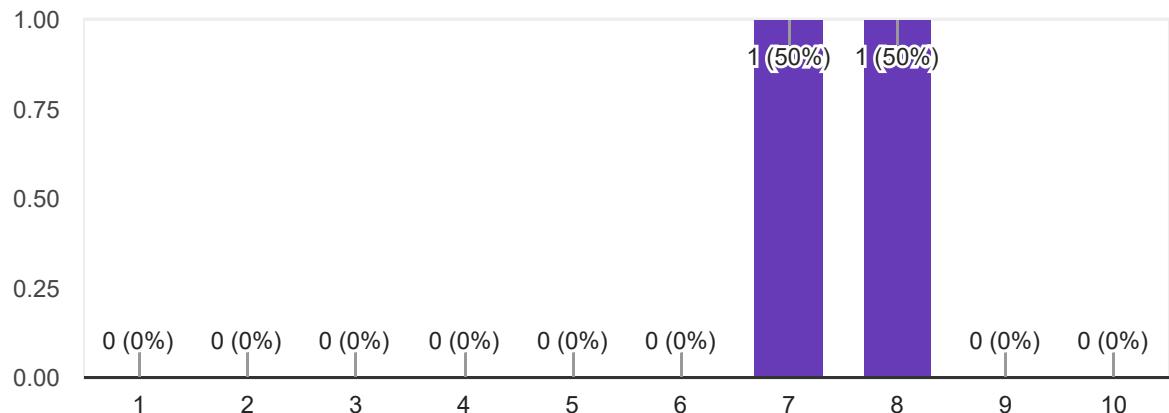
1 response

Veldig enkelt og greit å legge til tjenestemeldinger.

Helhetsvurdering og generelt inntrykk

Hvordan liker du designet til nettsiden?

2 responses



Feedback

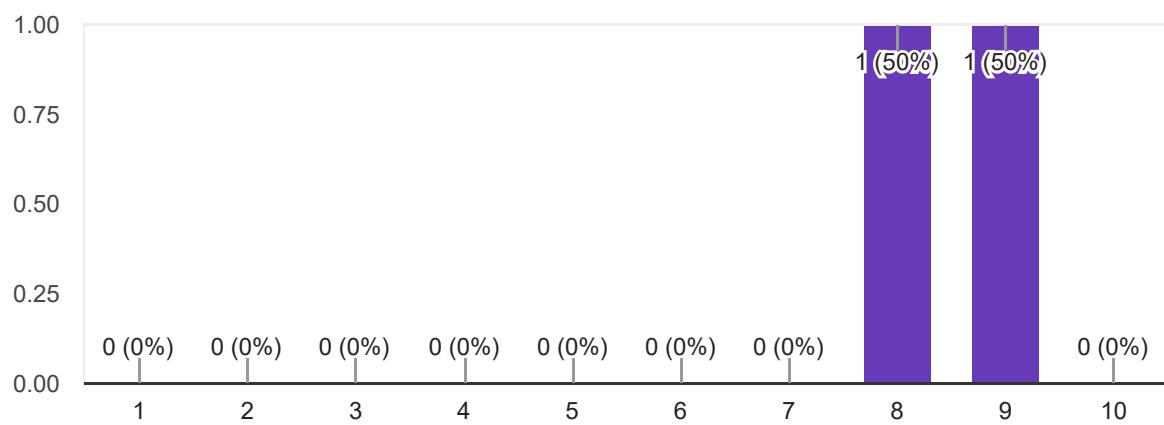
2 responses

Kunne vært penere, men det er enkelt og oversiktlig.

Designmessig er jeg stor fan av siden, men fargevalg har mye å si og her kunne det blitt gjort litt mer.

Hvordan vil du beskrive den totale opplevelsen av tjenesten?

2 responses



Feedback

1 response

Alt var intuitivt og raskt, noe småpirk her og der, men generelt veldig fornøyd.

Hva likte du mest med web-appen?

2 responses

At den fungerer slik som planlagt!

Hvor intuitivt det var.



Hva likte du minst?

2 responses

Det visuelle. Funkjsonelt, veldig bra!

Mangel på søkefunksjonalitet i brukerlisten.

Var det noe som overrasket deg over tjenesten?

0 responses

No responses yet for this question.

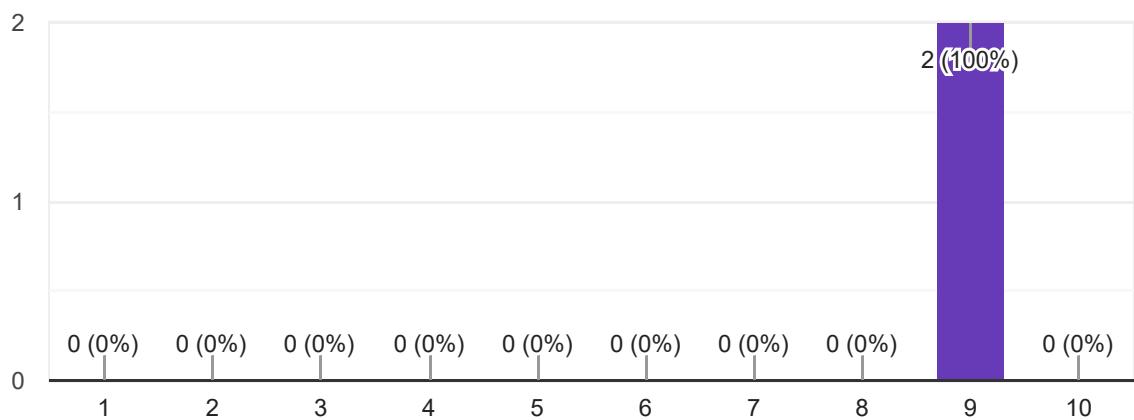
Var det noen frustrasjoner som oppstod ved bruk av tjenesten og i så fall hvilke?

1 response

At linkene sendt på mail ikke virket på min mobil. (Muligens pga accenture-blokninger.)

Totalt sett, hvor fornøyd er du med web-appen?

2 responses



Feedback

1 response

Synes dere har kommet veldig langt!

This content is neither created nor endorsed by Google. [Report Abuse](#) - [Terms of Service](#) - [Privacy Policy](#).

Google Forms



Tillegg I

Integrasjons og enhetstester

```
PASS tests/integration/routes/auth.test.js
api/auth
  POST /
    ✓ should return a JWT token if email and password is correct (236ms)
    ✓ Should return 400 if email is incorrect (93ms)
    ✓ Should return 400 if password is incorrect (145ms)
    ✓ Should return 400 if email is invalid (82ms)
    ✓ Should return 400 if password is invalid (82ms)
    ✓ Should return 403 if user is inactive(email has not been verified) (149ms)
POST /token/:jwt
  ✓ should return 400 if token is invalid (5ms)
  ✓ should return 200 if token is valid (13ms)
```

```
POST /auth/resetpassword
  ✓ should return 200 (9ms)
  ✓ should return 400 when email is not given (8ms)
```

```
PASS tests/integration/routes/rentals.test.js
```

```
/api/rentals
  GET /
    ✓ should return all rentals (105ms)
    ✓ should return only processed rentals (27ms)
    ✓ should return 401 if no JWT token is provided (15ms)
    ✓ should return 400 if JWT token is invalid (16ms)
    ✓ should return 403 if the authenticated user is not an admin (16ms)
  GET /returns/
    ✓ should return unprocessed returns (18ms)
    ✓ Should return processed returns (28ms)
    ✓ should return 401 if no JWT token is provided (16ms)
    ✓ should return 400 if JWT token is invalid (12ms)
    ✓ should return 403 if the authenticated user is not an admin (13ms)
  POST /
    ✓ should return new rental when requested with valid product that is available (54ms)
    ✓ should return 404 if product is not found (19ms)
    ✓ should return 400 if no entities are available (11ms)
    ✓ should return 401 if no JWT token is provided (6ms)
    ✓ should return 400 if JWT token is invalid (5ms)
```

```
PATCH("/:id")
  ✓ should return 200 if request body and admin token is provided (27ms)
  ✓ should return rental with dateOut on 200 response (17ms)
  ✓ should return 400 if pickUpInstructions is not provided (10ms)
  ✓ should return 400 if returnInstructions is not provided (9ms)
  ✓ should return 400 if rental id is not a valid mongoose objectId (9ms)
  ✓ should return 404 if rental id doesn't exist (12ms)
  ✓ should return 401 if no JWT token is provided (12ms)
  ✓ should return 400 if JWT token is invalid (13ms)
  ✓ should return 403 if user is not admin (18ms)
```

```
POST /returns/:id
  ✓ should return 200 when rental exists (26ms)
  ✓ should return rental body on 200 response (16ms)
  ✓ should return 404 when rental does not exist (11ms)
  ✓ should return 400 when rental id is invalid (9ms)
  ✓ should return 400 when user in JWT token is not the same as in rental document (13ms)
  ✓ should return 401 if no JWT token is provided (11ms)
  ✓ should return 400 if JWT token is invalid (13ms)
```

```
PATCH /returns/:id
  ✓ should return 200 when return is successful (36ms)
  ✓ should set confirmedReturned to true in rental document (28ms)
  ✓ should set availableforRental to true in product document (36ms)
  ✓ should set availableforRental to false in product document (18ms)
  ✓ should return 400 if objectId is invalid (12ms)
  ✓ should return 404 if rental does not exist (12ms)
  ✓ should return 400 if setAvailable is missing in request body (20ms)
  ✓ should return 404 if rental does not exist (15ms)
  ✓ should return 401 if authentication token is not provided (19ms)
  ✓ should return 400 if authentication token is invalid (15ms)
  ✓ should return 403 if user is not admin (10ms)
```

```
DELETE("/:id")
  ✓ should delete rental with given ID (48ms)
  ✓ should return 404 if an item with the given ObjectId doesn't exist (11ms)
  ✓ Should return 400 if the ID provided is invalid (9ms)
  ✓ should return 401 if no JWT token is provided (8ms)
  ✓ should return 400 if JWT token is invalid (8ms)
  ✓ should return 403 if JWT token is not of an admin user (13ms)
```

```
PASS tests/integration/routes/users.test.js
```

```
/api/users
```

```
  GET /
```

- ✓ should return all users when requested by an authenticated admin (68ms)
- ✓ should return 401 if no JWT token is provided (6ms)
- ✓ should return 400 if JWT token is invalid (5ms)
- ✓ should return 403 if the authenticated user is not an admin (5ms)

```
  GET /me
```

- ✓ should return the currently authenticated user (16ms)
- ✓ should return 401 if no JWT token is provided (7ms)
- ✓ should return 400 if JWT token is invalid (11ms)

```
  GET /:id
```

- ✓ should return 401 if no JWT token is provided (8ms)
- ✓ should return 400 if JWT token is invalid (5ms)
- ✓ should return 403 if the authenticated user is not an admin (8ms)
- ✓ should return a user when a valid ID is passed (14ms)
- ✓ should return 400 if user ID is invalid (7ms)
- ✓ should return 404 if user does not exist (9ms)

```
  GET /:id/rentals
```

- ✓ should return 401 if the user that request another users rentals is not admin (11ms)
- ✓ should return 200 if the user that requests another users rentals is an admin (16ms)
- ✓ should return 200 if a user requests it's own rentals (55ms)
- ✓ should return 401 if no JWT token is provided (9ms)
- ✓ should return 400 if JWT token is invalid (5ms)

```
  POST /
```

- ✓ should return new user when user is valid (93ms)
- ✓ Should return 400 when firstName is not given (11ms)
- ✓ Should return 400 when lastName is not given (5ms)
- ✓ Should return 400 when lastName is not given (5ms)
- ✓ Should return 400 when email is not given (5ms)
- ✓ Should return 400 when password is not given (5ms)
- ✓ Should return 400 when phone is not given (5ms)
- ✓ Should return 400 if user is already registered (98ms)

```
  PUT /:id
```

- ✓ should return new user when valid token and user is passed (22ms)
- ✓ should return 404 when the user to be updated doesn't exist (13ms)
- ✓ should return 400 when name is not provided (11ms)
- ✓ should return 400 when email is not provided (13ms)

```
  DELETE /:id
```

- ✓ should return 401 if no JWT token is provided (6ms)
- ✓ should return 400 if JWT token is invalid (6ms)
- ✓ should return 403 if the authenticated user is not an admin (6ms)
- ✓ should return 400 if product ID is invalid (6ms)
- ✓ should return 404 if user not found (11ms)
- ✓ should delete user if found (14ms)
- ✓ should return deleted user (8ms)

```
  POST /newPassword/:token
```

- ✓ Should change users password (151ms)
- ✓ Should return 400 when new password is missing (14ms)
- ✓ Should return 401 when token is invalid (12ms)

```
PASS | tests/integration/routes/products.test.js
```

```
/api/products
```

```
GET /
```

- ✓ should return all products (63ms)
- ✓ should return 401 if no JWT token is provided (19ms)
- ✓ should return 400 if JWT token is invalid (23ms)

```
GET /:id
```

- ✓ should return one product (29ms)
- ✓ should return 400 if product ID is invalid (15ms)
- ✓ should return 404 if product does not exist (14ms)

```
POST /
```

- ✓ should return 401 if no JWT token is provided (25ms)
- ✓ should return 400 if JWT token is invalid (14ms)
- ✓ should return 403 if the authenticated user is not an admin (10ms)
- ✓ should return new product when product is valid (23ms)
- ✓ should return 400 when name is not given (15ms)
- ✓ should return 400 when category.name is not given (13ms)
- ✓ should return 400 when entities is not given (13ms)
- ✓ should return 400 when numberOfLoans is a negative number (18ms)
- ✓ should return 400 when description is empty (12ms)
- ✓ should return 400 when details is undefined (14ms)
- ✓ should return 404 when category does not exist in database (15ms)
- ✓ should return 400 when category has an invalid ID (12ms)

```
PUT /:id
```

- ✓ should return 401 if no JWT token is provided (10ms)
- ✓ should return 400 if JWT token is invalid (10ms)
- ✓ should return 403 if the authenticated user is not an admin (9ms)
- ✓ should return 400 if product ID is invalid (10ms)
- ✓ should return 400 if product is invalid (14ms)
- ✓ Should return 404 if product is not found (14ms)
- ✓ should return updated product (28ms)

```
DELETE /:id
```

- ✓ should return 401 if no JWT token is provided (9ms)
- ✓ should return 400 if JWT token is invalid (10ms)
- ✓ should return 403 if the authenticated user is not an admin (10ms)
- ✓ should return 400 if product ID is invalid (9ms)
- ✓ should return 404 if product not found (13ms)
- ✓ should delete product if found (16ms)
- ✓ should return deleted product (13ms)

```
PASS | tests/integration/routes/categories.test.js
```

```
/api/categories
```

```
GET /
```

- ✓ should return all categories (54ms)
- ✓ should return 401 if no JWT token is provided (12ms)
- ✓ should return 400 if JWT token is invalid (7ms)

```
GET /:id
```

- ✓ should return 401 if no JWT token is provided (7ms)
- ✓ should return 400 if JWT token is invalid (7ms)
- ✓ should return one category (10ms)
- ✓ should return 404 when category with given id does not exist (8ms)
- ✓ should return 400 when id is invalid (8ms)

```
POST /
```

- ✓ should return 401 if no JWT token is provided (13ms)
- ✓ should return 400 if JWT token is invalid (6ms)
- ✓ should return 403 if the authenticated user is not an admin (7ms)
- ✓ should return new category with parent (21ms)
- ✓ should return a new category without parent (12ms)
- ✓ should return 400 when parent id is not a valid id (9ms)
- ✓ should return 400 when parent has a parent (9ms)
- ✓ should return 400 when parent name is given (8ms)
- ✓ should return 404 when parent with given id does not exist (9ms)

```
DELETE /:id
✓ should return 401 if no JWT token is provided (11ms)
✓ should return 400 if JWT token is invalid (8ms)
✓ should return 403 if the authenticated user is not an admin (6ms)
✓ should return deleted category (14ms)
✓ should delete category (18ms)
✓ should return 404 when category with given id does not exist (7ms)
✓ should return 400 when id is invalid (6ms)
```

```
PUT /:id
✓ should return 401 if no JWT token is provided (7ms)
✓ should return 400 if JWT token is invalid (10ms)
✓ should return 403 if the authenticated user is not an admin (7ms)
✓ should return updated category (15ms)
✓ should return 400 if parent has parent (10ms)
✓ should return 404 if parent does not exist (8ms)
```

PASS tests/integration/routes/suggestions.test.js

```
/api/suggestions
GET /
✓ should return all suggestions (55ms)
✓ should return 401 if no JWT is provided (5ms)
✓ should return 400 if JWT is invalid (5ms)
✓ should return 403 if the authenticated user is not an admin (4ms)
```

```
POST /
✓ Should save and return the newly created suggestion, given valid suggestion (24ms)
✓ Should return 400 if suggestion is not provided (6ms)
✓ should return 400 if JWT token is invalid (5ms)
✓ should return 401 if no JWT is provided (5ms)
```

```
DELETE /
✓ should delete suggestion with given ID (26ms)
✓ should return 404 if the suggestion with the given ID does not exist (8ms)
✓ should return 400 if suggestionId is invalid (6ms)
✓ should return 400 if JWT token is invalid (6ms)
✓ should return 401 if no JWT is provided (6ms)
✓ should return 403 if JWT token is not of an admin user (6ms)
```

PASS tests/integration/routes/serviceMessages.test.js

```
/api/serviceMessages
GET /
✓ Should return all serviceMessages (47ms)
✓ should return 400 if JWT is invalid (5ms)
✓ should return 401 if no JWT is provided (7ms)
```

```
POST /
✓ Should save and return the newly created service message, given valid service message (36ms)
✓ should return 400 if service message is not provided (6ms)
✓ should return 400 if service message is too long (6ms)
✓ should return 400 if jwt is invalid (4ms)
✓ should return 401 if no JWT is provided (5ms)
✓ should return 403 if the authenticated user is not an admin (8ms)
```

```
DELETE /
✓ should delete service message with the given ID (21ms)
✓ should return 404 if the service message with the given ID does not exist (7ms)
✓ should return 400 if service message ID is invalid (8ms)
✓ should return 400 if jwt is invalid (5ms)
✓ should return 403 if the authenticated user is not an admin (6ms)
✓ should return 401 if no JWT is provided (6ms)
```

```
PASS  tests/integration/middleware/auth.test.js
auth middleware
  ✓ should return an authenticated user if supplied jwt token is valid (60ms)
  ✓ should return 401 if no token is provided (6ms)
  ✓ should return 400 if token is invalid (4ms)
  ✓ should return 400 if an email verification token is supplied (4ms)
```

```
console.log startup/db.js:14
Connected to DB: mongodb://localhost:27017/loan-app_test
```

```
PASS  tests/integration/middleware/admin.test.js
admin middleware
  ✓ should return 403 if user is not admin (49ms)
  ✓ should not return 200 if user is admin (9ms)
```

```
console.log startup/db.js:14
Connected to DB: mongodb://localhost:27017/loan-app_test
```

```
PASS  tests/unit/middleware/auth.test.js
auth middleware
  ✓ should populate req.user with the payload of a valid JWT (9ms)
```

```
PASS  tests/unit/middleware/validateObjectId.test.js
validateObjectId middleware
  ✓ should call next() if ObjectId is valid (2ms)
  ✓ should return 400 if ObjectId is invalid (2ms)
```

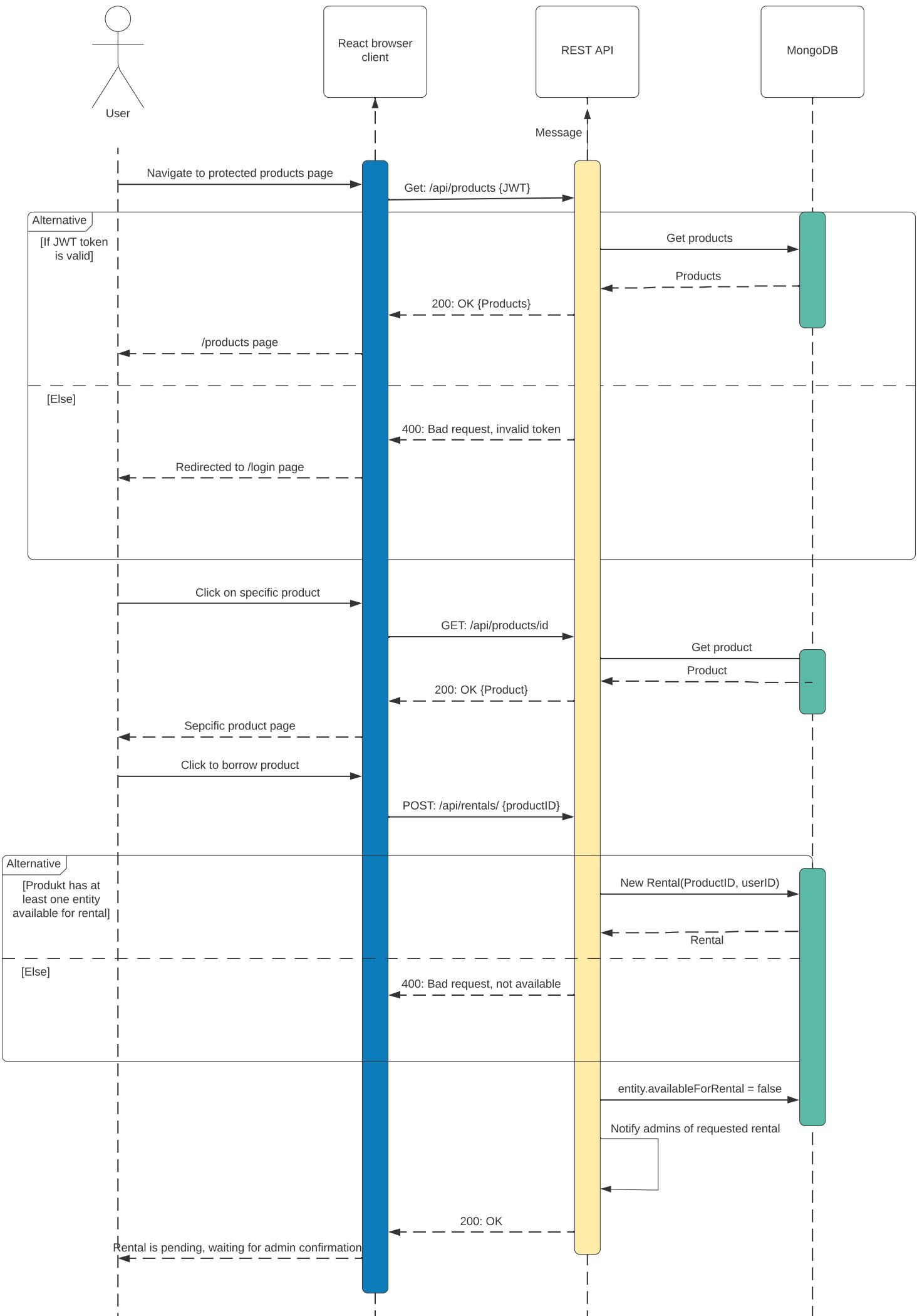
```
PASS  tests/unit/models/user.test.js
user.generateAuthToken
  ✓ should return a valid JWT for admin user (7ms)
  ✓ should return a valid JWT for non admin user (1ms)
```

```
Test Suites: 12 passed, 12 total
Tests:       200 passed, 200 total
Snapshots:   0 total
Time:        14.018s
Ran all test suites.
```

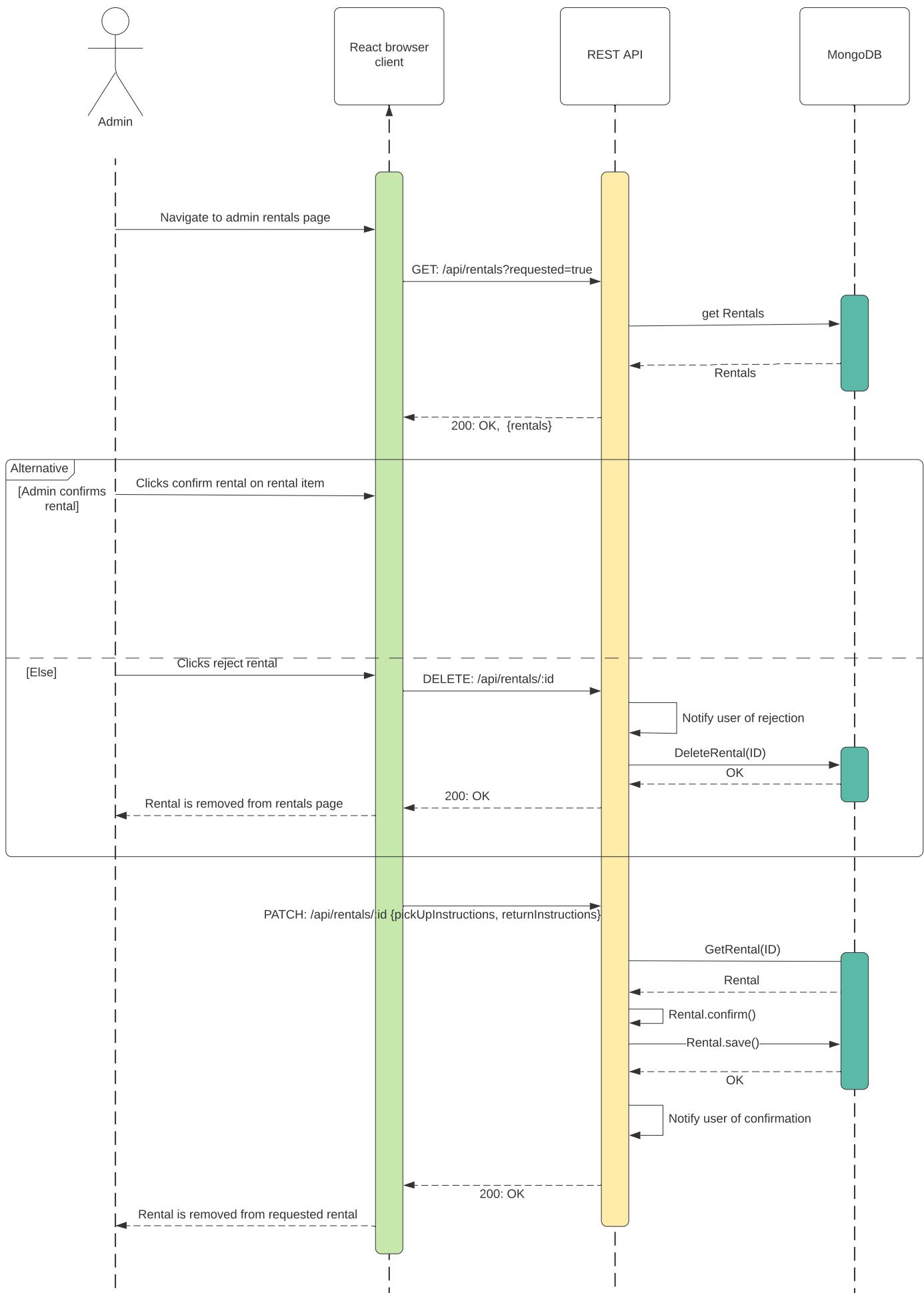
Tillegg J

**Sekvensdiagram av
utlånslivssyklusen**

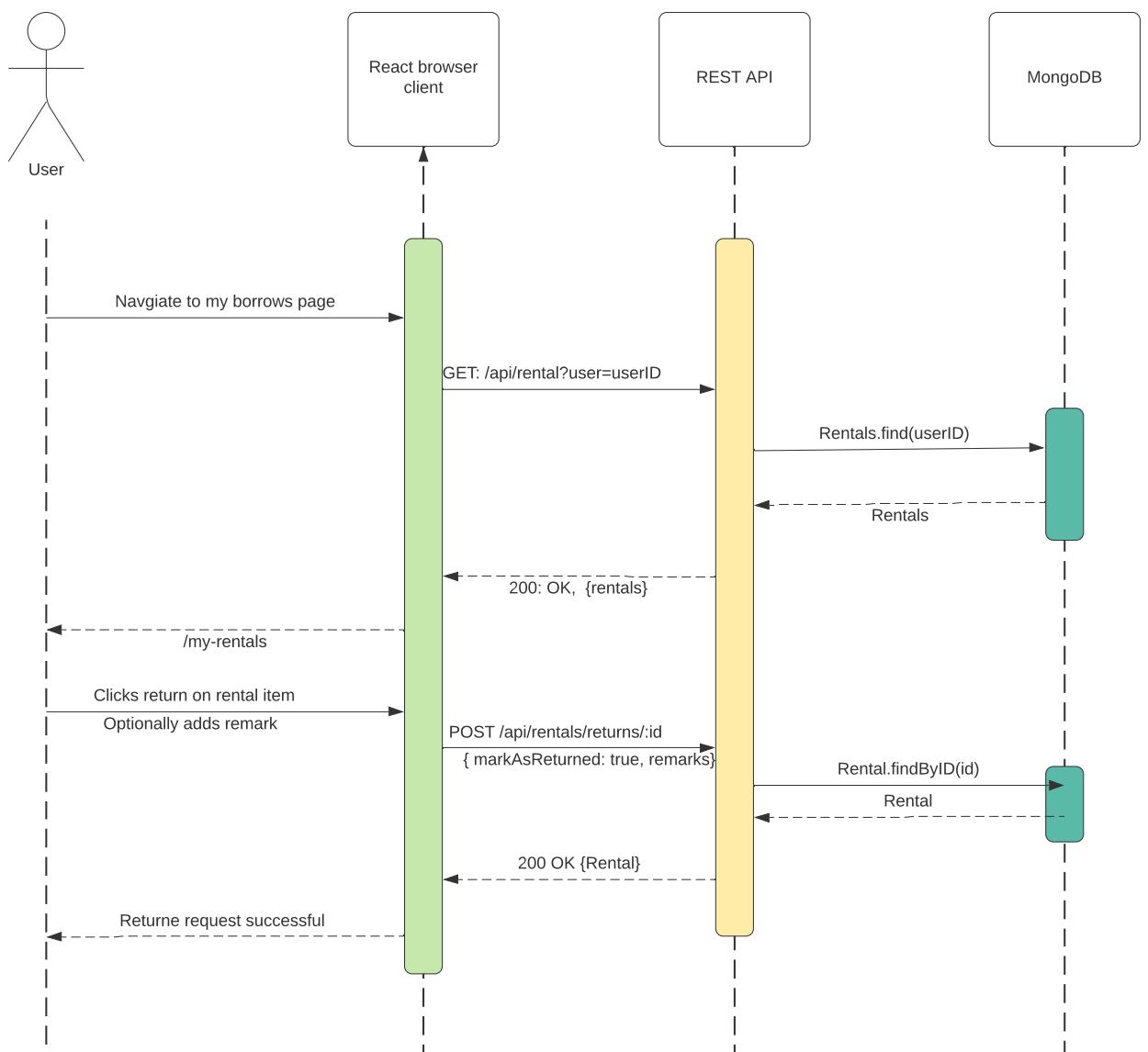
Request Rental



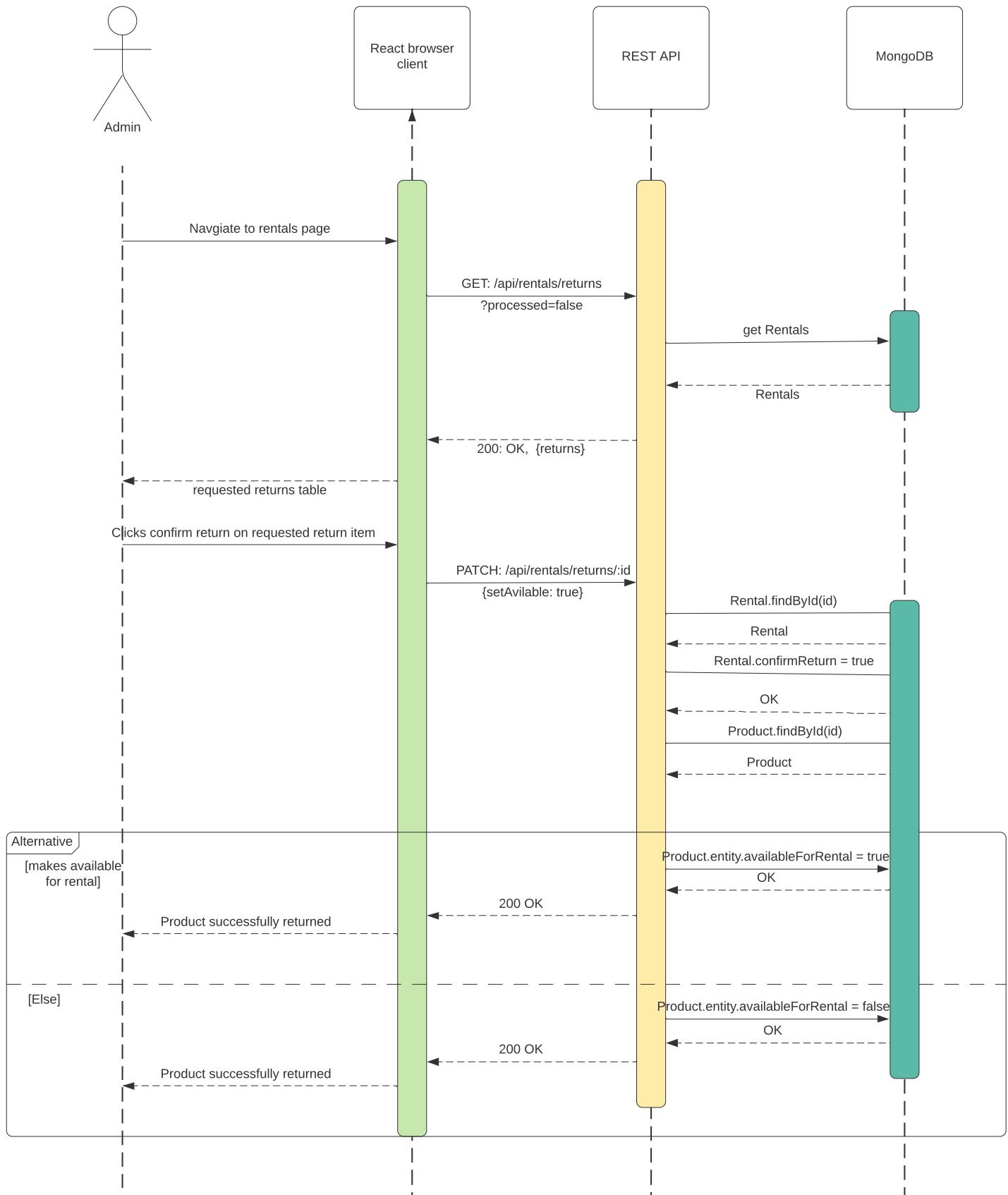
Confirm Rental



Request Return



Confirm Return



Tillegg K

Ukesrapporter

Uke 3: Oppstart Accenture, FORM kurs

Hva ble gjort:

- **13/1:** Felles oppstart med alle gruppene Accenture er veileder for. Studentene introduserte seg selv, fikk adgangskort og utdelt laptop til utlån under bacheloroppgaven. Videre fikk gruppene en gjennomgang av møteromskultur og dresskode fra HR. Deretter ble det visning av lokalet hvor det også ble anvist reserverte sitteplasser. Videre informasjon ble gitt om styringsgruppemøte og datoer ble notert for eksamen og innleveringer.
- **16/1:** Gruppen gjennomførte sitt første kurs hos Accenture, FORM kurs. Dette kurset gjennomgikk deler av Accenture sin arbeidsmetodikk. Kurset var delt med en teori og en praktisk del hvor i den praktiske delen brukte vi arbeidsmetodikken til å bedre forme bacheloroppgaven. Notater fra FORM kurs:

Discover:

- Se det store bildet
- Hvor befinner vi oss nå

Describe:

- Hvor er vi, hvorfor...

Co-CREATE:

- MVP, hvordan kan vi løse problemet på enklast mulig måte. Ideer -> prototyper.

Scale:

- Hvordan kan dette skaleres, og gir det mening?

Hva skal gjøres:

- Planlegge møte med veileder fra Accenture i uke 4. Forprosjektrapport. Lage gruppeside.

Hvilke utfordringer ble møtt:

- Det ble utfordringer med innleveringsfristen for forprosjektrapporten da oppstart hos Accenture kolliderte med gitt frist hos OsloMet. Dette ble løst med utvidet innleveringsfrist.

Uke 4: Oppstart Accenture, FORM kurs.

Hva ble gjort:

- **21/1:** Møte med veiledere fra Accenture (21/1)
- **23/1:** Møte med veileder fra OsloMet (23/1)
- Jobbet med og fullførte forprosjektrapport med fremdriftsplan og diskusjon rundt valg av teknologi.

Hva skal gjøres:

- Opplæring i NodeJS, Express og React. Gruppen fullfører kurs på nett.
- Forslag til kravspesifikasjon utarbeides i løpet av de neste 2 ukene og forhører oss med veiledere angående dette.
- Lage presentasjon til styringsgruppemøtet 12. Februar

Hvilke utfordringer ble møtt:

- Ikke fått skjermer hos Accenture enda
- Venter på dato for kurs hos Accenture, spesifikt Git og React kurset.

Uke 5: Opplæring i Node.js og React.js

Hva ble gjort:

- Gruppemedlemmene har jobbet med kurs i Node.js og React.js
- Vi har ordnet skjermer på plassene våre

Hva skal gjøres:

- Fullføre opplæring i NodeJS, Express og React.

Hvilke utfordringer ble møtt:

Uke 6: Opplæring i Node.js og React.js og kravspesifikasjon

Hva ble gjort:

- Gruppemedlemmene har jobbet videre med kurs i Node.js og React.js
- Startet med kravspesifikasjonen
- Jobbet med presentasjon for styringsgruppen som skal holdes 12/02/2020

Hva skal gjøres:

- Fullføre opplæring i NodeJS, Express og React.

Hvilke utfordringer ble møtt:

Uke 7: Presentasjon for styringsgruppe og dokumentasjon

Hva ble gjort:

- Jobbet videre med kurs i Node.js og React.js
- Skrevet en del i sluttrapporten om kravutviklingen og prosessen hittil.
- Arbeidet med presentasjonen for styringsgruppen
- **12/02:** Holdt en vellykket presentasjon for styringsgruppen

Hva skal gjøres:

- Fullføre opplæring i NodeJS, Express og React.
- Utvikle brukerhistorier til bruk i product backlog

Hvilke utfordringer ble møtt:

- Å finne en god struktur for sluttrapporten

Uke 8: Kursing i Spring, Spring Boot, Git, React, testing og Clean Code

Hva ble gjort:

- Gruppemedlemmene har gjennomført kurs hos Accenture hvor det ble kurset: Spring, Spring Boot, Git, React, testing av programvare og Clean Code.
- Tegnet og gjennomgått arkitektur for MongoDB.
- Satt opp repos for backend og frontend.
- Skrevet i sluttrapport.
- Laget plan for sprint 0.
- Laget brukerhistorier.

Hva skal gjøres:

- Gjennomføre sprint 0.
 - Designe endepunkter for API.
 - Rapportskriving, ref: trello.

Hvilke utfordringer ble møtt:

- Utfordrende å lage databasearkitektur ettersom MongoDB(NoSQL) er vesentlig forskjellig fra MySQL.

Uke 9: Sprint 0 - Slutt

Hva ble gjort:

- Laget utkast av databasearkitekturen
- Laget endepunkt for produkter med tilhørende dokumentasjon og integrasjonstester
- Laget et skjelett for frontend delen av applikasjonen
- Skrevet en god del på sluttrapporten

Hva skal gjøres:

- Prioritere backlog til sprint 1
- Ha første sprint planning møte med veileder
- Begynne på testdokumentasjon

Hvilke utfordringer ble møtt:

- Noe sykdom denne uken gjorde at vi mistet litt tid

Uke 10: Sprint 1 - Uke 1

Hva ble gjort:

- Hadde **sprint review** og **sprint planning** i ett og samme møte
- Hadde **sprint retrospektiv** møte med bare gruppen og kom frem til konkrete tiltak for å forbedre arbeidet.
- Lagde endepunkter for kategorier og endret på produktendepunktet slik at det kun er mulig å legge til produkter med gyldig kategori.
- Jobbet med frontend for produkter.
- Skrev en del på sluttrapporten..

Hva skal gjøres:

- Fortsette arbeidet i frontend
- Utvikle endepunkt for brukere og autentisering.

Hvilke utfordringer ble møtt:

- Design av kategori-endepunkt var utfordrende mtp. underkategorier. Det ble tatt et valg om å kun ha ett nivå med underkategorier.

Uke 11: Sprint 1 - uke 2

Hva ble gjort:

- Utviklet endepunkter for brukere og autentisering
- Laget oversikt over inventar til utlån i frontend
- Laget søk og filtreringsfunksjonalitet for inventaret i frontend
- Skrevet i rapporten
- Laget en side for registrering av brukere

Hva skal gjøres:

- Verifisering av email etter registrering
- Lage log in side
- Swagger dokumentasjon av endepunkter for brukere og autentisering
- Legge til detaljer om produkter i front-end

Hvilke utfordringer ble møtt:

- Koronavirus har medført at vi måtte jobbe hjemme fra og med torsdag 12/03. Vi fortsetter å ha daily standup via discord og parprogrammerer via Teamviewer.
- Testing gikk veldig sakte på grunn av treg accenture PC, men nå jobber vi hjemme på kraftigere PCer

Uke 12: Sprint 1 - Slutt

Hva ble gjort:

- Verifisering av email etter registrering
- log in side
- Swagger dokumentasjon av endepunkt for brukere og autentisering
- Skrevet i rapporten
- Laget side for detaljer om produkter i frontend

Hva skal gjøres:

- Prioritere backlog for sprint 2
- Ha digitalt sprint demo møte med veiledere fra Accenture

Hvilke utfordringer ble møtt:

- Samme som forrige uke m.t.p. koronavirus.

Uke 13: Sprint 2 - start

Hva ble gjort:

- Hadde sprint-planning møte med veiledere og sprint retrospektiv møte for sprint 1
- Skrev en del på rapport
- Fikset bugs fra sprint 1
- Begynte på adminfunksjonalitet i frontend
- Startet på endepunkte for utlån i backend

Hva skal gjøres:

- Skrive tester for utlån i backend
- Skrive Swagger-dokumentasjon for utlån i backend

Hvilke utfordringer ble møtt:

- Utfordringer rundt utformingen av endepunktene for utlån grunnet de forskjellige tilstandene et utlån kan være i (utlån forespurt, utlån godkjent, retur forespurt, retur godkjent og tilgjengelig for utlån, retur godkjent men ikke tilgjengelig for utlån)
- Utforming av admin-dashboard og funksjonalitet

Uke 14: Sprint 2 - uke 2

Hva ble gjort:

- Jobbet med API endepunkt for utlånsfunksjonalitet
- Lagde sekvensdiagrammer for utlånsfunksjonalitet
- Lage side i frontend for å legge til og endre på produkter
- Begynte på adminfunksjonalitet for utlån av produkter i frontend
- Fullførte utlånsfunksjonalitet for vanlige brukere i frontend

Hva skal gjøres:

- Lage side i frontend for utlånsfunksjonalitet
- Fullføre endepunkt for utlånsfunksjonalitet
- Lage side for administrering av produkter i frontend

Hvilke utfordringer ble møtt:

- Utfordringer med validering i frontend

Uke 15 og 16: Sprint 2 - uke 3

Hva ble gjort:

- Vi tok påskeferie fra onsdagen uke 14 og forlenger derfor sprintet ut uke 16.
- Jobbet med API endepunkt for utlånsfunksjonalitet
- Lage side i frontend for å legge til og endre på produkter
- Fullførte adminfunksjonalitet for utlån i frontend

Hva skal gjøres:

- Returfunksjonalitet
- Swagger dokumentasjon av utlåns-API
- Bilder av produkter

Hvilke utfordringer ble møtt:

- Utfordringer med validering i frontend
- Utfordringer med å lage nye entitites og produkter

Uke 17: Sprint 3 - uke 1

Hva ble gjort:

- Hadde sprint review med veiledere, sprint retrospektiv møte og planning poker for sprint 3
- Laget funksjonalitet for å bytte passord og brukerinfo
- Skrevet på rapport

Hva skal gjøres:

- Større fokus på rapport
- Rydde opp litt i kode fra tidligere
- Lage en brukertest

Hvilke utfordringer ble møtt:

- Slet med å få mongoDB til å fungere på Droplet VM'en
- Vansker med å implementere serviceMessage funksjonaliteten på tilfredsstillende vis.

Uke 18: Sprint 3 - uke 2

Hva ble gjort:

- Laget adminfunksjonalitet for serviceMessage
- Laget adminfunksjonalitet for administrering av brukere
- Laget funksjonalitet for å sende inn forslag
- Begynt på testdokumentasjon

Hva skal gjøres:

- Lage brukertest
- Deploye appen til droplet server
- Skrive rapport

Hvilke utfordringer ble møtt:

Uke 19 og 20: Sprint 3 - Uke 3 og 4

Hva ble gjort:

- Fullførte testdokumentasjon
- Implementerte sending av mail til admin ved nye låneforespørsler
- Lagde og avholdt en brukertest
- Deployet appen på VM.
- Skrevet og rettskrevet rapport

Hva skal gjøres:

- Fortsette med rapporten
- Levere rapporten
- Begynne med presentasjon

Hvilke utfordringer ble møtt:

- I og med at det var siste sprint, var det vanskelig å definere en klar avslutning på sprintet, da det alltid ville være rom for forbedringer.

Tillegg L

Møtereferater

Innføringsmøte

13.01.20

13:00

Accenture - Fornebu

Til stedet fra
Accenture

Kjell-Olav

Til stede fra
prosjektgruppen

Rany, Eirik, Markus

Referat

Diskusjon

- Adam er syk, burde vært her idag da han er produkteier
- Snakket om oppgaven og hva den vil gå ut på
- Anbefaler oss å bruke scrum
- Vi kommer til å ha møte med veiledere ca. Annenhver uke
- Form-kurs vil bli avholdt som vil hjelpe oss med å komme igang og forme idéer
- Testing er viktig. Det vil bli tilrettelagt for brukertesting
- Vi må bruke en egen innlogging da vi ikke kan få tilgang til Accenture sine systemer
- React vs React-native: Anbefaler oss å utvikle en webapplikasjon i React
- Spillgruppen har utleie av brettspill, kortspill og konsoller
- Litt om dagens situasjon: Microsoft Teams brukes for arrangementer og forespørsler om utleie - Oppleves ustukturert

Veien videre

- FORM-kurs

Til neste gang

Planleggingsmøte

21.01.20	14:30 - 15:45	OsloMet		
Til stedet fra Accenture	Adam, Mira			
Til stede fra prosjektgruppen	Rany, Eirik, Markus			
Referat				
Diskusjon				
<p>Snakket om hva vi har gjort til nå. Gikk gjennom forrapporten og detaljer ved dette. 1 leder og 2 underledere. Det er ca 50 medlemmer i spillgruppa. React fungerer fint til flerside apper også Redux: komplekst. Viktig å sammenligne for å finne alternativer. Trenger vi Redux? Hvordan sende data på tvers av komponenter. Accenture har mye kompetanse. Mobex? Typescript i React. Tsx i stedet for jsx. Anbefales. Er lettere samarbeide og utviklet i. Lager struktur som vi alle følger.</p>				
Veien videre				
<p>Kan bruke Node.js fremfor Springboot. Accenture bruker begge. Scrum: funker med 2 - 3 ukers sprint. DEMO på slutten av hver sprint med veileder. Jira på team nivå. Trello funker bra. Utvikle en kravspesifikasjon. Bruk de viktigste delen av post-it lappene fra FORM kurset. Lag use case diagram for dette. Noter ukentlig.</p>				
Til neste gang	<p>Gruppen fullfører forprosjektrapporten Kravspesifikasjon lages og godkjennes i samråd med produkteier/veileder Gruppen fortsetter opplæring i teknologiene som skal brukes. Accenture kommer tilbake med dato for kurs i Git og React.</p>			

Møte med intern veileder

23.01.20	09:00 - 10:00	OsloMet		
Til stede fra OsloMet	Qian Meng			
Til stede fra prosjektgruppen	Rany, Eirik, Markus			
Referat				
Diskusjon				
<ul style="list-style-type: none">- Ble kjent med veilederen og en annen bachelorgruppe.- Snakket om veilederens forventninger og rolle i prosjektet: Veilederen skal ivareta våres interesser og passe på at vi har progresjon.- Ble enige om å ikke ha ukentlige møter siden gruppen sitter på Fornebu, og dette ville tatt mye tid. Vi møter veileder ved behov og sender ukesrapporter til veileder slik at hun får oversikt over vår progresjon.				
Veien videre				
Lage kravspesifikasjon og dele link av ukesrapporter til veilederen.				
Til neste gang	<ul style="list-style-type: none">- Kravspesifikasjon lages og godkjennes i samråd med produkteier/veileder- Ved neste møte presenterer vi hva vi har gjort hittil og får feedback.			

Møte med ekstern veileder

10.02.20	08:30 - 10:00	OsloMet		
Til stede fra Accenture	Adam, Mira			
Til stede fra prosjektgruppen	Rany, Eirik			
Referat				
Diskusjon				
<ul style="list-style-type: none">- Gikk gjennom presentasjonen og gjorde revideringer etter behov- Gjennomgikk kravspesifikasjon og gjorde endringer/tilpasninger- Gikk gjennom prosjektmål				
Veien videre				
Fullføre kurs, lage arkitektur				
Til neste gang	<ul style="list-style-type: none">- Ha klart brukerhistorier- Diskutere arkitektur			

Sprint 1 - Sprint planning

02.03.20	15:30 -	OsloMet
Til stede fra Accenture	Adam, Mira	
Til stede fra prosjektgruppen	Rany, Eirik, Markus	

Referat

Diskusjon

Dato: spiller liten rolle om det er datoer eller periode. Adam tenker litt på dette og kommer tilbake.

MongoDB: Viser godt fordeler ved MongoDB med nåværende struktur.

Feilmelding: får opp en feilmelding generelle feil (eks feilmelding 500) eller mer spesifikke om vi vil. Ikke pri men viktig aspekt.

Testing av backend: bra tester.

Swagger: god tilbakemelding.

Code coverage (JEST): godt å med i rapporten.

Teste mot endeproduktene: anbefalt.

Appen: eget menyvalg for produkter. Søkefelt (elastic search) for produkter på forsiden/produktsiden.

Loan litt feil: Booking? Borrow my tech (mer Accenture)?

Akseptansestesting: teste kravet til brukerhistorier.

Fargekode for brukerhistorier: Grønt, antar blir ferdig. Blått antar vi at vi ikke blir ferdig med på men begynt på. Rødt er grønt som ikke ble ferdig.

Forvent å bomme på tids estimering.

Søking: forbereding i frontend.

Sprint goal: høres bra ut.

Veien videre

Produktfunksjon virker oppnåelig innen 3 uker. Mye kan bli gjort men mest sannsynlig ikke alt. Det er helt okey. Foreløpig Sprint 1 plan OK!

Til neste gang

- Ta kontakt ved behov.
- Gjennomgå sprinten.
-

Sprint 2 - Sprint review og sprint planning

23.03.20	13:30 - 14:30	Discord/TeamViewer
Til stede fra Accenture	Adam, Mira	
Til stede fra prosjektgruppen	Rany, Eirik, Markus	

Referat

Diskusjon

Viser sprint 1 fremskritt i appen.

Tilbakemeldinger fra demo:

- Bra og enkelt
- Minimalistisk
- Ved feil innlogging, respons tekst

Mongodb tilbakemelding:

- Rbac vs abac -> isAdmin-> ha en array med roller for utvidbarhet.

Nodejs tilbakemelding:

- Middleware: les array

Tilbakemelding sprint suggestion:

- Henteinstrukser og Returinstrukser
- Epost ok, sms ved tid
- Anstaffelse av Epostserver, send mail til Kjell Olav
- God estimasjon av arbeidsmengde

Legge til i backlogg: driftsmelding som kan gå på tvers av alle.

Veien videre

- Ta kontakt ved behov

Til neste gang

- Fullfør MVP
- Fullfør så mye som mulig av sprint backlog i prioritert rekkefølge

Sprint 3 - Sprint 2 review og sprint planning

23.04.20	14:30-15:00	Discord/TeamViewer
Til stede fra Accenture	Adam, Mira	
Til stede fra prosjektgruppen	Rany, Eirik, Markus	

Referat

Diskusjon

Viser sprint 2 fremskritt i appen.

Tilbakemeldinger fra demo:

- Fornøyde med utlånsfunksjonaliteten
- Minimalistisk design

Tilbakemelding sprint suggestion:

- Ikke viktig å legge til venteliste for utlån
- Bilder av produkter er ikke så viktig
- Administratorer må kunne legge til andre administratorer
- Design burde ikke prioriteres før funksjonaliteten er på plass i følge Adam.
Må forhøre oss med skolen om hvor viktig design er i oppgaven.

Annен info:

- Adam tar kontakt med Kjell for presentasjonstrening

Veien videre

- Ta kontakt ved behov

Til neste gang

- Skriv rapport
- Fullfør sprint 4
- Legg til forslagsboks, glemt passord, admin promtering

Avslutning - Sprint 3 review

05/20	14:30-15:00	Discord
Til stede fra Accenture	Adam, Mira	
Til stede fra prosjektgruppen	Rany, Eirik, Markus	

Referat

Diskusjon

Det er normalt at det gjenstår feil i en ferdigutviklet applikasjon. Derfor har man forvaltere.

Forbedringspotensiale:

Farger
(se brukerundersøkelse)

Hva som var bra:

Enkelt å bruke
Intuitivt
Lagde noe som fungerte godt
Trengte ikke bruksanvisning til å bruke (kan bruke dette som argument mot brukermanualer)
Adam skal jobbe for å ta applikasjonen i bruk.

Remote presentasjonskurs for Accenture, Adam skal prate med Kjell om dette på mandag.
Presentasjon for Mira og Adam.

Sensor kan spørre om svakheter i applikasjonen, feks: hva slags tanker vi har gjort oss om UX design.

Veien videre

- Ta kontakt ved behov

Til neste gang

- Skriv rapport