NTNU

Kunnskap for en bedre verden

Bacheloroppgave

IE303612 Bacheloroppgave Data

Kollektivet

Obligatorisk egenerklæring/gruppeerklæring

Kandidatnumre:

10005

10004

Totalt antall sider inkludert forsiden:

Ålesund, Innleveringsdato 30.11.2018

NTNU

Kunnskap for en bedre verden

Obligatorisk egenerklæring/gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

Du/dere fyller ut erklæringen ved å klikke i ruten til høyre for den enkelte del 1-6:		
1.	**Jeg/vi erklærer herved at min/vår besvarelse er mitt/vårt eget arbeid, og at jeg/vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.**	☒
2.	**Jeg/vi erklærer videre at denne besvarelsen:** • ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • ikke refererer til andres arbeid uten at det er oppgitt. • ikke refererer til eget tidligere arbeid uten at det er oppgitt. • har alle referansene oppgitt i litteraturlisten. • ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse	☒
3.	**Jeg/vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§14 og 15.**	☒
4.	**Jeg/vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert i Ephorus, se Retningslinjer for elektronisk innlevering og publisering av studiepoenggivende studentoppgaver**	☒
5.	**Jeg/vi er kjent med at høgskolen vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens studieforskrift §31**	☒
6.	**Jeg/vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider**	☒

Publiseringsavtale

Studiepoeng: 20

Veileder: Mikael Tollefsen

Fullmakt til elektronisk publisering av oppgaven

Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven §2).

Alle oppgaver som fyller kriteriene vil bli registrert og publisert i Brage HiM med forfatter(ne)s godkjennelse.

Oppgaver som er unntatt offentlighet eller båndlagt vil ikke bli publisert.

Jeg/vi gir herved NTNU i Ålesund en vederlagsfri rett til å

gjøre oppgaven tilgjengelig for elektronisk publisering: ☒**ja** ☐**nei**

Er oppgaven båndlagt (konfidensiell)? ☐**ja** ☒**nei**

(Båndleggingsavtale må fylles ut)

- Hvis ja:

Kan oppgaven publiseres når båndleggingsperioden er over? ☐**ja** ☐**nei**

Er oppgaven unntatt offentlighet? ☐**ja** ☒**nei**

(inneholder taushetsbelagt informasjon. Jfr. Offl. §13/Fvl. §13)

Dato: 30.11.2018

Forord

Oppgaven er skrevet av Frank Even Valde og Kristoffer Lie, Dataingeniørstudenter ved NTNU i Ålesund.

Vi har en egendefinert oppgave som går inn i mange av fagene vi har hatt, blant annet Objektorientert programmering, Webteknologi, Mobile og distribuerte applikasjoner, Systemadministrasjon og Datamodellering og databaseapplikasjoner.
Vi har valgt å bruke nye programmeringsspråk og biblioteker fordi vi har et positivt inntrykk av disse og tror de vil fungere bra for oss i denne situasjonen.
Vi vil få bedre kjennskap til Python, JavaScript, databasehåndtering, API og serveroppsett.

Veileder er Mikael Tollefsen.
Takk til Maiken Svinøy for fin logo.

Innhold

Figuroversikt
Sammendrag
Terminologi
Begreper
Biblioteker og rammeverk
Verktøy
Forkortelser
1. Innledning 1
1.1 Bakgrunn 1
1.2 Mål 1
1.3 Problemstilling 2
1.4 Kravspesifikasjon 2
1.5 Rapportens videre innhold 3
2. Teoretisk Grunnlag 4
2.1 Microservices 4
2.2 Frontend 4
2.3 Backend 4
2.4 Agile Metoder/Smidige metoder 4
2.5 Scrum 5
2.6 Planlegging i backlog 5
2.7 Sprint 6
2.8 Scrummøte 6
2.9 Tilbakeblikk 6
2.10 Design 6
2.10.1 Gestaltprinsippene 7
2.11 Python 8
2.12 Objektorientert Programmering (OOP) 9
2.13 pip 9
2.14 Django 9
2.15 Virtual environment i Python 9
2.16 Hypertext Transfer Protocol (HTTP) 10
2.17 HyperText Transfer Protocol Secure (HTTPS) 10

2.18 Representational State Transfer (REST) 10
2.19 Wireframe 11
2.20 Object-relational mapping (ORM) 11
2.21 JavaScript 11
2.22 JavaScript Object Notation (JSON) 12
2.23 Hypertext Markup Language (HTML) 12
2.24 Cascading Style Sheet (CSS) 12
2.25 React 13
2.26 Relasjonsdatabase 13
2.27 Relasjonsmodell 13
2.28 Structured Query Language (SQL) 13
2.29 MySQL 14
2.30 Datamodellering 14
2.31 Secure shell (SSH) 14
2.32 Cloud Server 14
2.33 Apache2 15
2.34 The Web Server Gateway Interface (WSGI) 15
2.35 Cross-Origin Resource Sharing (CORS) 15
2.36 Version Control System (VCS) 15
2.37 GIT 15
2.38 Gitlab 16
2.39 Discord 16
2.40 Wunderlist 16
3. Materialer og Metode 17
3.1 Arbeidsmiljø 17
3.2 Prosjektorganisasjon 17
3.2.1 Prosjektgruppen 17
3.2.2 Oppdragsgiver 17
3.2.3 Veileder 18
3.2.5 Prosjektorganisering 18
3.3 Utviklings metodikk 19
3.3.1 Backlog 19
3.3.2 Sprint 19

3.3.3 Sprint planlegging 19
3.3.4 Scrum 20
3.4 Valg av rammeverk 20
3.4.1 Valg av backend framework 20
3.4.2 Valg av front end bibliotek 20
3.5 Server 21
3.5.1 VPS/Host 21
3.5.2 Backup 22
5.5.3 Programvare 24
3.5.4 Sikkerhet 26
3.5.5 Operativsystem: Ubuntu 16.04.4 x64 27
3.6 Programmeringsspråk 28
3.6.1 Django/Python 28
3.6.2 JavaScript 29
3.6.3 HTML 32
3.6.4 CSS 32
3.7 Utviklerverktøy 33
3.7.1 IntelliJ IDEA 33
3.7.2 Pycharm 33
3.7.3 fluidui.com 33
3.7.4 Postman 33
3.7.5 Distribuering av applikasjon: 34
3.8 Design 35
3.8.1Farger 35
3.8.2 Logo 36
3.8.3 Gestalt 37
3.8.4 Wireframe 38
3.9 Dokumentasjon/kommunikasjon: 46
3.9.1 Discord 46
3.9.2 Google docs 46
3.9.3 Libreoffice 46
3.9.4 Slack 47
3.9.5 Wunderlist 48

4. Resultater 49
4.1 SYSTEMARKITEKTUR 49
4.1.1 Prosjektstruktur 49
4.2 Database 52
4.3 WebApp Design 53
4.3.1 Login/Registrering 53
4.3.2 Bli med Kollektiv / Lag et Kollektiv 55
4.3.3 Hovedside 56
4.4 API 57
4.5 Server 60
4.6 KJENTE FEIL ELLER MANGLER 61
4.6.1 Frontend 61
4.6.2 Manglende implementasjoner 61
5. Drøfting 62
5.1 EVALUERINGER 62
5.1.1 Resultatet 62
5.1.2 Rammeverk og verktøy 65
5.1.2.1 React 65
5.1.3 Prosjektet 66
5.2 VIDEREUTVIKLING AV PROSJEKTET 67
5.2.1 Statistikk 67
5.2.2 Forbedring av Brukergrensesnitt 68
5.2.3 Fullføring av oppgaver 68
5.2.4 Flere Kollektiv 68
5.3 ARBEIDSFORDELING 69
6. Konklusjon 70
7. Referanser 72
7.1 Internett 72
7.2 Litteratur 77
Vedlegg 78
Vedlegg 1: Readme 78
Vedlegg 2: Kildekode 79

Figuroversikt

Figur	Side
Scrum: sprint, figur 1	5
Gestalt, figur 2	7
Gestalt: closure, figur 3	8
Gestalt: Figure and ground, figur 4	8
Wunderlist figur, 5	16
Django URL for innlogging, figur 6	28
Innloggings metode, figur 7	28
React eksempelkode, figur 8	30
React diagram, figur 9	31
Fargekode, figur 10	35
Bilde av innstillinger, figur 11	36
Logo, figur 12	36
Bilde av innlogging mobil, figur 13	37
Wirefram innlogging mobil, figur 14	38
Wirefram innlogging mobil, figur 15	39
Wireframe generisk liste mobil, figur 16	40
Wireframe oppgaveliste mobil, figur 17	41
Wireframe oppretting av ny oppgave mobil, figur 18	42
Wireframe innlogging PC, figur 19	43
Wireframe hovedside PC, figur 20	43
Wireframe registrering PC, figur 21	44

Wireframe Hjemmeside PC (innlogget), figur 22	44
Wireframe oppgaveliste PC, figur 23	45
Wireframe oppretting av oppgave PC, figur 24	45
Wunderlist, figur 25	48
Kodesnutt av oppretting av kollektiv, figur 26	50
Kodesnutt registrering, figur 27	51
Registrering mobil, figur 28	53
Innlogging for mobil, figur 29	53
Innlogging for desktop, figur 30	54
Innloggingsbilde uten kollektiv, figur 31	55
Hovedside PC, figur 32	56
Mobil statistikk, figur 33	56
Mobil innlogging, figur 34	62
Statistikk mobil, figur 35	63

Sammendrag

I Norge bor det i gjennomsnitt 2 personer per hushold (SSB 21.06.2018).
Det gjør at veldig mange må forholde seg til andre mennesker når det kommer til oppgaver og gjøremål i hjemmet. For å hjelpe med denne problemstillingen ville vi utvikle en måte å strukturere og kartlegge prosessen på.
Målet vårt var å lage en applikasjon som fungerte på både mobil og web hvor du kan legge ut hva som trengs å gjøres og en tidsfrist for å fullføre de enkelte oppgavene. For å virke mer givende og for å måle innsatsen til personer som tar i bruk applikasjonen ville vi tildele oppgaver egendefinerte poeng. Poengene vil være synlig for alle som er medlem i gruppen eller bofellesskapet.

Utviklingen av applikasjonen resulterte i en webapplikasjon som kan brukes på alle enheter. Man kan legge til oppgaver og bli med i kollektiv. Prosjektet inneholder en frontend React og Android applikasjon, en backend i Django med REST servicer og database for lagring og manipulering av data.

Prosessen med å skape denne applikasjonen har vært en bratt og lærerik opplevelse, vi har lagt ned stor innsats i å sette opp og utvikle produktet til vår beste evne

Terminologi

Begreper

Agile metoder	Arbeidsmetode hvor en legger vekt på god kommunikasjon og fleksibilitet
Backend	Behandling og oppbevaring av data i bakgrunnen
Backlog	En liste med oppgaver som skal gjøres
Backup	En kopi eller erstatning i tilfelle noe skulle gå galt
Brukergrensesnitt	En interaktiv grafisk fremvisning av programvare
Brukervennlig	Noe som tar kort tid å lære og er intuitivt å bruke
Frontend	Det visuelle en kan se i programvaren
Git	Et versjonskontrollsystem
HTTP	Protokoll for overføring av informasjon over internett
Issue	En oppgave som skal gjøres for programvaren
Loose coupling	At komponenter ikke er avhengig av hverandre og kan operere godt eller helt uten hjelp av andre komponenter
OOP	Programmering hvor en behandler data som objekter
PIP	Pakkebehandler for programmer i Python
React	Bibliotek for JavaScript
Scrum	En smidig metode å arbeide på innen programutvikling
Sprint	En arbeidsiterasjon som ofte varer i 2 uker
Relasjonsdatabase	En database som følger relasjonsmodellen
VPS	Virtuell server
Virtual environment	Virtuelt miljø for programvare å kjøre i
WSGI	Programvare for bruk av applikasjoner på webserver

Biblioteker og rammeverk

Django	Rammeverk som forenkler oppgaven i å sette opp Rest API
React	Bibliotek for JavaScript som behandler data fra backend og gjør nettsiden mer dynamisk

Verktøy

Apache2	Vevtjenerprogram for å kjøre programvare via http
CSS	Et programmeringsspråk for å definere utseende på programmer skrevet i markeringsspråk
Gitlab	Web-basert verktøy for versjonskontroll og «Issue-tracking»
HTML	Markeringsspråk for formatering av nettsider
JavaScript	Et høynivå-programmeringsspråk for å behandle brukergrensesnittet og behandle data fra backend
MySQL	Verktøy som bruker for databaseadministrasjon
IntelliJ	IDE for diverse programmeringsspråk som blant annet JavaScript
Rsync	Programvare for kopiering og backup av filer
Python	Et programmeringsspråk for håndtering av data.
Pycharm	IDE for programmeringsspråket Python
Ubuntu	Operativsystem for server og pc
Virtualenv	Program for å skape virtuelt miljø for Pythonkode

Forkortelser

CSS	Cascading Style Sheets
GUI	Graphical user interface / Brukergrensesnitt
DevOps	Development/Operations
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
I.T	Information Technology / Informasjonsteknologi
JS	JavaScript
JSON	JavaScript Object Notation
ORM	Object-Relational mapping
OOP	Object-oriented programming
OS	Operativsystem
PIP	Pip installs packages
RD	Responsiv design
REST	Respresentational state transfer
SSH	Secure shell
SQL	Structured Query Language

URL	Uniform Resource Locator
VM	Virtual machine / Virtuell maskin
VPS	Virtual private server
WSGI	Web Server Gateway Interface

1. Innledning

1.1 Bakgrunn

1.2

For mange studenter er det å bo i kollektiv noe som kan være en ny opplevelse. Mange har bodd hjemme hos foreldrene sine og hatt mindre ansvar, mens andre har allerede bodd alene og er klar over alt som må gjøres i et hjem. I et kollektiv bli vant til å leve med andre personer og å vise hensyn. Folk har ofte forskjellige oppfatninger av hva som er greit og hva som ikke er det. For eksempel, for noen av oss er søndag en hviledag, er det da greit å støvsuge, vaske, eller skape støy på denne dagen? For enkelte av oss er dette helt greit, for andre er det ikke det.
Andre spørsmål en kan spørre seg selv når en flytter inn i et kollektiv er:

- Hvor ofte må man vaske?
- Hvor grundig må man gjøre det?
- Er det greit å ha en fest på en torsdag?

Noen ganger kan engstelsen for å skape dårlig stemning i kollektivet overgå fristelsen for å klage eller «rette på» de du bor med. Samtidig kan det å irritere seg over oppførselen til samboerne dine skape dårlig stemning uansett, og da kanskje til en verre grad fordi ikke alle partene vet hva som foregår.

1.2 Mål

Effektmål*:* Produktet skal effektivisere og forenkle måten vi tildeler eller strukturerer hverdagslige oppgaver og gjøremål.
Resultatmål: Prosjektet skal øke bevisstheten rundt oppgaver som trenger å fullføres og skal føre til en simplifisert organisering av disse.
Målsetting:

- Forenkle organisering/tildeling av oppgaver

- Lage en Webapplikasjon som er responsiv og brukervennlig
- Godt forståelig design

1.3 Problemstilling

Vår forståelse av bokollektiv og -felleskap er at det er problematisk å holde en struktur og orden på hva som må gjøres, av hvem, og når. Det er derfor viktig at vårt produkt er brukervennlig og lett forståelig. Vår oppgave er derfor å utvikle en webapplikasjon som både er responsiv og faktisk har en effekt på strukturen i et kollektiv.

1.4 Kravspesifikasjon

1.5

Kravspesifikasjonene har vi selv valgt og er basert på erfaring vi har fra å bo i kollektiv. Vi ønsker å samle og digitalisere alle oppgaver og regler som et kollektiv skal trenge i en applikasjon. Med applikasjonen skal alt det administrative i et kollektiv bli overlatt til telefonen.

I applikasjonen skal brukeren kunne gjøre følgende:

- Registrere brukeren
- Logge inn og logge ut
- Opprette gruppe eller kollektiv
- Bli med i en gruppe eller et kollektiv som allerede eksisterer
- Forlate kollektiv
- Endre egne opplysninger som navn, epost og passord
- Se liste over hvilke oppgaver som er ferdig eller ledig
- Opprette oppgaver for kollektivet som inneholder navn, beskrivelse, poengsum og detaljer om oppgaven skal gjentas og hvor ofte.
- Slette oppgaver

- Markere oppgaver som ferdig og få poengene den er verdt
- Se statistikk over hvor mye arbeid som har blitt gjort over forskjellige perioder, og hvem som har gjort det
- Legge til regler for kollektivet
- Slette regler
- Se alle reglene som er i kollektivet
- Stemme over hvilken regel og oppgave som skal være med i kollektivet

I tillegg skal administratorer ha tilgang til en helsesjekk hvor de kan se om alle servicene kjører som de skal.

1.5 Rapportens videre innhold

Kapittel 2 - Teoretisk grunnlag
I kapittel 2 forklarer vi teori som brukes videre i rapporten og prosjektet.
Kapittel 3 - Materialer og metode
Her beskriver vi hvordan vi har brukt metodene, verktøyene og teorien til å bygge produktet vårt.
Kapittel 4 – Resultater
Her viser vi til resultatene vi har kommet frem til.
Kapittel 5 – Drøfting
I kapittel 5 drøfter vi metodene mot teorien og hvordan de henger sammen med resultatene våre.
Kapittel 6 – Konklusjon
Vi ser hvordan resultatet holder opp mot problemstillingen, kravspesifikasjonen og målsettingen.
Kapittel 7 – Referanser
Her viser vi til kildene og referansene våre.

2. Teoretisk Grunnlag

2.1 Microservices

Microservices er en type infrastruktur av en applikasjon som består av flere “loose coupled” tjenester. Med denne type infrastruktur er det mulig å utvikle de forskjellige tjenestene hver for seg uten at det går ut over applikasjonen som en helhet og øker modulariteten.
Flere teams kan utvikle forskjellige tjenester parallelt som gjør det mulig å levere funksjonalitet så snart de er ferdig istedenfor å måtte ha fullført hele prosjektet før det kan bli tatt i bruk.

2.2 Frontend

Frontend er alt som involverer det brukerne ser. Det er også behandling av data på brukerens side (Klientside)

2.3 Backend

Backend eller server-side er i utgangspunktet hvordan nettstedet/applikasjonen fungerer (logikken og databehandling). Dette refererer til alt brukeren ikke kan se i nettleseren/applikasjon som databaser og servere.

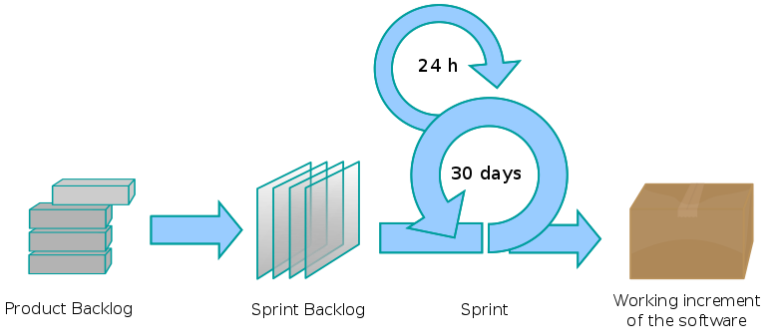
2.4 Agile Metoder/Smidige metoder

Smidige metoder deler utviklingen inn i iterasjoner, der en har et kort tidsrom for å gjøre et antall oppgaver, det skal være en gradvis utvikling av prosjektet som en helhet. Fordeler ved å bruke smidige metoder er at du er tilpasningsdyktig til endringer i oppgaven eller problemer.

2.5 Scrum

Scrum er en metode innenfor smidige metoder.
Med scrum er produktet bygd opp i løpet av flere iterasjoner kalt sprints som gir gruppene en plan for utvikling. Det er viktig å ha god kommunikasjon og samarbeid mellom utviklingsteamet og produkteier for å sikre god kvalitet og at eieren får levert et produkt som oppfyller kravene han/hun har satt.
Korte iterasjoner gjør at det blir lettere å estimere videre iterasjoner og gir et bedre blikk på hvor lang tid helheten tar.
Disse iterasjonene inneholder små oppgaver som blir hentet fra backloggen. I backloggen ligger alle oppgavene som må til for å fullføre prosjektet. Den kan endres på underveis, men er til for å ha et estimat på hvor lang tid det kommer til å ta.
Scrum har daglige møter hvor utviklingsteamet og scrum-master møtes for å diskutere hva som ble gjort siden forrige møte, hva som må gjøres til neste og om det er hindringer som står i veien.
Som vist på figur, består Scrum av flere faser:

- Planlegging i backlog
- Sprint
- Scrum-møte
- Tilbakeblikk



Figur 1

2.6 Planlegging i backlog

Alle oppgavene som må gjøres vil bli lagt i en “product backlog”, teamet jobber sammen for å finne ut hva som må gjøres og hva som trengs for å fullføre prosjektet. Backloggen fungerer som en måte å ha kontroll på alle oppgavene på. Etter hele “produkt backlog” er fylt opp av oppgaver, vil det bli ført over oppgaver til “Sprint backlog” som er oppgavene som skal bli gjort i løpet av en sprint. I starten av et prosjekt vil det være vanskelig å vite hvor lang tid et gitt antall oppgaver tar, så en sprint vil ikke

være fastsatt helt, det vil ta flere sprinter før et bedre estimat blir gitt. For hver sprint vil det være lettere å estimere gitt arbeidsmengde.

2.7 Sprint

I agile metoder er hele arbeidet inndelt i iterasjoner, det kan variere fra en uke til flere uker. Her velges det oppgaver fra backloggen som føres videre til sprint backloggen. Utviklerne velger hva som må gjøres gitte iterasjon og prøver å gjøre dem i løpet av tiden.

2.8 Scrummøte

Hver dag skal det utføres noe som kalles et Scrum-møte. Her skal de fortelle hva de jobber med og eventuelle problemstillinger de møter. Det skal ikke brukes alt for mye tid på disse møtene, ofte er de bare 10-15 minutter lange.

2.9 Tilbakeblikk

Etter å ha fullført en sprint skal det avklares hva som fungerte bra og hva som ikke gikk som det skulle. Det skal reflekteres over og finne ut måter å forbedre fremtidige sprinter.

2.10 Design

For design har vi tatt utgangspunkt i boken *Designing with the Mind in Mind* (Johnson, 2010) hvor de skriver om teorier og prinsipper bak design av brukergrensesnitt. Boken tar for seg temaer som fargevalg og utforming, hvor former og farger utgir uttrykk og assosiasjoner for brukeren. Noe boken legger vekt på er Gestaltprinsippene, som

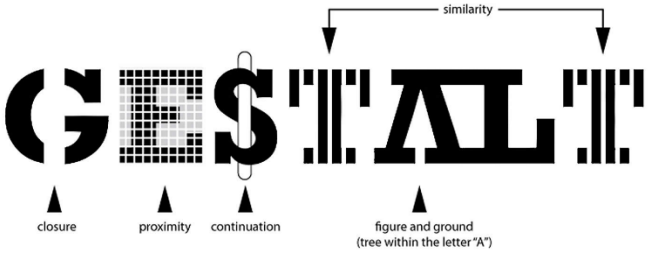
stammer fra persepsjonspsykologiske forklaringer på hvordan vi sanser og organiserer visuelle inntrykk.

2.10.1 Gestaltprinsippene

Gestalt er et begrep fra psykologien som betyr «enhetlig helhet» (Spokane falls 2018). Teoriene går ut på hvordan folk organiserer visuelle elementer i grupper.

Gestaltprinsippene inneholder:

- Similarity
- Continuation
- Closure
- Proximity
- Figure and Ground



Figur 2

2.10.1.1 Similarity

Similarity er engelsk og betyr «likhet». Likhet bruker vi ved å benytte oss av like farger og former for å få en nettside eller applikasjon til å skape et bilde av at dette er en og samme applikasjon.

2.10.1.2 Continuation

Continuation er engelsk og betyr «fortsettelse». Dette skjer når et objekt får øyet til å føre deg til et annet objekt.

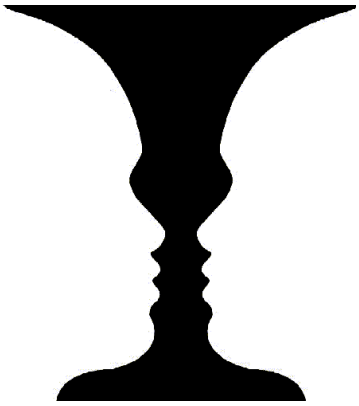
Figur 3

2.10.1.3 Closure

Closure skjer når et objekt ikke er ferdig, men at vi mennesker fyller inn tomrommet for å skape et helt bilde.

2.10.1.4 Proximity

Proximity er engelsk og betyr nærhet. Når vi ser at objekter som er nær hverandre kan vi se på de som en gruppe.



Figur 4

2.10.1.5 Figure and ground

På en pc-skjerm er egentlig alt veldig flatt, derfor bruker vi figurer og bakgrunner som får bilde til å føles ut som det er mer klart eller er foran noe annet. Ofte bruker vi skygger for å demonstrere at noe er i fokus eller over andre objekter.

2.11 Python

Python er et objekt orientert programmeringsspråk, det er kjent for sine simple og lett leselige syntakser. Det er ofte brukt i Rapid Application Development (en utviklingsmetodikk) siden det gir evnen til å bruke dynamisk skriving og binding.

2.12 Objektorientert Programmering (OOP)

Et programmeringsspråk som bruker objekter i motsetning til prosedyrer blir kalt et objektorientert programmeringsspråk. Objektorientering er en måte å strukturere programmer inn i objekter og klasser, hvor objekter er instanser av klasser.
Du kan tenke deg at vi har en klasse for en firkant og da objektet er en instans av en firkant.

2.13 pip

pip er en “package manager” for Python brukt til å installere og bruke bibliotek eller “packages”. Pip har en kommandolinje-interface som gjør det enkelt å installere nødvendige moduler til prosjekter i Python.
Om vi for eksempel skal laste ned en pakke som heter «math», kan vi skrive «pip install math» så ordner den resten for oss.

2.14 Django

Django er et applikasjonsrammeverk skrevet og brukt i Python med åpen kildekode. Det er sammensatte komponenter som hjelper deg å utvikle webapplikasjoner og websider raskere og mer effektivt. Noen av de mest kjente hjelpemidlene til Django er dens rest framework og urlresolver. Rest framework hjelper deg å sette opp støtte til rest api og urlresolver er en måte å binde lenker og metoder.

2.15 Virtual environment i Python

Virtual environment i Python er et verktøy for å isolere det prosjektet du jobber med fra andre eventuelle prosjekter.

2.16 Hypertext Transfer Protocol (HTTP)

HTTP er den mest kjente og brukte protokollen for overføring av informasjon over Internett. Den sender forespørsel eller respons mellom klienter og tjenere. En http-tjener ser på en port og venter til den får en forespørsel (for eksempel POST). Om du sammenligner HTTP med andre protokoller som for eksempel FTP, vil du se at så fort den har gjort det den skal vil den fjerne forbindelsen.

2.17 HyperText Transfer Protocol Secure (HTTPS)

HTTPS brukes for å kryptere all http-trafikk, det er som en sikrere versjon av HTTP. HTTPS bruker en kode på SSL for å sende informasjon frem og tilbake.

2.18 Representational State Transfer (REST)

Rest er en arkitekturstil for å ha en standard mellom maskinsystem over nettet.
Ved Rest er det lettere for systemer å kommunisere. Rest bruker ofte operasjonene tilhørende HTTP.
PUT for å endre tilstanden til eller oppdatere en ressurs.
DELETE for å fjerne en ressurs.
UPDATE for å oppdatere.
POST for å lage en ressurs.
GET for å hente en ressurs.

2.19 Wireframe

Wireframe er en skisse eller et visuelt eksempel som representerer rammeverket for en nettside. Det er for å ha noe å jobbe mot slik vi har en plan for hvordan det skal se ut. Wireframes viser posisjonen til elementer og funksjonaliteten.
Målet med wireframes er å vise funksjonene, plasseringene og regler forbundet med elementene. Det er ikke meningen å følge disse blindt, men heller bruke de som en mal.

2.20 Object-relational mapping (ORM)

ORM er en programmeringsteknikk for å konvertere forskjellige datatyper som ikke er kompatible ved bruk av objektorientert programmering. Dette lager en såkalt virtuell objektdatabase som blir brukt fra utviklingsspråket.

2.21 JavaScript

JavaScript er et programmeringsspråk som brukes til å implementere mer komplekse elementer på et nettsted. Det gjør en nettside håndtert til å gjøre mer enn bare å vise frem statisk data. Med JavaScript er det mulig å gi funksjonalitet til klienten sin side, nettsiden blir mer dynamisk hvor den kan vise oppdaterende informasjon eller metoder (for eksempel animasjon, vise tiden og lignende).

2.22 JavaScript Object Notation (JSON)

JSON er en standard for formatering av meldinger i dataoverføring eller utveksling. For å gi et eksempel på hvordan dette formatet ser ut kan vi bruke et dataobjekt som beskriver en person:

{
“fornavn”: “Geir”,
“etternavn”: “Olsen”,
“alder”: “29”,
“Sivilstatus”: “gift”,
}

JSON er bygges opp som key-value pairs, eller nøkkel-verdi-par. Hver nøkkel (“fornavn” i eksempelet over), har en tilhørende verdi (her: ”Geir”).

2.23 Hypertext Markup Language (HTML)

HTML er det mesteparten av nettsider består av, det er et markeringsspråk. Det definerer betydningen og strukturen av webinnhold. HTML bruker vanligvis CSS til å beskrive en webside sitt utseende og funksjonalitet ved hjelp av JavaScript.

2.24 Cascading Style Sheet (CSS)

CSS er et “stilspråk” som blir brukt for å beskrive hvordan ting skal se ut hos et markeringsspråk. CSS fungerer med å direkte beskrive utseende på et element som for eksempel farge eller størrelse. Det kan også definere forskjellige “id” eller referansepunkter som hvordan en liste skal se ut.

2.25 React

React eller ReactJS er et JavaScript bibliotek for å skape brukergrensesnitt. Det er hovedsakelig utviklet av Facebook samt andre utviklergrupper. React er et verktøy som gjør det enkelt å skape dynamiske nettsider, som gjør at resultatet føles mer ut som en applikasjon enn en statisk nettside ville gjort.

2.26 Relasjonsdatabase

Relasjonsdatabase er en database som følger relasjonsmodellen, den er en samling med tabeller som oppretter relasjoner mellom radene i tabellen.

2.27 Relasjonsmodell

Relasjonsmodellen er en modell med mål å unngå unødvendigheten ved å skrive programmer for å uttrykke databasespørringer.

2.28 Structured Query Language (SQL)

SQL blir brukt for å kommunisere og håndtere relasjonsdatabaser. SQL forespørsler blir brukt for å oppdatere data, hente data, legge til data og slette data.
Et eksempel på bruk: SELECT * FROM table_name;
der du henter all data fra tabellen med gitt navn. Som vist over er språket veldig lesbart og enkelt å forstå.

2.29 MySQL

MySQL er et SQL-basert databaseadministrasjonssystem som har åpen kildekode. Det er basert på en klient-server modell. Kjernen i MySQL er MySQL-server, som håndterer alle databasens instruksjoner (eller kommandoer). Serveren er tilgjengelig som et eget program for bruk i et nettverksmiljø for klient-server og som et bibliotek som kan kobles til applikasjoner.

2.30 Datamodellering

Datamodellering blir brukt for å sette opp datastrukturer.
Med hensyn til relasjonsdatabaser har vi relasjonene mellom entitetene.
Det kan være forskjellige relasjoner, som foreksempel:
Mange-til mange, mange-til-en, en-til-en og en-til-mange.

2.31 Secure shell (SSH)

SSH er en nettverksprotokoll som gir brukere en sikker måte å få tilgang til en datamaskin over et nettverk. All data mellom klient og server blir kryptert i motsetning til over telnet.
Bruksområder går fra eksterne kommandolinje innlogging til eksterne kommandoer, men alle nettverkstjenester kan bruke SSH.

2.32 Cloud Server

En skyserver er en server som er bygget og levert via en skytjeneste over nettet.
Den oppfører seg og fungerer på samme måte som en vanlig server bare at man får tilgang til den over internett.

2.33 Apache2

Apache er den mest brukte webserver program med åpen kildekode.
Det store flertallet av Apache Servere kjører på Linux-distribusjon, men kjøres også på Windows og Unix systemer.

2.34 The Web Server Gateway Interface (WSGI)

WSGI er en metode for webservere å kommunisere med webapplikasjoner eller rammeverk som er skrevet i Python.

2.35 Cross-Origin Resource Sharing (CORS)

CORS er en mekanisme som tillater begrensede ressurser på en nettside å bli forespurt fra et annet domene, utenfor domenet hvor det ble forespurt.
For eksempel på et cross-origin request kan det være en frontend kode som spør en applikasjon fra http://eksempel.com om en httpRequest fra http://min.api.com/informasjon.json
De har, som man kan se her, ikke samme "origin" eller domene.

2.36 Version Control System (VCS)

VCS er styringen eller kontrollen på endringer i filer som dokumenter, dataprogram, nettsider og lignende.

2.37 GIT

Git er et versjonskontrollsystem for å kartlegge forandringer i datafiler (ofte/mest brukt i programvareutvikling).

2.38 Gitlab

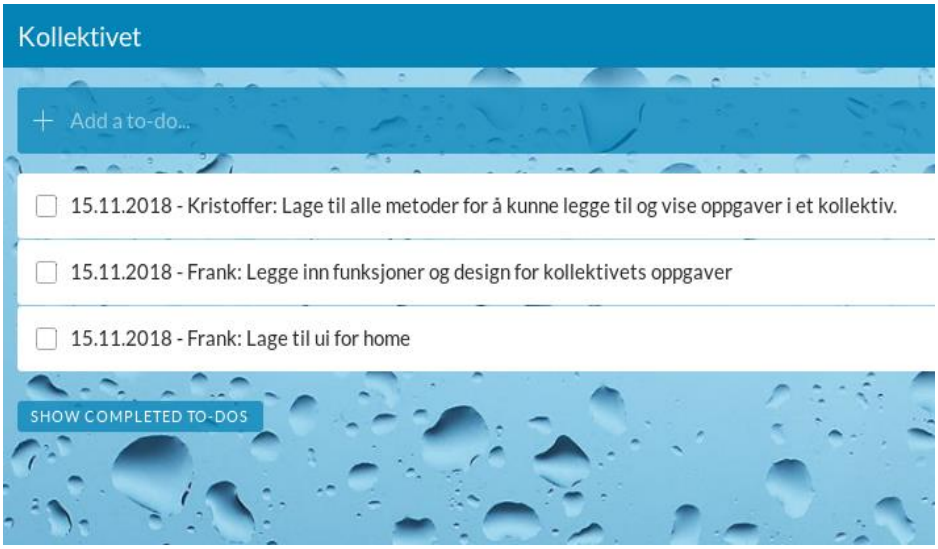
GitLab er en applikasjon for hele DevOps livssyklusen. Det er en web-basert Git-versjonsmanager hvor du har tjenester som wiki og “issue-tracking”.

2.39 Discord

Discord er en kommunikasjonstjeneste over nettet. Du kan sende bilde, tekst, video, og lyd kommunikasjon mellom brukere.

2.40 Wunderlist

Wunderlist er et skybasert administrasjonsprogram. Den lar brukerne administrere sine oppgaver fra en smarttelefon, datamaskin og andre smart-enheter.



Figur 5

3. Materialer og Metode

3.1 Arbeidsmiljø

Ettersom vi bor et godt stykke unna hverandre og det ikke alltid er nødvendig at vi sitter i samme rom, har vi i stor grad brukt Discord som kommunikasjonstjeneste. Her har vi delt bilder, kode og filer, samt kommunisert via “voice-chat”.
Jobbing hjemmefra kunne blitt distraherende, men når vi har avtalte tidspunkt og begge arbeider jamt sammen var det ikke noe problem å konsentrere seg.
Det kan være flere fordeler ved å arbeide hjemmefra, det er lettvint å lage seg et måltid, vi var ikke bundet opp av transport og det ble ingen dødtid.
Vi har begge stasjonær-datamaskin som jobber raskere og har flere skjermer disponibelt enn om vi skulle tatt med laptop og møttes.

3.2 Prosjektorganisasjon

3.2.1 Prosjektgruppen

Gruppen for dette prosjektet består av studenter med kandidatnummer

- 10005
- 10004

Begge medlemmene er studenter ved studiet Bachelor i ingeniørfag - data, ved NTNU Ålesund.

3.2.2 Oppdragsgiver

Siden dette prosjektet er egendefinert virker Mikael Tollefsen som oppdragsgiver. Oppgaven er definert ved at vi skal lage en webapplikasjon som lar brukere planlegge oppgaver og plikter som må gjøres i et bofelleskap. Det ble utviklet en ide fra erfaringer med å bo i et slikt felleskap og problemer som kan oppstå i en slik situasjon.

3.2.3 Veileder

Veileder for prosjektet var Mikael Tollefsen. Han er universitetslektor ved Institutt for IKT og realfag host NTNU i Ålesund (2018). Vi fastsatte alltid å møte med bortimot to ukers intervaller og diskuterte hvilke verktøy vi hadde planer om å ta i bruk.

3.2.4 Rådgivning

Hovedsakelig har alle råd vi har fått kommet fra vår veileder, men det er gitte problem vi har fått råd fra erfarne utviklere som tidligere har brukt både Python og Django.

3.2.5 Prosjektorganisering

Grunnet gruppestørrelsen på to personer har organiseringen blitt at begge to jobber med alt, vi har begge utviklet og passet på at begge skriver rundt det som er relevant for arbeidet.

3.3 Utviklings metodikk

3.3.1 Backlog

Backloggen vår er oppfylt med issues vi på forhånd ble enige om trengtes og var nødvendige for utviklingen av prosjektet. En issue beskriver hva som skal implementeres men ikke hvordan det skal gjøres. Et eksempel på et issue:
“#9 - Knytt sammen frontend og backend for registrering”.
Før vi startet på eksempelvis denne issuen var tidligere krav fullført, backend hadde en metode for registrering og frontend var klar til å ta imot den via api.
For hver sprint fant vi ut flere nødvendigheter og disse blir da lagt inn i backloggen før de blir ført over til en potensiell sprint. Oppgavene som da ligger sortert på Gitlab har statusen “åpen”, “Må gjøres”, “i arbeid” og “ferdig/merged”. Når de ligger “åpen” er det mulighet å lage et kall for å automatisk opprette en gren å arbeide. Grenen blir lukket og slått sammen når den av oss som ikke arbeidet med grenen har sett over og godkjent.

3.3.2 Sprint

Sprinten eller iterasjonene våre har en lengde på 2 uker omtrent, det har hendt at vi har utvidet noen iterasjoner med 1-2 dager.

3.3.3 Sprint planlegging

Etter hver sprint satt vi oss ned og pratet om hva som trengtes til neste iterasjon og om det var noe vi ikke ble ferdig med la vi inn dette i neste. Vi tok også råd fra veileder om hva vi kunne legge til iterasjonen.

3.3.4 Scrum

Vår form for scrum ble gjort hver dag vi startet å arbeide med prosjektet, vi har en dialog hele veien. Siden vi bare var to medlemmer av prosjektet hadde begge parter god forståelse ovenfor problem som oppstår og hva som måtte gjøres.

3.4 Valg av rammeverk

For å oppnå vår kravspesifikasjon måtte vi se på potensielle rammeverk som støttet våre krav. Vi forhørte oss med bekjente i industrien og undersøkte videre på disse.

3.4.1 Valg av backend framework

Når det kom til backend rammeverk fikk vi anbefalt Django og Python for funksjonalitet, vi endte med å søke opp eksempler og artikler rundt Django.

Django viste seg å håndtere restkall veldig effektivt, det var innebygde metoder som støttet å knytt metoder til api endepunkt veldig simpelt. I Django brukes ORM for databasehåndtering, det gjorde at vi satte opp modeller som ble generert i databasen og kunne manipulere virtuelle instanser som representerte data på en enkel måte.

3.4.2 Valg av front end bibliotek

For valg av et frontend rammeverk eller bibliotek såg vi på de mest brukte typene, det stod mellom Angular, React og Vue. Når vi søkte opp forskjellene og brukervennlighet virket det som Vue var det simpleste, men med minst eksterne biblioteker og støtte. Angular og React derimot har store selskap bak seg og veldig stor støtte og miljø bak seg. Både React og Angular kunne utføre det vi trengte.

Videre undersøking viste at valget stod mest på preferanser og meninger. Vi endte med å velge React etter å ha konkludere med at det ikke var store forskjeller for vårt

prosjekt. Vi merket også at det var større aktivitet rundt React noe vi fant betryggende for fremtidig støtte. (Jonas Bandi 2018)

3.5 Server

3.5.1 VPS/Host

For web hosting har vi valgt å bruke Digital Ocean. Ved å bruke Digital Ocean har vi mulighet til å endre på størrelse eller hastighet med et tastetrykk og forvente få minutter nedetid. Nedetiden varierer i forhold til hvor mye plass som allerede er i bruk på serveren. I vårt tilfelle vil det ta ca 2 minutter for å endre størrelse (Digital Ocean 2018). Om vi skulle unngå 2 minutter nedetid kan vi sette opp en ny server og til slutt flytte domene navnet over til den nye serveren med mer plass og høyere hastighet. Dette er noe vi kan gjøre hvis vi skulle ha flere aktive brukere.

3.5.1.1 Spesifikasjoner

RAM: 1 GB
Harddisk: 25 GB
Operativsystem: Ubuntu 16.04.4 x64
Offentlig Ipv4: 104.248.44.55
Host: Digital Ocean

3.5.1.2 Virtualisering

Hos Digital Ocean betaler vi 5$ (rundt 42.26,- kr) per måned. Serveren er virtuell, som for oss betyr ingenting annet enn at det er billigere å leie. Om alle skulle leie en fysisk server ville det blitt mye dyrere på grunnlag av at fysiske maskiner bruker plass. Eksempel: Om en kan ha fem separate servere, så er det bedre å kjøpe en server som kan virtualisere 5 servere, enn å kjøpe 5 separate servere. Dette er ikke bare bra for økonomien vår, men det er også bra for miljøet om vi tenker på hva som må til for å lage fem middelmådige maskiner istedet for 1 god maskin.
Det som kunne vært et alternativ er å kjøre alle applikasjonene som vi skal ha på en maskin uten virtualisering. Den negative siden med dette er at vi ikke får separert oppgavene som vi har kjørende. Hvis noe feil skjer eller maskinen trenger en omstart så slipper en at alle de andre servicene blir påvirket av dette. Dette skaper mindre nedetid, og gjør det enklere å separere prosjektene og servicene vi har på serveren.

3.5.2 Backup

Redundans er noe av det viktigste i I.T verden. Oppetid er viktig for at brukerne skal få et godt inntrykk. En tjeneste som har høy nedetid vil ikke være fristende for enten kunder eller brukere. Redundans betyr å ha mer enn en. I I.T verden er det viktig at vi alltid har ekstra i tilfelle noe skulle skje. Når det generelt gjelder data så kan alt bli ubrukelig om en ting slutter å fungere. For eksempel om nettverket slutter å fungere i en bedrift og mye av programvaren de bruker er web basert, så vil de stå igjen med ingenting. Derfor trenger de en ekstra tilkobling til internett om de skal ha høyest mulig oppetid.
Det er veldig viktig for oss at dataene til brukerne ikke forsvinner. Vi mister raskt brukere om alt blir slettet. Hvis vi tenker på målet med applikasjonen så er det å holde styr på arbeidet som har blitt gjort i et kollektiv, om alt blir slettet så vil alle brukerne komme tilbake med ingenting og må starte alt på nytt.
Backup er en viktig redundans, og for applikasjonen «kollektivet» har vi flere metoder for backup.

3.5.2.1 Rsync

Rsync er ikke noe vi har tatt i bruk fordi vi ikke har penger til å sette opp server som backup. Men her er ideen om vi skulle ha brukt det.
Rsync er en programvare for Linux som synkroniserer filer over til en annen lokasjon som kan være til en annen mappe på samme maskin, eller til en annen server via ssh. Vi kopierer over prosjektene våre til en annen server for å ha programvaren vår på 2 lokasjoner, slik at vi enkelt kan hente filer og database informasjon om noe uventet skulle hende med serveren.
Rsync kjører automatisk en gang i timen via «cron» som er script som kjøres automatisk en gang i timen. Rsync kjører bare inkrementell backup, noe som betyr at vi ikke tar full backup og bare får med oss endringene som har blitt gjort.
Vi legger også inn dato på mappene som blir tatt backup av slik at om vi skulle ønske å gå en uke tilbake så har vi også muligheten til dette.
For ekstra sikkerhet så har vi satt opp backup serveren hos noen andre enn Digital Ocean slik at vi fortsatt har tilgang til dette om Digital ocean mister dataene sine. Nå har Digital Ocean en egen versjon av backup som vi kan betale for å bruke, men mister vi Digital Ocean så mister vi også denne. Med mye penger og mest mulig oppetid som mål så kunne vi brukt begge.

3.5.2.2 Versjonskontroll som backup

En av de tingene vi bruker gitlab til er versjonskontroll. Versjonskontroll har vi for å kunne se tilbake på hvilke endringer som har blitt gjort. Dette kan være fint i situasjoner hvor noe har sluttet å fungere slik at vi kan hoppe tilbake en versjon for å slippe nedetid. Vi kan også hente ned siste versjonen av programvaren vi har laget, og vi kan derfor bruke dette som en backup til programvaren vår. Vi vil ikke få database filene som gjør at vi ikke kan bruke dette som en fullstendig backup.

3.5.2.3 Server redundans

Hvis vi skulle være helt sikker på at tjenesten vår har 100% oppetid burde vi leie to servere fra 2 forskjellige tjenere med synkroniserende databaser. Vi kan enkelt legge inn en sjekk i koden som ser om vi får http respons fra serveren. Om denne feiler så skifter vi hvilken server vi skal bruke.
Med to servere som opereres samtidig, og som har tilgang til samme informasjon, så er vi ganske sikker på at vi har høg oppetid.

5.5.3 Programvare

På serveren er det flere programvarer vi bruker for å kjøre applikasjonen vår. For å kjøre applikasjonen må vi ha en http server tjeneste for å holde oppe nettsiden og gi respons, database for å lagre informasjonen til brukerne, backup i tilfelle noe får galt og vi mister data, programvare som kan behandle data, og diverse metoder for å gjøre serveren sikker.

3.5.3.1 Apache2

Vi bruker Apache2 for å kjøre webapplikasjonen vår. Her kjører både React prosjektet som er det visuelle vi kan se, og Django prosjektene som er det som behandler data som skal inn og ut fra databasen.
Apache2 tar all styring for hvilke porter som blir brukt av hvilken service. Den styrer også brukerne automatisk over til https i stedet for http. De siste årene har vi sett at kryptering overalt er viktig, det er også nødvendig i vår situasjon hvor vi skal både håndtere privat informasjon og passord.
For å la apache2 kjøre frontend så tar vi bare filene som React prosjektet produserer, og legger de inn i en mappe som apache2 har gjort tilgjengelig for hvem som helst å komme seg inn på, slik at nettleseren kan til brukerne kan lese filene og vise fram nettsiden.

Apache2 bruker WSGI for å gjøre backend tilgjengelig for brukeren. Prosjektet er skrevet i python og vil derfor ikke ha noen filer som apache2 forstår uten at vi bruker WSGI.
Hver applikasjon får hver sin port og konfigurasjonsfil slik at vi kan separere de og legge til spesifikke konfigurasjoner til hver microservice og

3.5.3.2 WSGI/Django

Django prosjektet blir lagt inn i en mappe på serveren der den blir kjørt av en konfigurasjonsfil i Apache2 som bruker konfigurasjonen som ligger i WSGI filen i prosjektet. I WSGI filen står det hva som trengs for å kjøre prosjektet, som inkluderer virtuelt miljø og hvilen Python versjon som skal brukes.

3.5.3.3 Mysql/Database

Databasen er hvor vi lagrer all informasjon som brukeren gir oss. Her lagrer vi informasjon som epost, navn, passord, kollektiv informasjon, oppgave informasjon og alt annet vi skulle trenge for å gjøre det applikasjonen skal gjøre.
Vi har valgt å bruke MySQL fordi vi vet at dette er noe som fungerer godt og har brukt det tidligere til andre prosjekt. Vi har søkt litt og ser at det er andre programvarer vi kunne gjort jobben like bra. Et alternativ som vi ser begynner å bli populært er Postgresql. Vi ser på internett at det er ekstremt mange meninger og at begge har positive og negative sider ovenfor hverandre, men siden vi vet at MySQL klarer jobben, så gikk for den.
Når vi kommuniserer med databasen så gjør vi det bare gjennom backend med Django. Her bruker ORM (Object-relational mapping) slik at vi slipper å skrive SQL selv, noe som gjør arbeidet vårt mye enklere.

3.5.4 Sikkerhet

Det viktigste når det gjelder sikkerhet er å holde programvaren som vi bruker oppdatert, sikkerhetshull blir funnet hele tiden i kjente programvarer som kan bli brukt til å stjele informasjon. Vi har automatisk oppdateringer på serveren vår for å unngå sikkerhetshull. Når oppdateringer blir kjørt oppdaterer den både operativsystemet og andre programvarer som apache2 og MySQL.
For å få tilgang til serveren må en koble seg på via ssh (Secure shell), en må også ha en kryptert nøkkel og et passord for å komme seg inn.

3.5.4.1 HTTPS

Vi bruker https slik at all trafikk gjennom web er krypter. Dette er spesielt viktig for behandling av passord. Apache2 vil automatisk sende brukere til https slik at det ikke er mulig å bruke http.

3.5.4.2 Cors

Cors blir brukt av oss for å hindre cross-site script. Vi bruker dette i Django for å unngå at ukjente prøver å kjøre restmetodene vi har tilgjengelig. Her har vi en hviteliste hvor vi legger inn domenet vårt, slik at kun applikasjonen vår har tilgang til backend metodene. Om vi en dag trenger å gi tilgang til noen andre som ønsker å bruke våre metoder, så er det mulig å gi tilgang ved å legge til domener i hvitelisten.

3.5.5 Operativsystem: Ubuntu 16.04.4 x64

For operativsystem så har vi valgt å bruke Ubuntu 16.04 LTS (Long Time Support). Dette operativsystemet er gratis og har support og oppdateringer helt frem til 2021. Grunnen til vi velger et Linux basert operativsystem er fordi det er gratis og det er veldig mye brukt for web hosting. Et alternativ ville vært Windows, et operativsystem som også er populært for web hosting. I en undersøkelse gjort av Montis (Montis 2017) ser vi at det ikke er store forskjeller i oppetid eller hastighet for respons.
Det er mange operativsystemer vi kunne valgt av, vi ser raskt at det er mange som kan gjøre jobben som trengs. Vi har derfor valgt Ubuntu fordi det er gratis, og fordi det er et operativsystem vi er kjent med fra før og ønsker derfor å bruke den kunnskapen vi allerede har rundt dette.
Ubuntu veldig lite fra oss (Ubuntu 13.11.2018):

- 300 MHz x86 prosessor
- 268 MB Ram
- 1,5 GB lagringsplass
- Grafikkort og skjerm som håndterer 640x480

3.6 Programmeringsspråk

3.6.1 Django/Python

Django er brukt i hele backend som rammeverk for applikasjonen, her setter vi opp hele databasen ved hjelp av modellering i Python. Vi definerer alle tabellene og hvilken datastruktur de forskjellige tabellene har. Django er bygget for å forenkle oppsetting av webapplikasjoner som gjorde at backend ble effektivt bygget opp.

Ved å definere en metode og deretter linke den til en url har du allerede laget et endepunkt for api. Et eksempel for login:

```
urlpatterns = [

    path('user/login/', login),
```

Figur 6

Her binder vi metoden login til user/login.

Metoden den binder:

```
@csrf_exempt
@api_view(["POST"])
@permission_classes((AllowAny,))
def login(request):
    email = request.data.get("email")
    password = request.data.get("password")
    print(email)
    print(password)

    if email is None or password is None:
        return Response({'error': 'Please provide both email and password'},
                        status=HTTP_400_BAD_REQUEST)
    user = authenticate(email=email, password=password)
    if not user:
        return Response({'error': 'Invalid Credentials'},
                        status=HTTP_404_NOT_FOUND)
    if user.activated == False:
        return Response({'error': 'Activate your user via email'},
                        status=HTTP_404_NOT_FOUND)
    token, _ = Token.objects.get_or_create(user=user)
    return Response({'token': token.key, 'id': token.user_id, 'name':user.name,'email':user.email},
                    status=HTTP_200_OK)
```

Figur 7

3.6.1.1 Pip

Pip ble brukt til å installere eksterne bibliotek som er tilgjengelig i Python og Django. Ved å bruke en kommandolinje kan vi laste ned og installere alt vi trengte.
Et eksempel på bruk: “pip install django-cors-headers”

3.6.1.2 ORM

I Django bruker vi ORM for å gjøre endringer eller kall til databasen. Et typisk kall kan være:
Kollektiv.objects.get(kollektivid=X)
hvor vi henter en instans av et kollektiv med en spesifisert id eller navn.

3.6.1.3 Eksterne biblioteker

django.contrib.auth importeres for håndtering av modellene, vi tar inspirasjon fra den eksisterende User modellen i Django, men har gjort endringer for vårt bruk. Ved å importere denne har vi muligheten å kalle på allerede definerte metoder som create_user(paramter), men vi måtte modifisere denne og alle andre metodene vi tok i bruk. HttpResponseRedirect blir importert til django for håndtering av http statuskoder, det gjør det mulig å sende de via Response til frontend. rest_framework importeres for bruk av rest, den gjør det mulig å sammenslå lenker med metoder, bruk av ORM, autentisering med 0auth og bruk av serialization.

3.6.2 JavaScript

Vi brukte JavaScript for å håndtere all informasjon fra backend, og for å få en dynamisk og rask applikasjon. For å bruke kortere tid brukte vi en bibliotek som heter React.

3.6.2.1 React

React brukte vi for hele frontend. Når React prosjektet bli kjørt blir det eksportert html, css og JavaScript så den er klar for å bli lest av nettleseren. For mesteparten av designet er det kun html og css som styrer, mens det dynamiske er kontrollert av JavaScript.

Når vi sier det dynamiske så mener vi at applikasjonen ikke kal være lik for alle og skal endre seg etter behov. I vårt tilfelle leverer vi forskjellige nettsider baser på om:

- Brukeren er logget inn
- Brukeren er med i et kollektiv
- Brukeren er på telefon, pc eller nettbrett
- Hvilket kollektiv er brukeren med i
- Hvor mange poeng hver bruker i kollektivet har

For å sjekke opp informasjon om brukeren brukte vi som oftest en funksjon som heter «Fetch», som er en metode for å kalle på Rest API som returnerer informasjon eller en feilmelding om noe ikke gikk som det skulle. Med denne informasjonen kan vi forme applikasjonen for hver enkelt bruker. Med andre ord: React manipulerer applikasjonen ut i fra hvilken informasjon som er gitt.

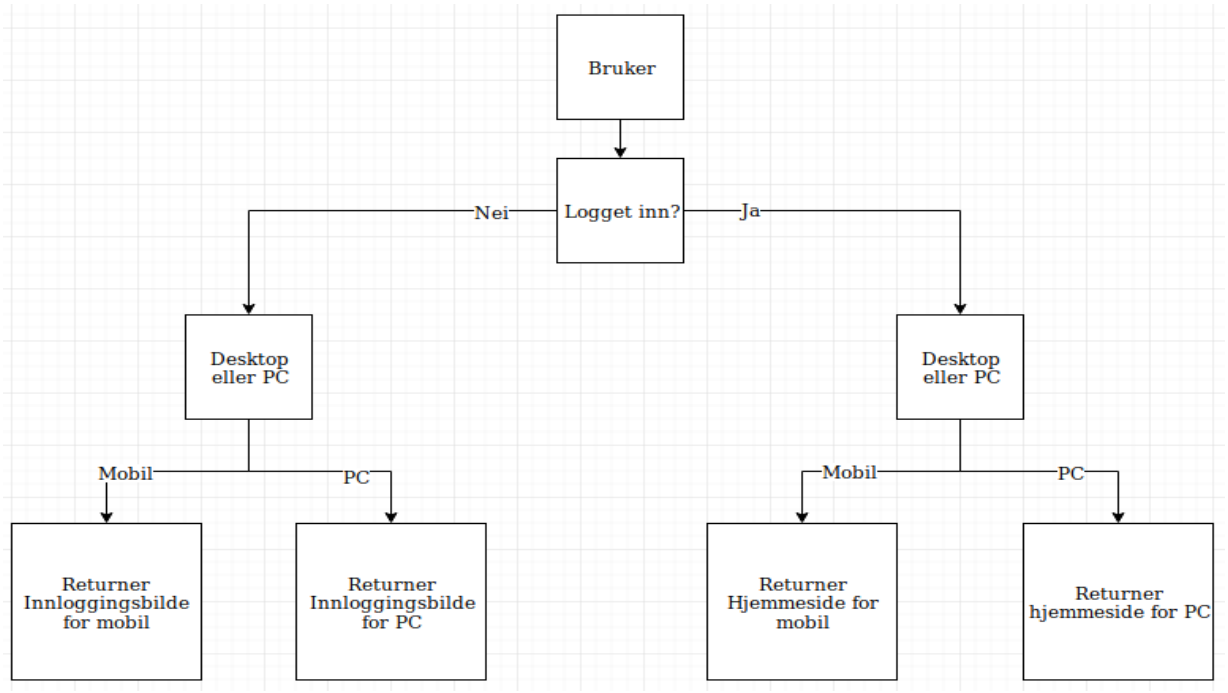
```
class App extends Component {

    render() {

        //Checks if we are logged in and then returns the home page.
        if(AuthService.loggedIn()) {
            if (KollektivService.isMemberInKollektiv) {
                return (
                    <HomePage/>
                );
            }
            else {
                return (
                    <NotInKollektivPage/>
                );
            }
```

Figur 8

React deler opp de forskjellige komponenter som gjør det lett å holde kontroll og vite hvor diverse kode skal være. Som vi kan se i figur 7.

I koden som vises i figur 7 sjekker React først om brukeren er logget inn, og hvis den er logget inn vil den dermed sjekke om den er i et kollektiv for å sende den til riktig komponent. Med komponenter kan alt rundt som bakgrunn eller navigasjonsmeny være likt hele tiden, mens komponenten i midten endrer seg etter hvilken tilstand brukeren er i.

Mindre komponenter betyr mer gjenbruk. Vi ser at det er mye vi har brukt flere ganger både på mobil og PC versjonen. Med dette halverer vi arbeidsmengden vår og får tid til å utvide applikasjonen på andre områder.



Figur 9

En god egenskap React har er at den lagrer alt i minne og velger å bare oppdatere det som endrer seg, og ikke hele bildet hver gang. Å minimalisere endringene som enheter må håndterer sparer både brukeren og serveren fra å gjøre mye arbeid om igjen, dette resulterer i at applikasjonen blir raskere og vi kan skape mer kompleksitet før vi får problemer med ytelse.

3.6.2.2 Eksterne biblioteker

3.6.2.2.1 react-device-detect

URL: https://www.npmjs.com/package/react-device-detect

Kildekode: https://github.com/duskload/react-device-detect

React-device-detect er et bibliotek for å automatisk legge merke til om enheten som bruker applikasjonen er mobil eller pc. Her leser den «headers» og vi kan selv velge hva en mobil eller en pc skal få ut av applikasjonen.

3.6.3 HTML

Markeringsspråket HTML ble skrevet inn i React prosjektet i metoder «render». Forskjellige deler av HTML ble spredt blant komponenter slik at vi kan gjenbruke koden.

3.6.4 CSS

Alt av statisk design ble laget i CSS, det inkluderer:

- Farger
- Plassering
- bakgrunner
- Egenskaper for tekst og bilde
- Form og størrelse på komponenter
- Skygger

3.7 Utviklerverktøy

3.7.1 IntelliJ IDEA

Vi har brukt IntelliJ for å utvikle frontend i språket reactJS. Intellij er hovedsakelig en IDE for java, men med “ultimate edition” hadde vi støtte for javascript, noe vi har mulighet for å bruke med studentlisens.
Valget av IDE er basert på tidligere erfaring, og det er mange andre programvarer som kunne gjort samme jobben som intelliJ. Uten studentlisens hadde vi ikke brukt dette.

3.7.2 Pycharm

For Backend har vi brukt Pycharm for programmering av Django. Her har vi brukt Professional edition med studentlisens. Med vår bruk hadde det gått fint å bruke gratisversjonen av Pycharm,

3.7.3 fluidui.com

Fluidui er et nettsted for oppretting av wireframes, her kan du også redigere sammen med andre i sanntid, det var dette vi brukte for våre wireframes.

3.7.4 Postman

Postman er et verktøy for bruk ved API utvikling, det gjør det enklere å teste, utvikle og dokumentere API ved å putte inn adressen og de parametrene som hører til. Alle API kall blir lagret for rask tilgang til senere anledning slik brukeren ikke trenger å begynne på nytt. Vi tok godt nytte av Postman ved testing av våre mange API-endepunkt og testet alltid at de fungerte som de skulle før vi pushet til master.

3.7.5 Distribuering av applikasjon:

Applikasjonen vår er alltid tilgjengelig på web, som betyr at brukeren ikke trenger å installere noe annet enn en nettleser for å kunne bruke den.

Vi har også tilgjengelig et «webview» for android, og i fremtiden et for IOS også, hvor en får et ikon på mobilen eller nettbrettet sitt og kan bruke applikasjonen ved å trykke på denne. Det er flere grunner til å bruke et «webview» som:

- Gjøre applikasjonen mer tilgjengelig
- Applikasjonen vil alltid være i fullskjerm
- Applikasjonen husker brukernavn og passord til og med etter å ha slettet informasjonskapsler fra nettleseren din
- Tilgang til push-varsler på mobilen

Siden applikasjonen bare er på web, så kan vi enkelt legge inn oppdateringer uten nedetid. Alt vi trenger å gjøre er å legge inn de oppdaterte filene i mappene som http serveren vår bruker så vil applikasjonen automatisk oppdatere seg hos brukeren. Både React og Django oppdaterer seg med en gang det blir endringer i filene, som gjør at vi unngår nedetid.

3.8 Design

Når det gjelder design så er det viktig med et enkelt design. Brukere i dag ønsker minst mulig knapper og mindre valg for å minke kompleksiteten. Det enkle ofte det beste, derfor ønsket vi å bruke mest mulig ikoner og bilder for å vise hvordan applikasjonen fungerer. Hvis brukeren må lese mye tekst eller lete etter hvordan applikasjonen fungerer så har vi et dårlig design.

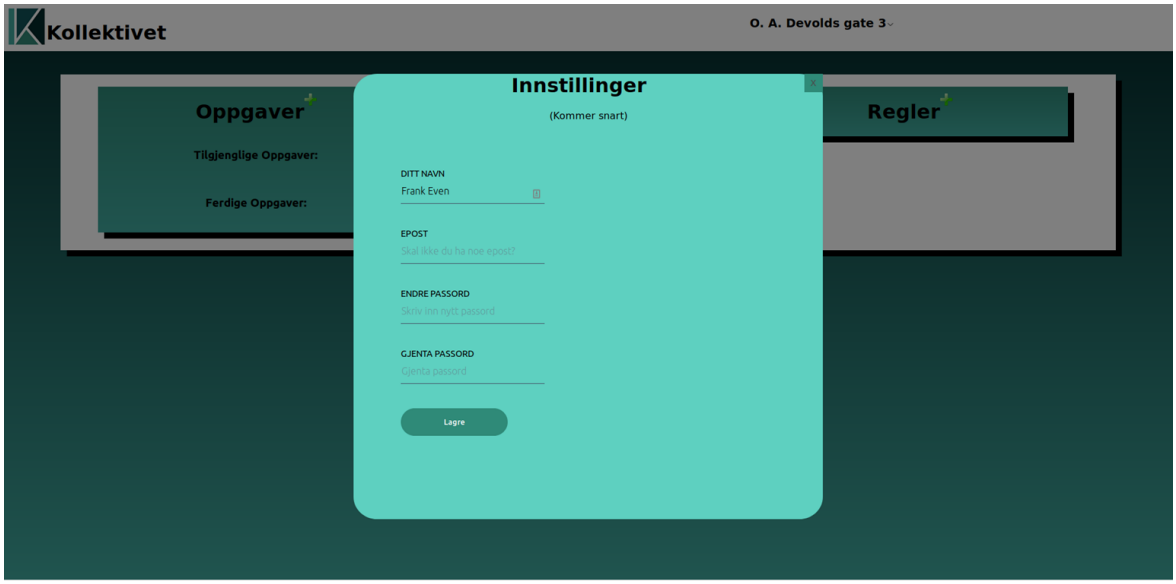
3.8.1Farger

Vi har bestemt oss for en grønne farger:

Figur 10

og vi vil bruke disse fargene gjennom utviklingen av applikasjonen. Det er viktig å holde farge og design valg konsekvent for å få applikasjonen til å føles ut som en applikasjon. Vi valgte fargen grønn som vår hovedfarge i applikasjonen. Fargen grønn er en frisk og positiv farge som vi assosierer med at ting fungerer.

Fargevalg er viktig for å sende melding. Ting med svart tekst blir ofte ignorert. Det er derfor en feilmelding som oftest er rød slik at en skal legge merke til den. Grønn er et



Figur 11

tegn på at alt er i orden, og det er en melding vi ønsker å sende ut for de som bruker applikasjonen.

Fargene kan en finne igjen i logoen.

3.8.2 Logo

Logoen er delt opp i alle de fargene vi har valgt å bruke og har bokstaven “k” som står for “kollektivet”. Denne logoen bruker vi igjen både som ikon på web og mobil. Logoen er laget av tidligere designer for Ålesund Studentsamfunn Maiken Svinøy.

KOLLEKTIVET

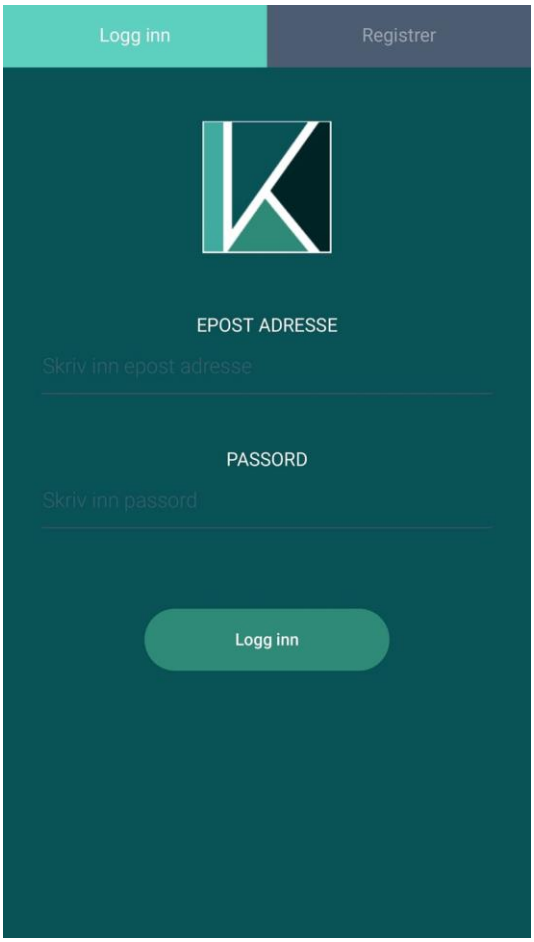
Figur 12

3.8.3 Gestalt

Når vi jobber med design var det veldig viktig for oss å bruke gestaltprinsippet «Similarity» eller «Likhet». Vi holdt derfor konsekvent på farger som vi i planleggingsfasen bestemte oss for, se mer om dette i kapittel 3.8.1.

I applikasjonen bruker vi også skygger for å skape mer dybde på diverse komponenter. For eksempel:

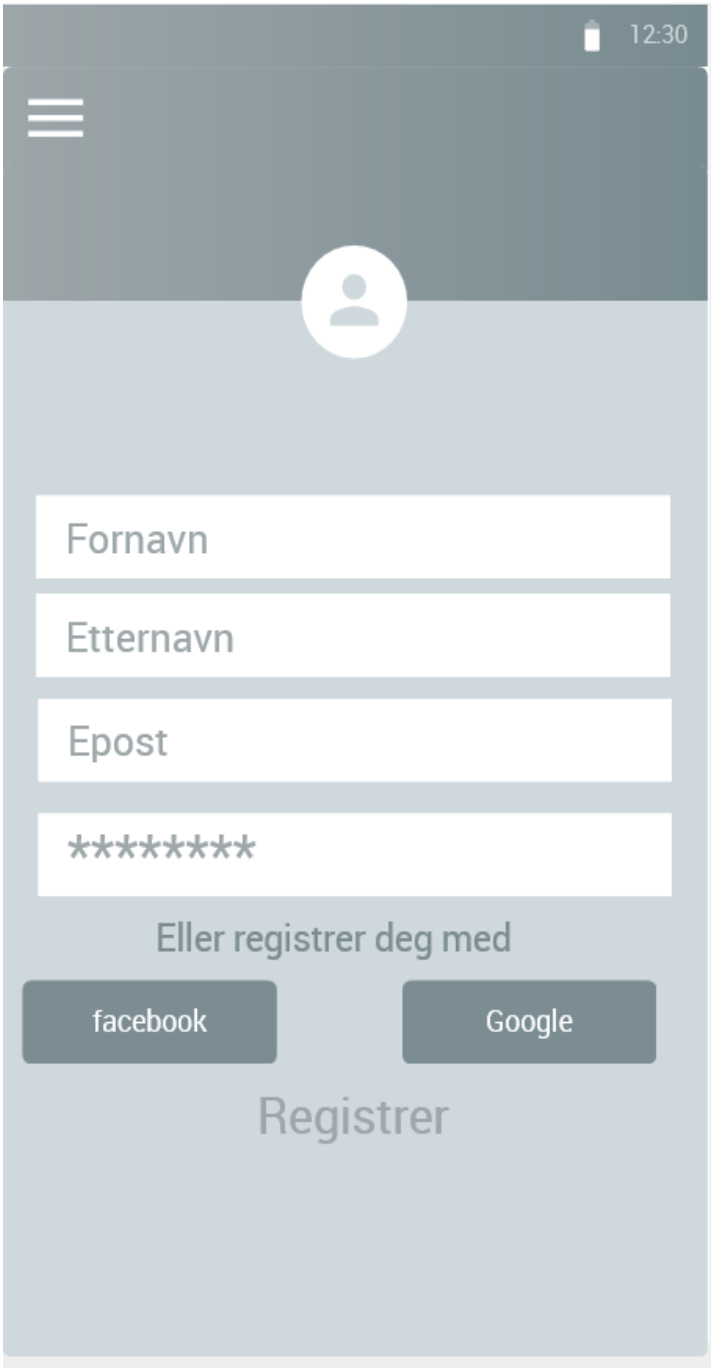
Når en skal legge til en ny oppgave eller endre innstillinger kommer det frem et nytt vindu som ligger fremfor alt annet på skjermen, her er det viktig å skape en mørkere farge på alt bak for å skape dybde på objektet.



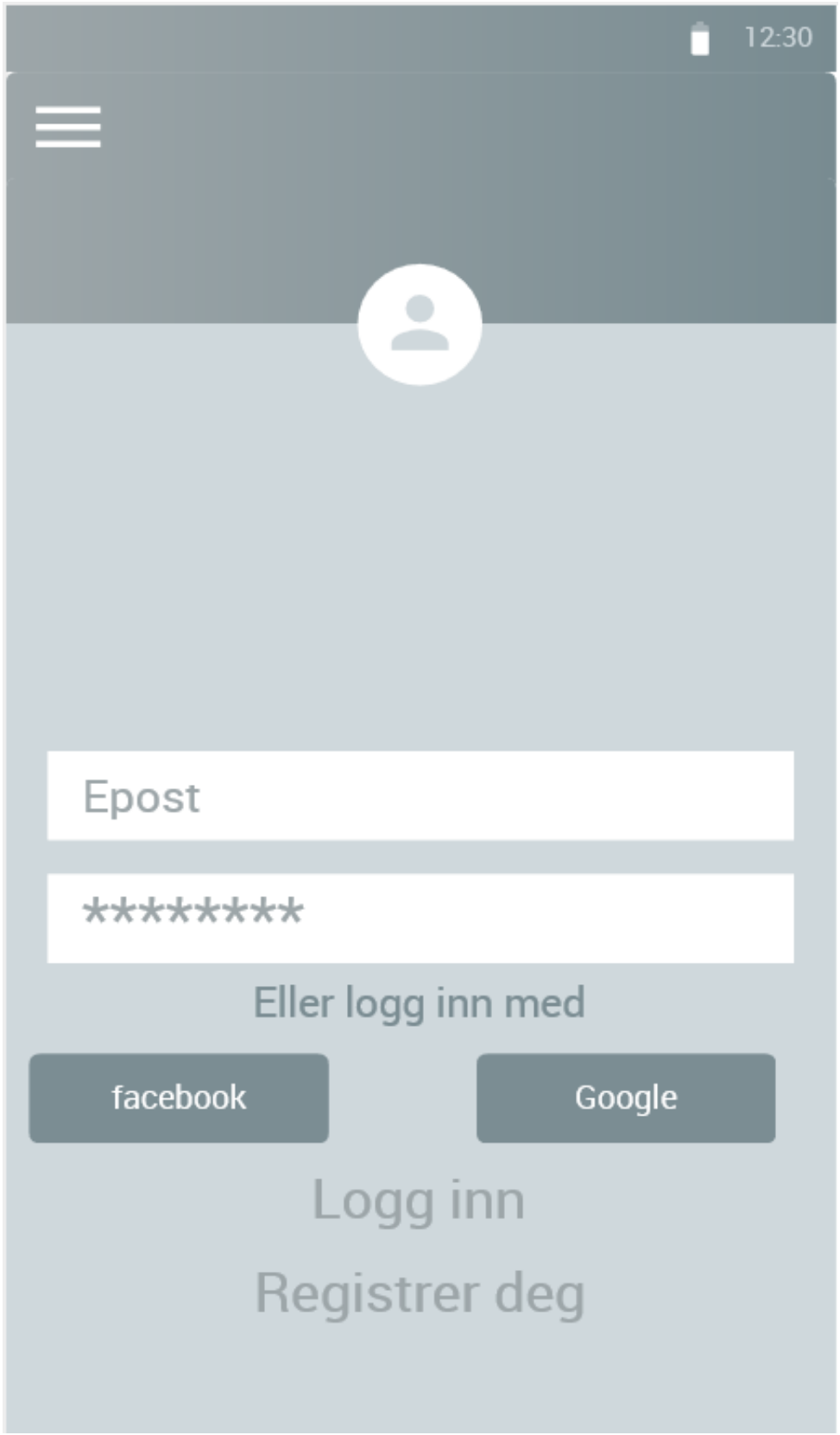
Figur 13

3.8.4 Wireframe

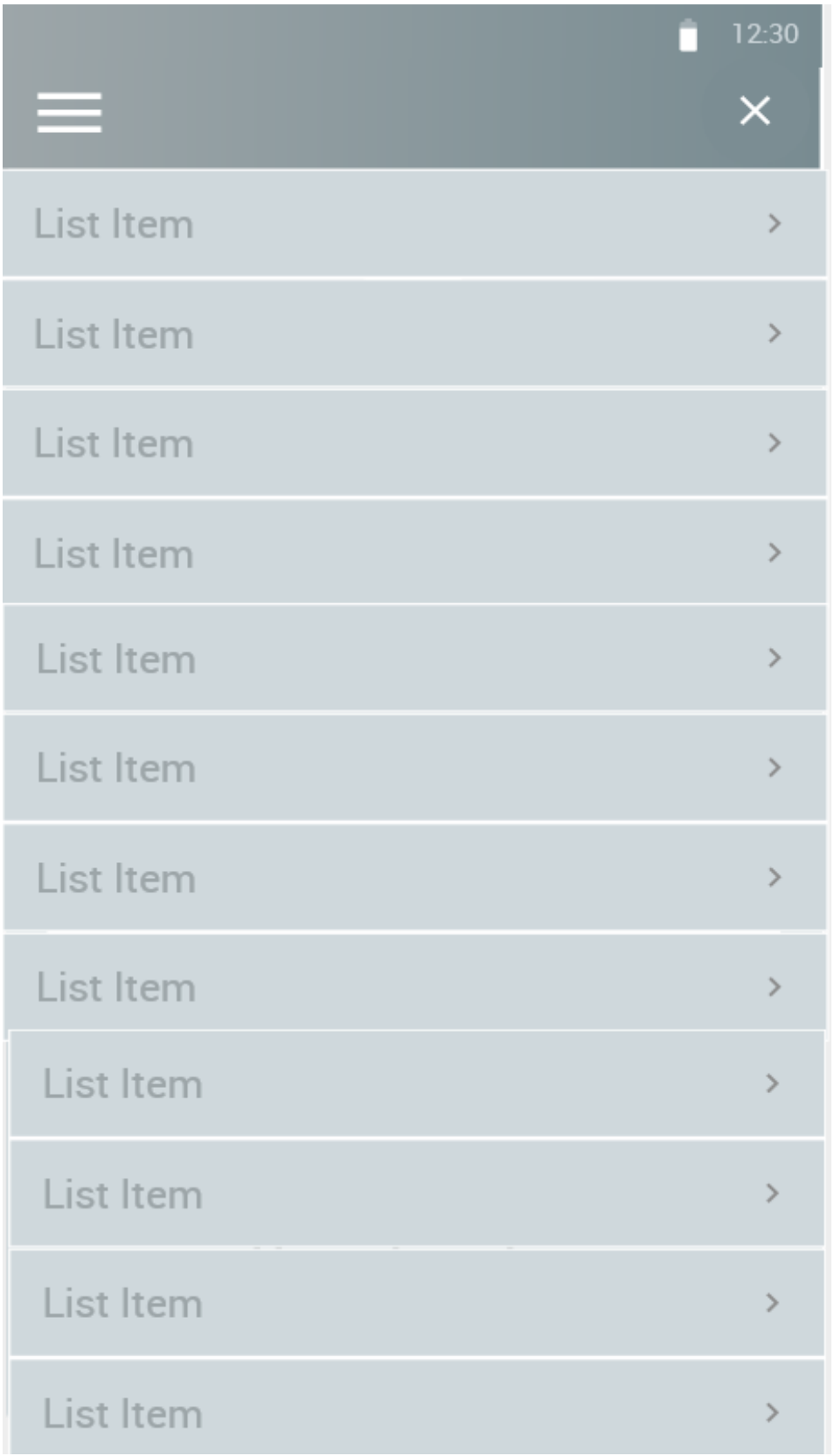
For design så er det enkle ofte det beste. Vi ønsket å bruke mest mulig ikoner og bilder for å vise hvordan en bruker applikasjonen. Hvis brukeren må lese mye tekst eller lete etter hvordan applikasjonen fungerer.



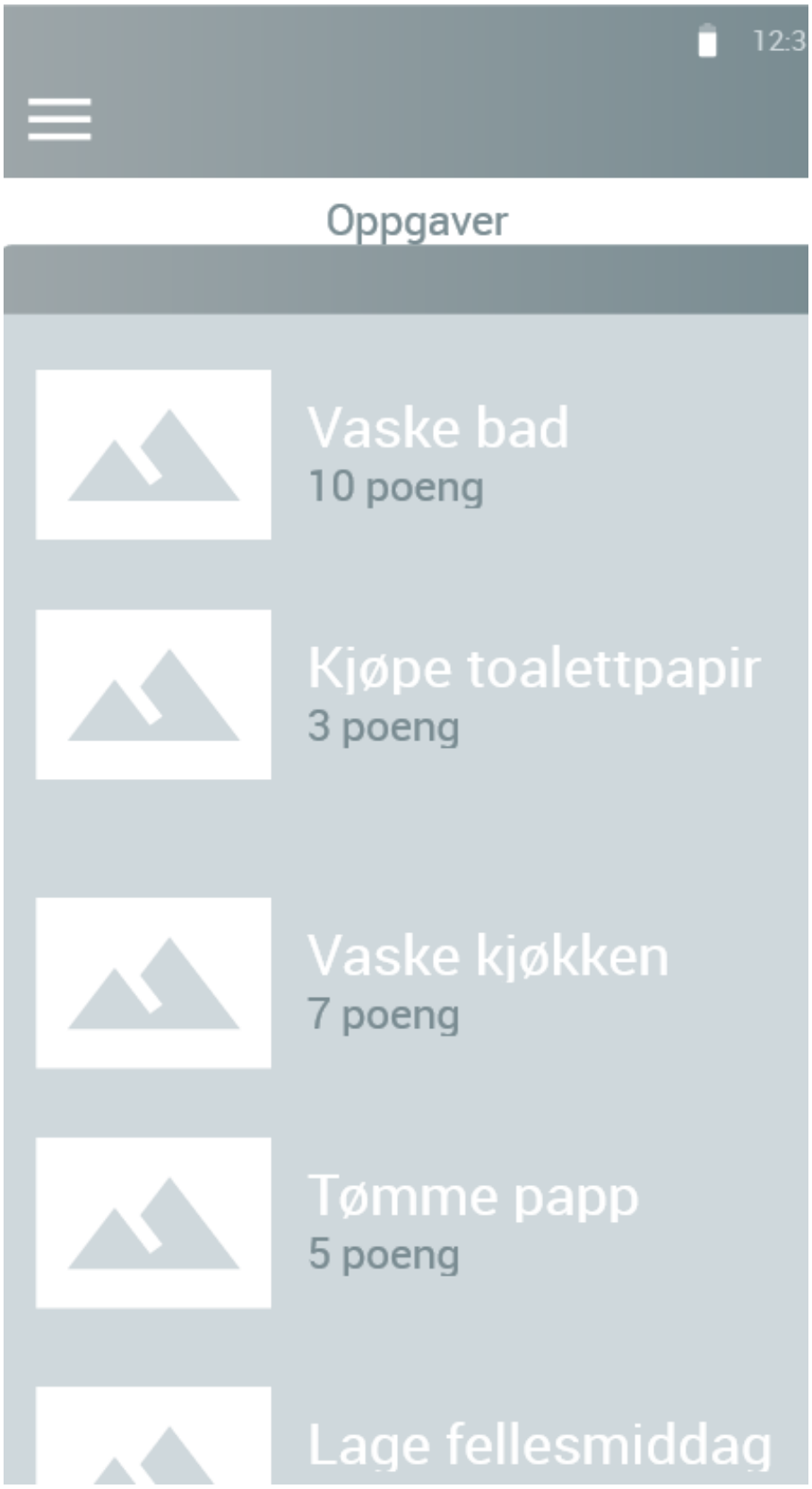
Figur 14


12:30
Epost
****
Eller logg inn med
facebook
Google
Logg inn
Registrer deg

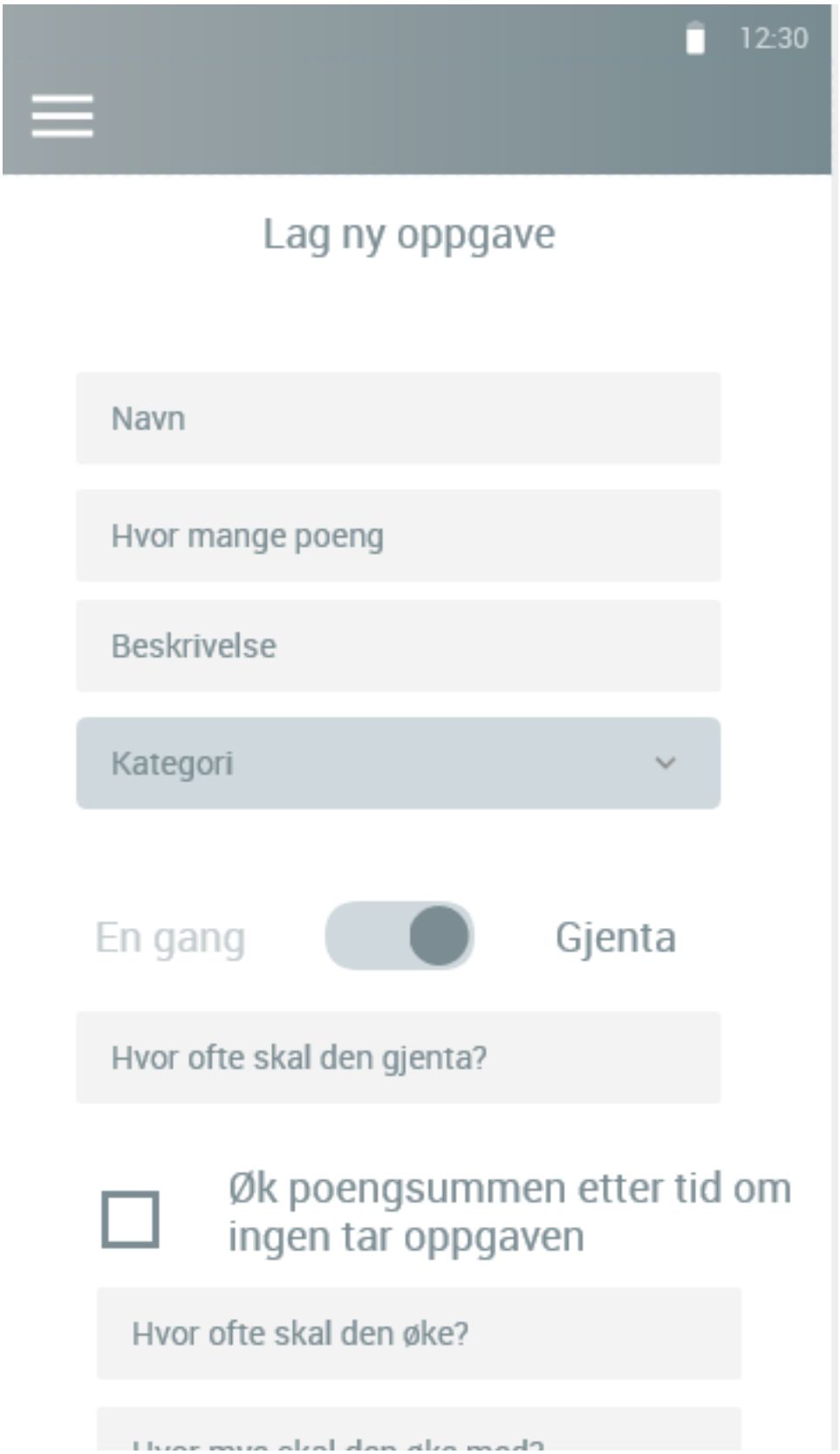
Figur 15


12:30
List Item
List Item
List Item
List Item
List Item
List Item
List Item
List Item
List Item
List Item
List Item
List Item

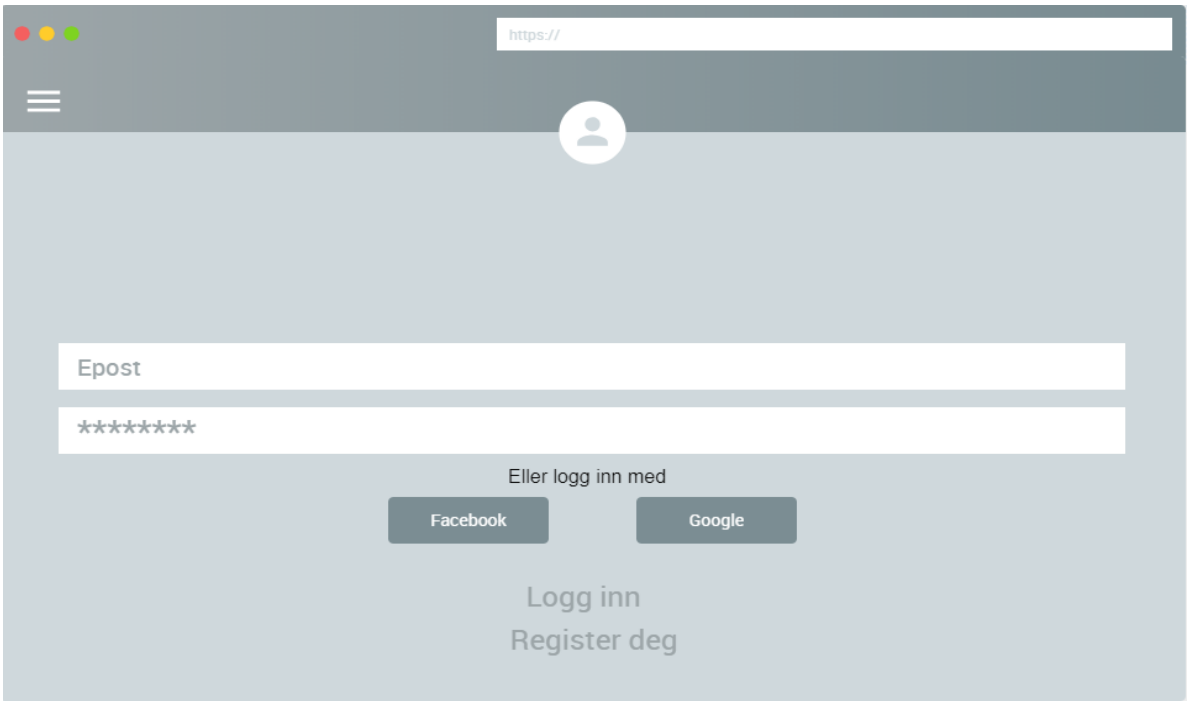
Figur 16


12:3
Oppgaver
Vaske bad
10 poeng
Kjøpe toalettpapir
3 poeng
Vaske kjøkken
7 poeng
Tømme papp
5 poeng
Lage fellesmiddag

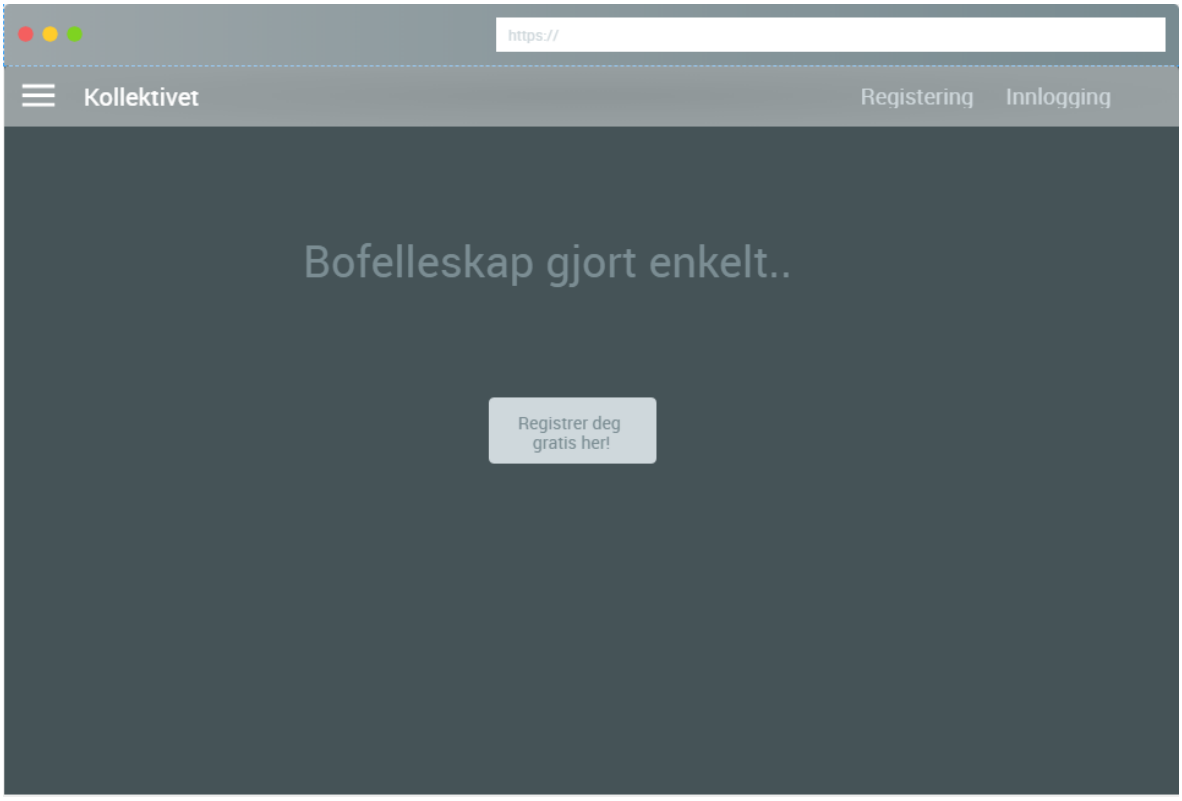
Figur 17


12:30
Lag ny oppgave
Navn
Hvor mange poeng
Beskrivelse
Kategori
En gang
Gjenta
Hvor ofte skal den gjenta?
Øk poengsummen etter tid om ingen tar oppgaven
Hvor ofte skal den øke?

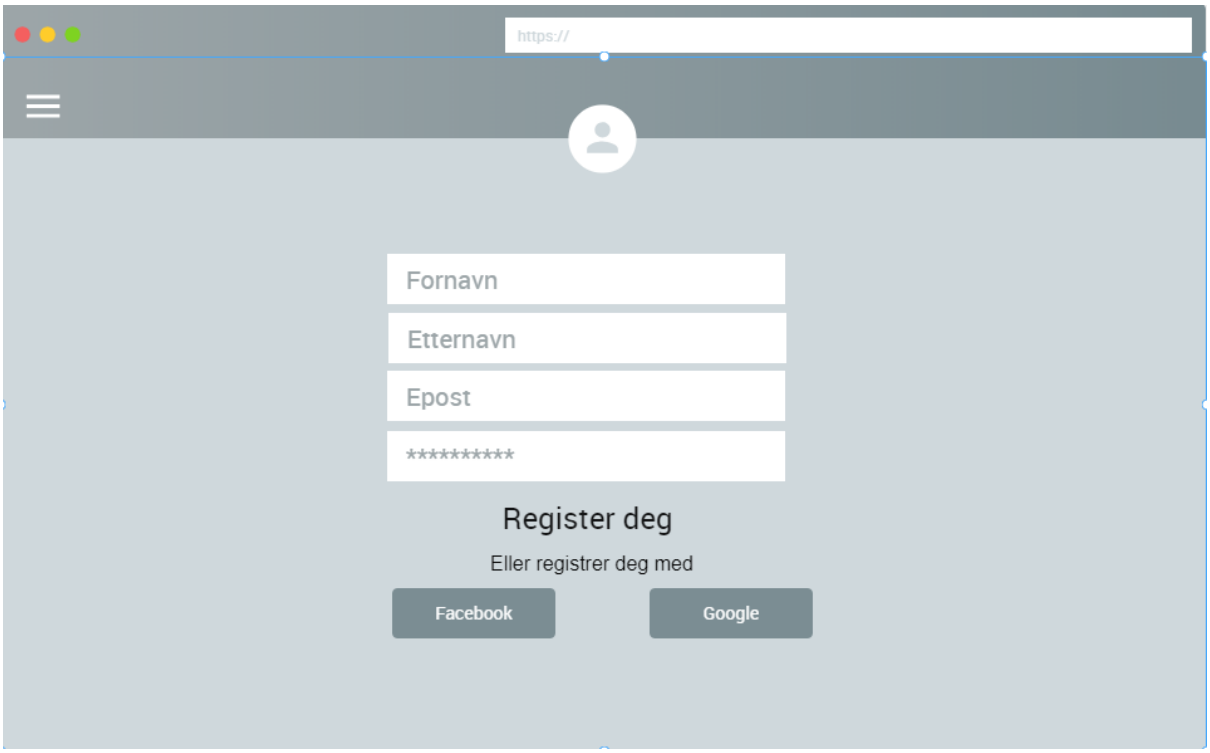
Figur 18



Figur 19

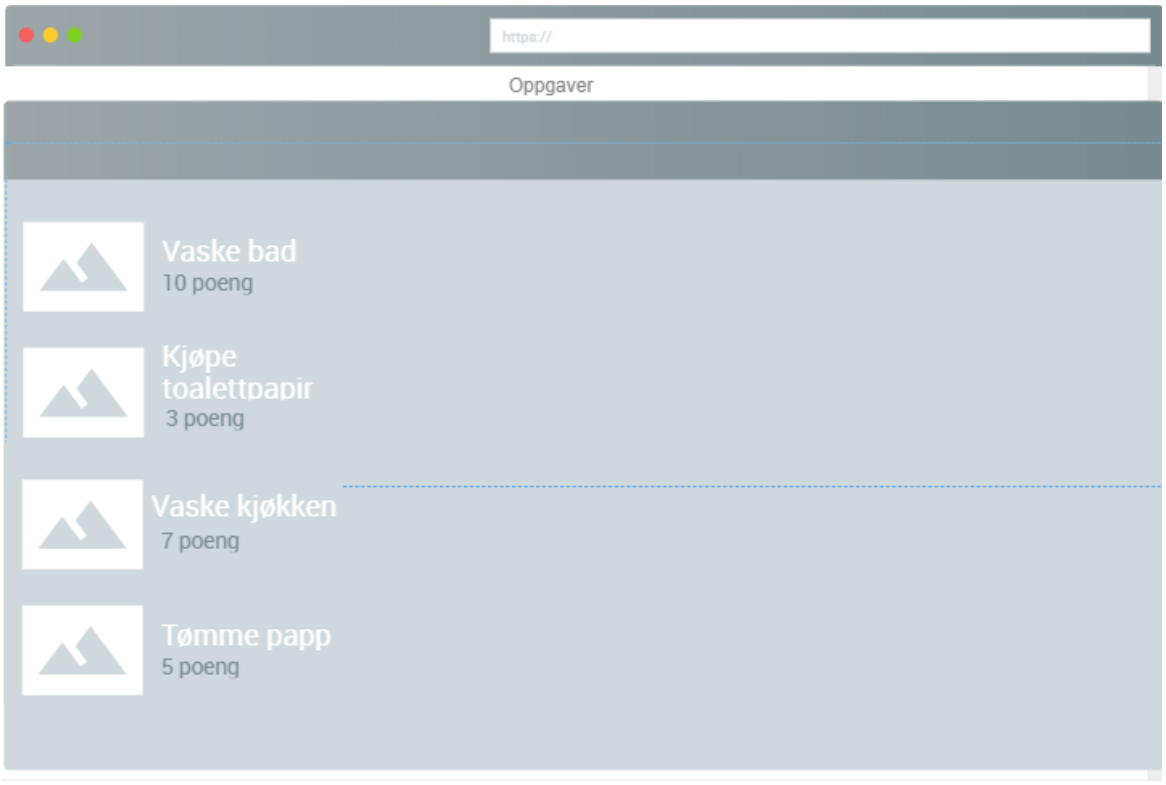


Figur 20

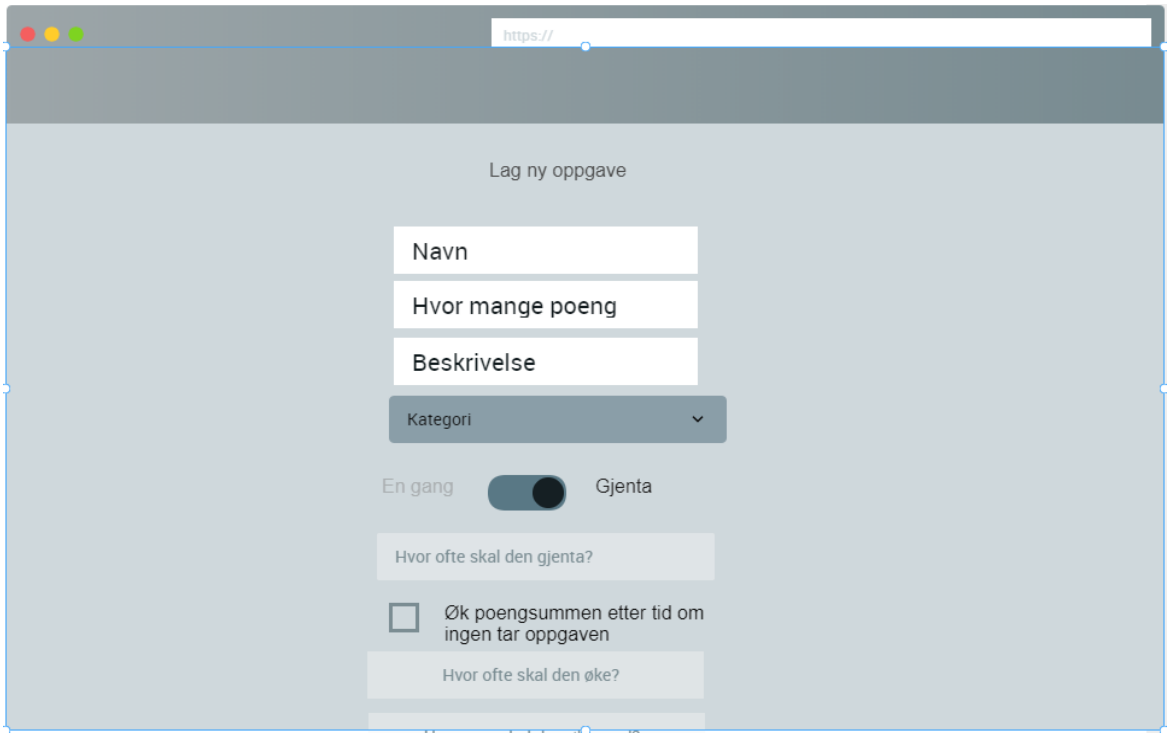


Figur 21

Figur 22



Figur 23



Figur 24

3.9 Dokumentasjon/kommunikasjon:

3.9.1 Discord

Mesteparten av kommunikasjonen har gått gjennom Discord. Her har vi både kommunisert via tekst og tale, vi også av og til delt skjermbilde for å vise eller demonstrere diverse problemer vi har hatt.
Discord er hovedsakelig brukt for kommunikasjon for med stemme. Når vi har arbeidet har vi for det meste satt hjemme og kommunisert via Discord.
Den eneste negative delen med Discord er at den ikke klarer å formatere kodetekst slik at vi måtte ta i bruk Slack.

3.9.2 Google docs

For å samarbeide om skrivingen og raskt kunne få tilbakemelding fra hverandre har vi brukt Google docs. Her kan vi se hverandre skrive i sanntid, og samtidig markere og kommentere ut deler av rapporten som må diskuteres.

3.9.3 Libreoffice

Libreoffice er en av de 2 skriveprogrammene vi har brukt for å skrive rapporten.
Libreoffice har blitt brukt til å sette sammen rapporten når vi har blitt ferdig å skrive alle delene i google docs.
Libreoffice er et enklere skriveprogram å formatere tekst og bilder i enn google docs. Vi har derfor knyttet alt sammen i dette programmet.

3.9.4 Slack

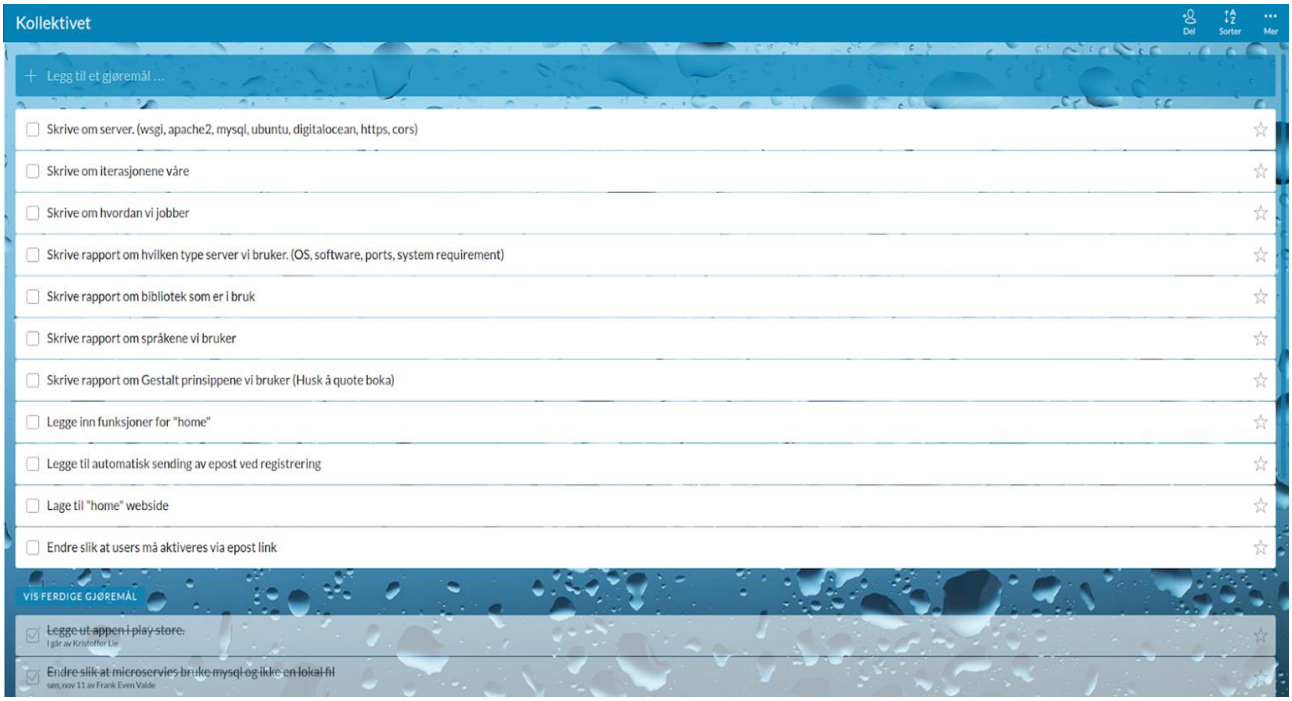
Discord manglet funksjon for deling av kode, og vi brukte derfor Slack for å holde kontroll på alt av kode, feilmeldinger og informasjon vi skulle trenge i prosjektet. Slack er laget til for kommunikasjon blant utviklere, og den har derfor funksjoner for å dele kode, noe som vi ofte brukte når vi trengte hjelp eller når vi bare ville diskutere kodesnutter.
Slack har muligheten for flere kanaler for å holde samtaler rundt forskjellige emnet separate. Vi valgte å bruke kanalene:

- Android
- Arbeidsplan
- design
- email service
- general
- healthcheck
- kollektiv service
- random
- server
- userservice
- webapplikasjon

Med kanalene kunne vi enkelt og ryddig måte lete opp informasjonen vi trengte som for eksempel kode, farger, bilder og annen informasjon vi tidligere har diskutert.3.9.5

3.9.5 Wunderlist

Wunderlist har blitt brukt for å holde oversikt over hver iterasjon. I starten av hver iterasjon så legger vi inn det som skal gjøres i listen vår på Wunderlist, dette gjelder arbeid med server, programmering, skriving av rapport, og eventuelt andre oppgaver vi møter på. Her informerer vi hverandre hvilken oppgave vi tar, for at vi ikke skal ta samme oppgave. Når vi er ferdig med oppgaven er det bare å krysse av oppgaven av listen så er den markert som ferdig.



Figur 25

4. Resultater

4.1 SYSTEMARKITEKTUR

4.1.1 Prosjektstruktur

Kollektivet er delt inn i 4 deler, Webapplikasjon som er frontend laget i React og de 3 mikroservicene, Kollektiv service, User service og Email service.

4.1.1.1 Webapplikasjon

Vi endte opp med et ganske greit og responsivt design, både for mobil og PC. Alle type skjermen har nå muligheten til å åpne og bruke applikasjonen uansett hvor mye en minimerer eller maksimerer skjermbildet så vil det nesten være brukelig.

Applikasjonen har mye funksjonalitet som:

- Registrering
- Logge inn
- Opprette kollektiv
- Bli med i et kollektiv
- Opprette oppgaver
- Hente invitasjonskode til kollektiv
- Forlate kollektiv
- Logge ut
- Aktivere konto

Det er også en del oppgaver vi ikke enda har rukket å implementere som:

- Ta oppgaver
- Sletting av oppgaver

- Fullføre oppgaver
- Sletting av kollektiv
- Hente statistikk om kollektivet
- Endre innstillinger
- Legge til og vise regler

Grunnen til at implementeringen ikke er ferdig er fordi vi brukte for mye tid på feil og manglende kunnskap som vi resulterte å bruke mye tid og energi på. Planen senere er å implementere disse for å få se en ferdig applikasjon som folk kan bruke.

4.1.1.2 Kollektiv service

Kollektiv service er mikroservicen som håndterer oppgaver og kollektiv, den har en egen database hvor data om disse er lagret. Kollektiv service er skrevet i Python/Django. Metodene som blir kallet via API i front end ved redigering, sletting, oppretting og lignende er i denne servicen.
Her ser vi et eksempel fra oppretting av et kollektiv, vi har kollektivnavn og brukerid som parameter for oppretting. Det blir også laget en hashet kode for å bli med i kollektivet, denne kan endres til hva du vil, men er til og begynne med hashet ut i fra kollektivnavnet og salt.

```
def createkollektiv(request,):
        kollektivName = request.data.get("kollektivName", "")
        userid = request.data.get("userid", "")
        salt = uuid.uuid4().hex
        code = hashlib.sha512(kollektivName.encode('utf-8') + salt.encode('utf-8')).hexdigest()[:8]
        if not kollektivName:
            return Response(
                data={
                    "message": "Du må fylle inn et navn for kollektivet ditt"
                },
                status=status.HTTP_400_BAD_REQUEST
            )
        new_kollektiv = kollektiv.objects.create_kollektiv(
            kollektivName=kollektivName, code=code)
        new_kollektivmember = kollektivmember.objects.create_kollektivmember(kollektivid=new_kollektiv.pk, userid=userid, points=0, isAdmin=True)

        return Response(status=status.HTTP_201_CREATED)
```

Figur 26

Om du den som lager et kollektiv ikke har puttet inn et navn vil en feilmelding med http status 400 bli sendt tilbake, men om opprettingen er en suksess brukes ORM til å sende data til databasen.

4.1.1.3 User service

I User service blir all data tilhørende brukere håndtert, den logger deg inn, oppretter token, registrer brukere. Den har også en egen database for brukere. Det er via api front end har tilgang til servicen. Servicen er laget i Python/django.
Her ser vi metoden for oppretting av bruker, metoden krever en epost, et passord og et navn. Det første den gjør er å lage til en aktiveringskode for brukeren ved å hashe emailen med et salt, denne blir sendt til gitt epost med en lenke som gir et kall for aktivering.
Etter å ha laget en kode for aktivering sjekker den om feltene for epost og passord er fylt inn, om det ikke er det blir det sendt en http status 400.
Om brukeren og all riktig informasjon er på plass vil brukeren bli opprettet i databasen og en «http 201 created» sendt til front end.

```
def register(request, ):
    email = request.data.get("email", "")
    password = request.data.get("password", "")
    name = request.data.get("name","")
    salt = uuid.uuid4().hex
    activation_key = hashlib.sha512(email.encode('utf-8') + salt.encode('utf-8')).hexdigest()[:8]

    if not email and not password:
        return Response(
            data={
                "message": "username, password and email is required to register a user"
            },
            status=status.HTTP_400_BAD_REQUEST
        )
    new_user = User.objects.create_user(
        email=email, password=password, name=name, activated=False, confirmToken=activation_key
    )
    print(activation_key)
    url = domain+":8001/email/register/?activationcode=" + activation_key + "&recipient=" + email
    response = requests.post(url).text
    emailreg = json.loads(response)
    return Response(emailreg, status=status.HTTP_201_CREATED)
```

Figur 27

4.1.1.4 Email service

Fra Email service er en rest API som kan sende ut epost fra hvilken som helst adresse fra domenet «kollektivet.app». Koden sender epost ved hjelp av epost tjenesten fra Domeneshop hvor vi også har kjøpt domene navnet.
Fra Email service kan en sende:

- Epost til alle brukarene samtidig
- En enkel epost til hvem som helst
- Automatisk aktiveringslink for nye brukere
- Automatisk epost om brukere har endret epost eller passord
- Automatisk «glemt passord» epost

Epost service gjør ikke dette helt av seg selv, men blir kjørt via andre servicer som ønsker å bruke denne. Dette gjør at vi enkelt kan gjenbruke koden.
Som de andre mikroservicene er denne også skrevet i Python/Django.

4.2 Database

Kollektiv service og User service har begge hver sin database hvor tilhørende informasjon ligger. Databasen i Kollektiv service er bestående av tabellene:

- Kollektiv - Her lagres selve kollektivnavnene, den har en primærnøkkel id, et navn på kollektivet (kollektivName) og en kode for å kunne bli med (code).
- KollektivMember - Her ligge sammenslåingen mellom bruker og kollektiv. KollektivMember består av en primærnøkkel, en fremmednøkkel tilknyttet kollektivid, en fremmednøkkel for brukerid, antall poeng bruker har i gitt kollektiv og om vedkommende er administrator i kollektivet (isAdmin).
- Task - I Task blir alt om oppgaver lagret, det har en primærnøkkel id, navn på oppgave (taskName), en fremmednøkkel til kollektivet den hører til hos(kollektivid), beskrivelse av oppgaver (taskDescription), hvor mange poenger den er verdt (taskPoints), datoen den er opprettet (dateCreated), tidsrommet den skal gjøres i (taskDuration), en sjekk om oppgaven er tildelt (taskTaken), hvem

som har tatt gitt oppgave (taskTaker), en måte å sjekke om oppgaven er fullført (taskDone), om oppgaven skal repeteres (taskRepeat) og hvor mange stemmer for godkjenning oppgaven har (taskVotes)

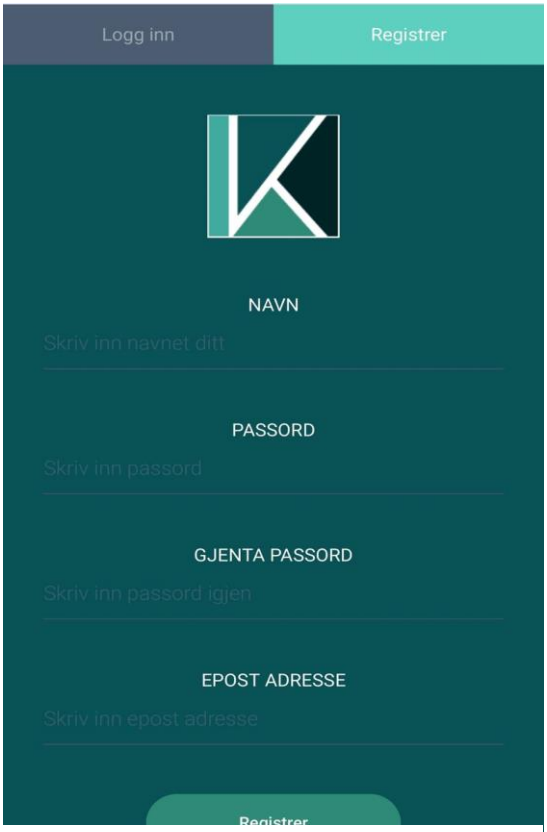
- Votes - En tabell for å oppbevare avstemminger til oppgaver.
  Votes lagrer en primærnøkkel id, en fremmednøkkel for oppgaven den tilhører(taskID), fremmednøkkel for hvilken bruker som har stemt (userid) og om vedkommende har stemt eller ikke (hasVoted).

User service har tabellen:

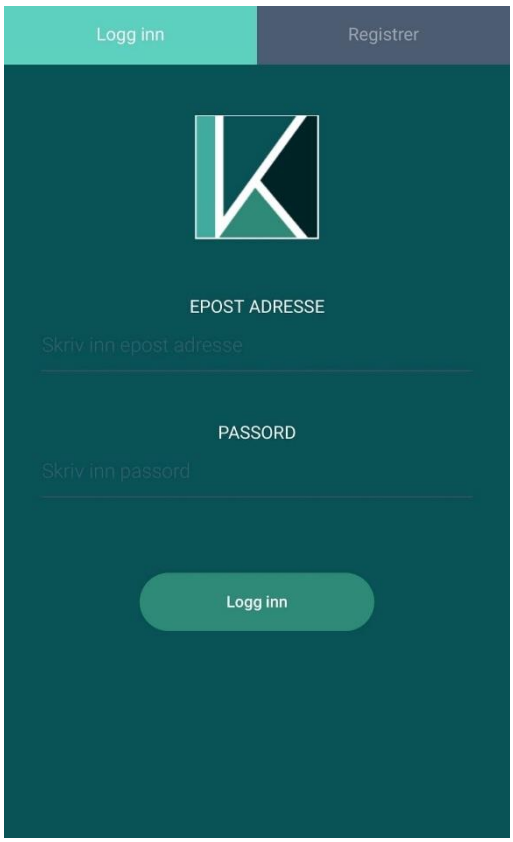
- User, den består av en primærnøkkel userid, email som brukes som brukernavn til innlogging(email), navn på brukeren(name), et felt som sjekker om brukeren er aktivert(activated) og en hashet variabel for aktivering av kontoen, denne blir sendt på mail ved registrering(confirmToken).

4.3 WebApp Design

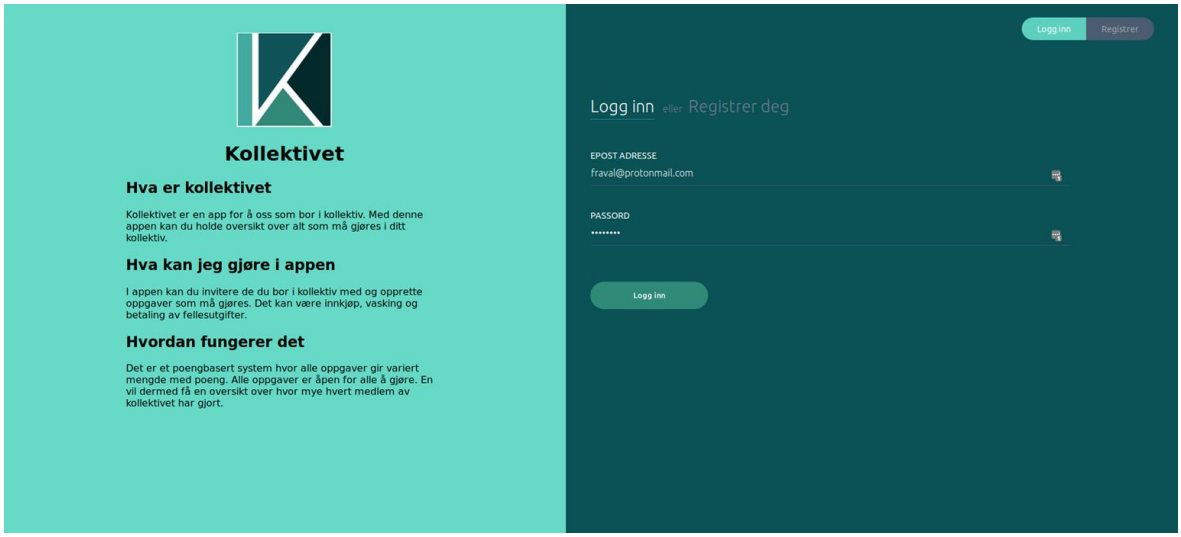
4.3.1 Login/Registrering



Figur 28



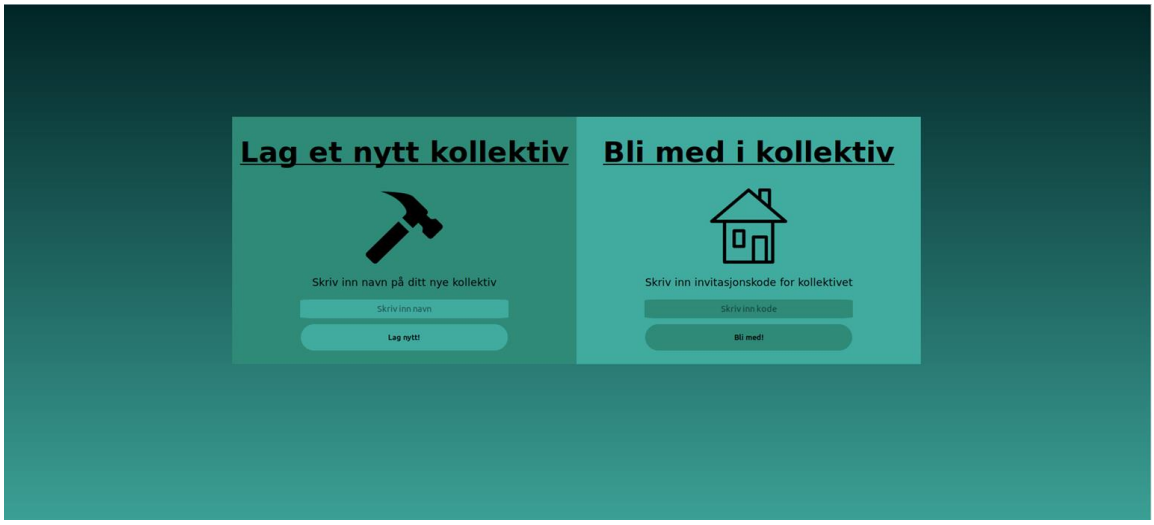
Figur 29


Kollektivet
Hva er kollektivet
Kollektivet er en app for å oss som bor i kollektiv. Med denne appen kan du holde oversikt over alt som må gjøres i ditt kollektiv.
Hva kan jeg gjøre i appen
I appen kan du invitere de du bor i kollektiv med og opprette oppgaver som må gjøres. Det kan være innkjøp, vasking og betaling av fellesutgifter.
Hvordan fungerer det
Det er et poengbasert system hvor alle oppgaver gir variert mengde med poeng. Alle oppgaver er åpen for alle å gjøre. En vil dermed få en oversikt over hvor mye hvert medlem av kollektivet har gjort.
Logg inn
Registrer
Logg inn eller Registrer deg
EPOST ADRESSE
fraval@protonmail.com
PASSORD
Logg inn

Figur 30

Når en besøker kollektivet på en datamaskin er dette skjermbildet det første som møter deg. Logg inn-siden består av et todelt skjermbilde, det til venstre viser litt informasjon om applikasjonen og logoen. På Høgresiden er da logg inn-feltet eller registreringen ut i fra hvilken du har valgt. For å skifte mellom Logg inn og registrering er det en sideskifter eller «page switcher» på toppen til høgre, denne er også koblet sammen med teksten Logg inn og Registrer deg for brukervennlighet.
Om du bruker mobilapplikasjonen til kollektivet er det et litt mer simpel design som er deg i møte, alt er minimalisert for mobilbruk.
Vi er godt fornøyd med designet på mobil, det har en mer moderne tone.
Som på dataversjonen er det også en sideskifter eller «page switcher» for å lett bytte mellom Logg inn og Registreringen.

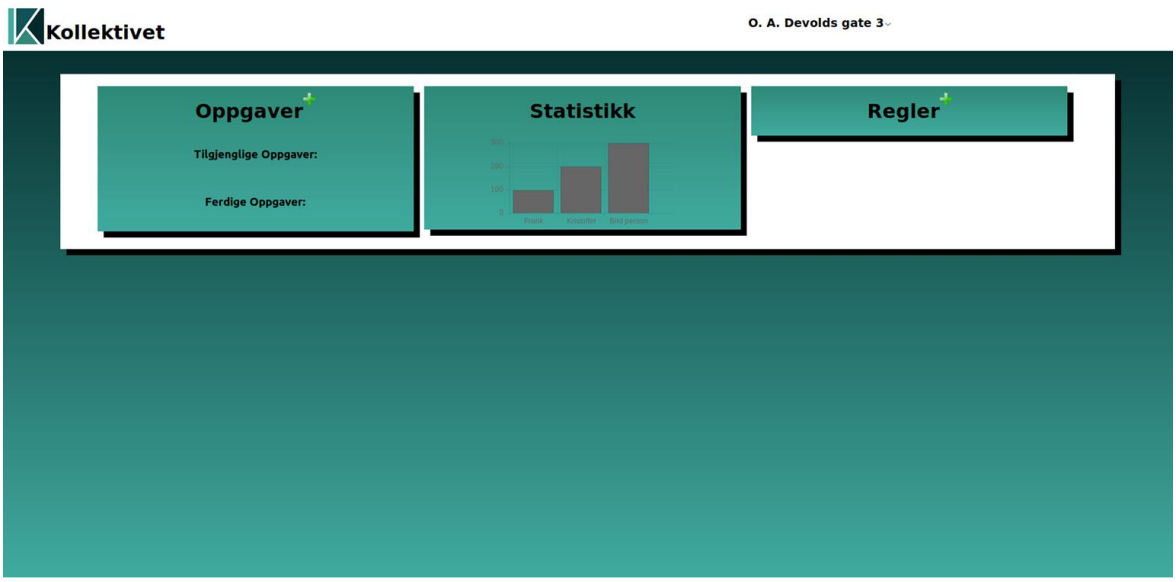
4.3.2 Bli med Kollektiv / Lag et Kollektiv



Figur 31

Om du nettopp har registrert deg i kollektivet blir du møtt av en side som forteller deg at du må enten bli med i et kollektiv eller opprette et nytt et. For å bli med i et kollektiv trenger du en invitasjonskode, den kan alle som er medlem i et kollektiv hente fra hovedsiden sin meny ved å klikke på kollektivnavnet.
Om du vil opprette et nytt kollektiv skriver du inn et navn på kollektivet og trykket «Lag nytt!».

4.3.3 Hovedside



Figur 32

Figur 33

Når du har logget inn og er medlem av et kollektiv vil du møte hovedsiden til kollektivet, her ser du alle opprettede oppgaver under oppgave delen av siden, til høgre for den er statistikk delen her vil du fremtidig kunne se hvor mange poeng hvert medlem av ditt kollektiv har opptjent seg. Og til slutt er det regler, her er det mulig å legge til egendefinerte regler for bokollektivet eller huskeliste.
Øverst på hovedsiden er navnet til kollektivet, trykker du på denne vil du få en meny med flere valg:

Legg til medlem	Her får du kollektivets invitasjonskode
Innstillinger	Her skal brukerinnstillinger være
Forlat kollektiv	Trykker du på denne forlater du ditt nåværende kollektiv
Logg ut	Knapp for å logge ut av kollektivet

4.4 API

Applikasjonen fungerer ved API-kall som sendes over HTTP fra frontend til backend.
I Webapplikasjonen vår har vi noen lenker som sender kall:

- /login sender passord og email til backend for autentisering.
- /register sender informasjonen din til backend for registrering av bruker.
- /home er hovedsiden, om du er innlogget ser du oppgaver, statistikk og regler.
  Dersom du ikke er innlogget vil du bli videresendt til Innloggingsbilde.
  Om du har aktivert brukeren din vil du få alternativet mellom å bli med i et kollektiv eller lage et kollektiv.

Videre funksjonalitet ligger under følgende kall:

- /user/ er hvor alle kall tilhørende brukere ligger
- user/login håndterer alle kall om å logge inn og gir token til brukeren.
- user/register er kallet som lager brukeren i databasen
- user/ChangePasswordView her håndterer endringer av passord
- user/ChangeEmailView er kallet for å endre epost adresse for brukeren.
- user/EmailList kalles fra Email service for å få liste over alle registrerte emailer, den er laget for å sende ut nyheter eller lignende til alle registrerte hos applikasjonen.
- user/postid brukes fra Kollektiv service for å kunne lagre riktig bruker hos riktig kollektiv.
- user/activated kalles fra Email service, den bruker denne til å lage en lenke for å aktivere brukere.
- user/getNames brukes i frontend for å vise brukeren sitt navn.
- user/inKollektiv/ er til for å sjekke om brukere er medlemmer av kollektiv, om den returnerer 0 vil frontend vite at den må gi brukeren mulighet til å bli med i et kollektiv.

- /kollektiv/ er hvor kall tilhørende kollektiv og oppgaver ligger.
- kollektiv/createKollektiv/ kalles for å opprette kollektiv, alt den trenger er et kollektivnavn.
- kollektiv/deleteKollektiv/ kalles om et kollektiv skal slettes.
- kollektiv/showKollektiv/ brukes for å vise hvilket kollektiv brukeren er medlem av.
- kollektiv/showAllKollektiv/ kalles når administratorer trenger en liste over alle kollektiver som er opprettet
- kollektiv/createTask/ blir kallet når oppgaver skal legges til i kollektiv, her fyller vi inn navn på oppgave, beskrivelse av oppgaven, hvor mange poeng en oppgave er verdt og kollektivets id som automatisk blir lagt inn fra front end.
- kollektiv/deleteTask/ gjør det mulig å fjerne oppgaver som allerede er opprettet.
- kollektiv/getUserTasks/ sender alle oppgavene som er tilknyttet brukeren og kollektivet.

- kollektiv/getUserTasksDone/ sender tilbake en liste over alle oppgaver som er fullførte.
- kollektiv/getIdinKollektiv/ kalles for å lage en liste over alle brukere tilhørende et gitt kollektiv.
- kollektiv/getPointsInKollektiv/ sender alle poengene til hver bruker i et gitt kollektiv.
- kollektiv/editTask/ kalles når det krever endringer i en allerede opprettet oppgave.
- kollektiv/takeTask/ gjør at en bruker kan ta på seg en oppgave som vedkommende skal gjøre.
- kollektiv/taskDone kalles når en oppgave er fullført, den putter den i listen over ferdige oppgaver og gir brukeren poeng for fullført arbeid.
- kollektiv/taskNotDone/ kalles om en oppgave må endre status fra ferdig til ikke ferdig.
- kollektiv/joinKol/ brukes for å melde en bruker inn i et kollektiv.
- kollektiv/isMember/ sjekker om en bruker er medlem av et kollektiv, den returnerer true eller false.
- kollektiv/getInvite/ kalles når en bruker vil ha en invitasjonskode til et gitt kollektiv.
- kollektiv/leaveKollektiv/ kalles for å forlate kollektivet du er medlem av.
- kollektiv/getKollektivID/ brukes for å få kollektivets id ut fra en bruker.

- /email/ sender mail til brukerne.
- /email/register/ sender email til brukeren etter vedkommende har registrert seg.
- /email/resetpassord/ kalles når brukeren ber om å endre passord.
- /email/sendtoall/ sender en mail til alle registrerte brukere, brukes i sammenheng av nyheter eller beskjeder.
- /email/newsletter/ brukes for å sende nyheter, den kaller /sendtoall/.
- /email/passwordchanged bekrefter at en bruker har byttet passord.
- /email/emailchanged bekrefter at en bruker prøver å bytte email.

4.5 Server

Vi bruker en virtuell server som vi leier av Digital Ocean. Serveren har Ipv4 adressen «104.248.44.55» med DNS «kollektivet.app", domenet er kjøpt hos Domeneshop.no.

Serveren bruker Ubuntu som operativsystem og kjører http server tjenesten via programvaren Apache2. Serveren er ferdig konfigurert med automatisk https, det er derfor ikke mulig å komme inn på nettsiden med vanlig http. Serveren har signert SSL sertifikat av «Let's Encrypt», som automatisk oppdaterer seg når den går ut på dato.

Web serveren er ferdig konfigurert til å kjøre hver microservice vi har laget. Det eneste vi trenger å gjøre er å gjøre før vi kopierer over prosjektet er å forsikre oss om at de virtuelle miljøene har installert de pakkene som de trenger for å kjøre prosjektet. Ellers er det bare å kopiere over.

I dag er det ingen script som automatisk tar seg av utrulling av siste versjonen av programvare. Dette er noe vi tenker å sette opp i fremtiden for å slippe menneskelige feil under denne prosessen.

4.6 KJENTE FEIL ELLER MANGLER

I slutten av prosjektet merket vi at vi ikke hadde tid til å implementere alle metodene til front end, vi måtte fokusere på at all funksjonalitet vi allerede hadde fungerte som det skulle. Vi måtte avslutte arbeidet på statistikk-delen av siden, regler, avstemninger, sletting av oppgave og fullføring av oppgaver. Det blir utsatt til senere iterasjoner. Det hender at frontend kaller api-kall uten at det er behov for det, noe som kan føre til feilmeldinger i backend, uten at applikasjonen kræsjer, men vi kan se det i konsollen. Det er heller ikke feilmeldinger for alle brukerfeil som blir gjort.

4.6.1 Frontend

Vi mangler funksjoner for:

- Ta oppgaver
- Sletting av oppgaver
- Fullføre oppgaver
- Sletting av kollektiv
- Hente statistikk om kollektivet
- Endre innstillinger
- Legge til og vise regler

Se begrunnelse i kapittel 4.1.1.1

4.6.2 Manglende implementasjoner

Vi hadde planer om å implementere en helsesjekk for å sjekke om alle servicene våre var oppe og kjørte, men vi fikk ikke nok tid til å prioritere dette enda, det er flyttet til en senere iterasjon. Planen var å kunne ha en side hvor alle servicene var listet opp, om sjekken gikk gjennom ville det vise et grønt lys ved siden av navnet og rødt lys om den var nede.

5. Drøfting

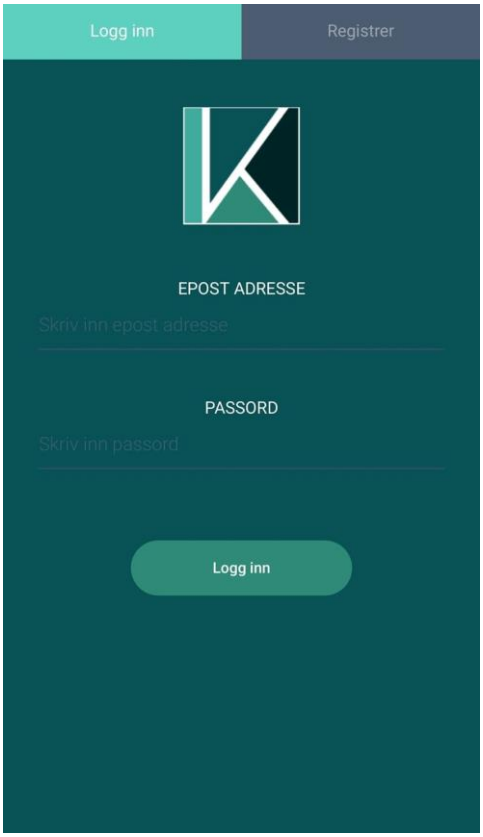
5.1 EVALUERINGER

5.1.1 Resultatet

Hovedfunksjonaliteten er fungerende, man kan logge inn, registrere seg, legge til oppgaver og bli med i kollektiv. Applikasjonen er også fungerende på både mobil og web.

5.1.1.1 Designet

Planen med brukergrensesnittet var å utforme noe som var simpelt, lett forståelig og funksjonelt. Vi ønsker ikke å lage noe nytt og avansert som ofte gjør applikasjonen mindre brukervennlig. Vi holder oss til samme temaet i hele applikasjonen med diverse grønne farger, for å gi inntrykk av at dette er en sammenhengende. Vi følger Gestaltprinsippet "kontinuitet" og holder et lik design for både mobil og skrivebord, slik at brukere føler at de alltid er i samme applikasjon (Spokane falls community college, 2018). All design og plassering er laget selv i CSS uten hjelp av biblioteker for å skape et unikt og eget design. Veldig mange av komponentene har egen klasse fordi vi enkelt skal kunne endre plassering og stil på enkelte elementer uten at det påvirker resten.



Figur 34

Mobil og skrivebord deler på CSS fil, men har helt separate klasser for å skille mellom disse. Mobil og nettbrett får nettsiden strukket utover hele skjermen ellers blir det veldig smått. Skrivebord har flere elementer fremme samtidig for en delt visning for å bruke det store området en pc skjerm har, og for å redusere antall museklikk før en finner det en skal ha tak i det en skal ha. Ikonene som er brukt i grensesnittet er hentet fra iconarchive.com hvor de har tilgjengelig ikoner som vi kan bruke gratis på vår nettside (Icon Archive, 2018). Ikonene som er valgt er laget av enkle svarte streker for å holde på det enkle designet vi går for og samtidig gjøre brukergrensesnittet finere. Brukergrensesnittet er responsivt og vil fungere fint uansett hvilken størrelse en skjerm en velger å bruke. Selv synes vi at designet ble enkelt og funksjonelt, men ikke så fint som vi skulle ønske. Ellers synes vi brukergrensesnittet utfører jobben den er designet for.

Figur 35

5.1.1.2 Frontend

I frontend har vi brukt html og CSS for å representere informasjon til nettleser. Vi har også brukt et JavaScript bibliotek som heter React for å generere JavaScript som skaper et mer dynamisk brukergrensesnitt. React er også ansvarlig for å bruke metodene vi har i backend og behandle denne dataen slik at vi kan representere den på best mulig måte.

5.1.1.3 Backend

I Backend er det brukt Python/Django og tar for det meste imot API-kall fra front end med noen parameter. Front end sender for eksempel email og passord, backend vil da godkjenne og sende tilbake en token for videre autorisering ved bruk av applikasjonen. Det tok en del tid å skjønne hvordan alt fungerte i Django og det gjorde denne delen mer komplisert enn tidligere antatt. Når vi først kom litt i gang gikk det derimot raskt å utvikle metoder i backend.
Det var mye likt ved sending av informasjon og vi kunne derfor modifisere tidligere brukte metoder.

5.1.1.4 Sikkerhet

For sikkerhet bruker vi https med signert sertifikat for å holde all kommunikasjon mellom frontend og backend kryptert. Metoder som er laget i Django er beskyttet fra “Cross-site scripting” med Cross-Origin Resource sharing (CORS) som forhindrer andre fra å bruke skript fra andre kilder enn de vi velger. For å koble på serveren via Secure Shell (SSH) må vi bruke en nøkkel med et passord. Mye av sikkerheten ligger i programmene vi bruker, det gjelder alt fra operativsystemet til http server programvare. Det er derfor veldig viktig å holde programvare oppdatert til enhver tid slik av vi unngår å ha sikkerhetshull.
Den eneste kritiske informasjonen vi har i databasen er passord. Om vi skulle være så uheldig at noen får tak i all informasjon som vi har i databasen vår så er alle passordene kryptert, og det vil derfor ta tid før de får tak i passordene. Om vi finner ut at dette skjer vil vi så raskt som mulig sende ut en epost til alle brukere om å endre passordene sine.

5.1.2 Rammeverk og verktøy

Under utføringen av prosjektet tok vi i bruk noen rammeverk for å gjøre noen oppgaver mer effektiv og enklere å utføre. Her vil vi diskutere hva som gikk bra eller dårlig med de brukte rammeverkene og hvor det kunne vært en fordel å bruke et rammeverk.

5.1.2.1 React

For å unngå å få en statisk nettside valgte vi et JavaScript bibliotek som heter React. Ingen av oss har tidligere erfaring rundt noen JavaScript rammeverk eller bibliotek, og vi brukte derfor litt tid for å sette oss inn i hvordan dette fungerte. React brukte mye mer tid enn forventet og er det mesteparten av ressursene våre gikk til. React er grunnen til at vi ikke rakk å gjøre alt som vi først hadde planer om.

5.1.2.2 Django

Det var mange positive sider ved Django, det er lett kommunikasjon med database og den setter også opp databasen ved å definere modeller som hører til tables.
Django er bygget opp for enkelt å håndtere API, det er standard i en Django applikasjon å ha en fil som håndterer alle de forskjellige lenkene der du fritt kan endre på hvilke metoder som tilhører en gitt lenke.

5.1.3 Prosjektet

Prosjektet fikk en treg start med mye å lære på kort tid og eventuelt gikk det raskere ettersom vi lære oss hvordan både Django og React fungerte. Mesteparten av tiden har blitt brukt på å gjøre feil og finne ut hvordan vi på best måte skulle skape grensesnitt og behandle data. Med erfaringen vi har i dag hadde vi klart dette prosjektet på en mye kortere tid og fått inn flere ferdige funksjoner.

5.1.3.1 Scrum

Som beskrevet tidligere har vi brukt Scrum. Siden vi bare var to personer var det litt mer uformelt enn vanlig scrum. Vi synes det fungerte bra, men vi tok for eksempel ikke 15 minutter hver dag for å gå igjennom hva vi holdt på med, det var mer kontinuerlig kommunikasjon om dette.
Måten vi kunne selv velge hvilke deler av prosjektet vi trengte å arbeide på til enhver tid gjorde det enklere for arbeidsfordelingen og prioriteringen.

5.1.3.2 Samarbeid

Vi arbeidet godt i lag og har tidligere erfaring med å jobbe sammen.
Det som gikk bra:

- God arbeidsflyt når vi arbeidet sammen
- God inndeling av oppgaver, den som fikk til backend/frontend mest effektivt jobbet med kritiske deler.
- Innsats, vi arbeidet alltid for å fullføre så mye av iterasjonene som mulig.

Det som ikke gikk så bra:

- Manglende kompetanse og for stor oppgave for to personer.
- Lite forhåndskunnskap om React/Django.
- Ingen med stor erfaring innen design.

Vi arbeidet for det meste hjemmefra med kommunikasjon over nettet, vi satt nesten hele tiden sammen og arbeidet, med noen unntak. Vi har hele tiden hatt god hjelp fra hverandre om en står fast på et problem og har hatt god flyt av informasjon.
Arbeidet gikk ofte langt over avsatt tid, det var som regel overtid hver arbeidsdag.

5.2 VIDEREUTVIKLING AV PROSJEKTET

Prosjektet ved levering har bortimot all funksjonalitet klar i backend og trenger kun å sammenkobles. Versjonen har delvis funksjonalitet med unntak som blir listet under.

5.2.1 Statistikk

Det er et diagram klar til å ta imot data, men problemer i slutten av siste iterasjon gjorde det ikke mulig å få disse sammenkoblet. Det er derfor bare et eksempel av et diagram som er synlig. Vi har planer om å legge inn enda mer statistikk slik at et kollektiv kan få full oversikt over hvor mye som har blitt gjort. Vi skal også legge til grafer for perioder

hvor en enkelt kan lete opp hvor mye som har blitt gjort av hver person etter at en ny person flyttet inn i kollektivet.

5.2.2 Forbedring av Brukergrensesnitt

Utseende til applikasjonen har stort forbedringspotensialet, ingen av oss var spesielt erfarne innen grafisk design noe som vises ved enkelt design. Samtidig er vi fornøyd med det vi har laget og tror at brukergrensesnittet gjør jobben den skal.

5.2.3 Fullføring av oppgaver

I backend har vi allerede klart hvordan oppgaver skal håndteres når de er fullførte, men vi fikk aldri knyttet en sammenheng I React. Planen fremover er å ha en knapp for å fullføre noe som tildeler poeng til ansvarlig og putter den i en ny liste for fullførte oppgaver.

5.2.4 Flere Kollektiv

Tidlig i oppgaven var planen å gjøre det mulig å være medlem av flere kollektiv, men vi fant ut det var en nedprioritert del av prosjektet. De fleste vil nok kun ha et kollektiv siden det først og fremst er for ditt bofellesskap.

5.3 ARBEIDSFORDELING

Gruppen ble delt inn i backend og frontend, med unntak og samarbeid på deler av disse.

10005:

- Backend - User service og Kollektiv service
- Frontend - Noen funksjoner for henting av data, JavaScript

10004:

- Backend - Email service
- Frontend - Strukturen til Webapplikasjon, css, html og JavaScript
- Server - Konfigurasjon og drifting av webapplikasjonen på server

6. Konklusjon

Endemålet med oppgaven var å utvikle en webapplikasjon som var simpel og brukervennlig, applikasjonen skal kunne:

- Hente data fra backend
- Sende data til backend
- Fungere på mobil og web
- Være så simpel som mulig for brukeren
- Forenkle prosessen ved å håndtere gjøremål i kollektiv

Vår webapplikasjon fyller alle kravene til varierende grader, vi henter og sender all nødvendig data, men fikk ikke et sluttresultat som viser frem all informasjonen. Problemstillingen var å lage et produkt som var både brukervennlig, lett forståelig, med et responsivt design.
I rapporten blir problemstillingen oppnådd i varierende grad av resultatene vi oppnådde. Vi har laget en applikasjon som:

- Det som er på plass er responsivt
- Klar for videre utvikling ved kall til backend
- Kan brukes på mobil og web
- Kan legge til oppgaver
- Kan forlate eller bli med i kollektiver
- Henter og sender informasjon til og fra backend.

Sluttresultatet er vi ikke fornøyd med. Vi mangler deler av frontend som gjør at applikasjonen ikke er klar for utrulling. Vi ønsker senere å fullføre applikasjonen og legge den ut «play store».
Grunnen til at ting ikke gikk så raskt som planlagt var fordi vi ofte satt fast med problemer vi ikke har sett før. Problemene kunne fort vare opptil en uke, noe som er alt for mye tid å bruke på en enkelt funksjon eller komponent. Hadde vi gjort dette en gang til ville det tatt mye kortere tid.

Det vi er fornøyd med er det vi er ferdig med som:

- Rest API med Django
- Responsivt design med React
- Ferdig konfigurert server

En ting vi vet sikkert etter å ha fullført dette prosjektet er at kode tar tid.

7. Referanser

7.1 Internett

Chris Mills (i.d) What is JavaScript

URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript(03.11.2018)

Dave Kuhlman (07.08.2008) Python 101

URL:https://web.archive.org/web/20120719050408/http://cutter.rexx.com/~dkuhlman/python_101/python_101.html (03.11.2018)

Hovhannes Avoyan (31.10.2017), Linux versus Windows: OS impact on uptime and speed, URL: http://www.monitis.com/blog/linux-versus-windows-os-impact-on-uptime-and-speed/ (14.10.2018)

Ingen forfatter (2018), Mobile operating system market share worldwide, Statcounter Globalstats URL: http://gs.statcounter.com/os-market-share/mobile/worldwide (12.10.2018)

Ingen forfatter (2018), Mobile operating system market share worldwide, Statcounter globalstatus URL: http://gs.statcounter.com/os-market-share/mobile/norway/2018 (12.10.2018)

Ingen forfatter (Mars 2016), ReactJS vs Angular5 vs Vue.js, medium.com

https://medium.com/@TechMagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d (09.10.2018)

Ingen forfatter (2017), Share of individuals with iOS and Android phones in Norway 2017, Statista URL:https://www.statista.com/statistics/875407/share-of-individuals-with-ios-and-android-phones-in-norway/ (26.10.2018)

Ingen forfatter (2017), Smartphone OS market share, IDC, URL: https://www.idc.com/promo/smartphone-market-share/os (26.10.2018)

Ingen forfatter (2018), Mobile operating system market share worldwide, Statcounter Globalstats URL: http://gs.statcounter.com/os-market-share/mobile/worldwide (12.10.2018)

Ingen forfatter (2018), Mobile operating system market share worldwide, Statcounter globalstatus URL: http://gs.statcounter.com/os-market-share/mobile/norway/2018 (12.10.2018)

Ingen forfatter (Mars 2016), ReactJS vs Angular5 vs Vue.js, medium.com

https://medium.com/@TechMagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d (22.10.2018)

Ingen forfatter (i.d), The Gestalt Principles URL: http://graphicdesign.spokanefalls.edu/tutorials/process/gestaltprinciples/gestaltprinc.htm (2018)

Ingen forfatter (i.d), The Ubuntu lifecycle and release cadence, URL: https://www.ubuntu.com/about/release-cycle (13.11.2018)

Ingen forfatter (2009), What is the longest uptime you have ever seen?, URL: https://www.reddit.com/r/linux/comments/r411j/what_is_the_longest_uptime_you_have_ever_seen/ (28.10.2018)

Ingen forfatter(i.d), Installation/SystemRequirements, URL: https://help.ubuntu.com/community/Installation/SystemRequirements (13.10.2018)

Ingen forfatter (i.d), Microservices

URL:https://microservices.io/ (01.11.2018)

Ingen forfatter (2017), Share of individuals with iOS and Android phones in Norway 2017, Statista URL:https://www.statista.com/statistics/875407/share-of-individuals-with-ios-and-android-phones-in-norway/ (26.10.2018)

Ingen forfatter (2017), Smartphone OS market share, IDC, URL:https://www.idc.com/promo/smartphone-market-share/os (26.10.2018)

Ingen forfatter (i.d) Tool Recommendations

URL: https://packaging.python.org/guides/tool-recommendations/ (03.11.2018)

Ingen forfatter (i.d) Django

URL: https://tutorial.djangogirls.org/en/django/ (03.11.2018)

Ingen forfatter (i.d) HTML

URL: https://developer.mozilla.org/en-US/docs/Web/HTML (03.11.2018)

Ingen forfatter (i.d) Intro SQL

URL: http://www.sqlcourse.com/intro.html (03.11.2018)

Ingen forfatter (i.d) CORS

URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (02.11.2018)

Ingen forfatter (August 2018) Web Server Survey August 2018

URL: https://news.netcraft.com/archives/2018/08/24/august-2018-web-server-survey.html (03.11.2018)

Ingen forfatter (2018), The Gestalt principles, Spokane falls community college URL: http://graphicdesign.spokanefalls.edu/tutorials/process/gestaltprinciples/gestaltprinc.htm

Ingen forfatter (2018), Terms of service, Icon Archive URL: http://www.iconarchive.com/terms.html (02.10.2018)

John Hannah (05.02.2018), JavaScript Frameworks by the Numbers, Javascriptreport

URL: https://javascriptreport.com/javascript-frameworks-by-the-numbers-winter-2018/ (22.10.2018)

Kjetil Sander, ansvarlig redaktør (28/01/2018), Prosjektmodell, estudie

URL: https://estudie.no/prosjektmodell/ (02.10.2018)

Kutlu Sahin (26.01.2017), 7 Reasons why you should use React, Jotform.com

URL: https://stories.jotform.com/7-reasons-why-you-should-use-react-ad420c634247(04.10.18)

Ulf Eriksson (2013), Is Agile just Scrum? Reqtest

URL: https://reqtest.com/agile-blog/is-agile-just-scrum/ (02.10.2018)

7.2 Litteratur

Johnson, J. (2010) Designing with the Mind in Mind. 2. utg. Waltham, USA: Elsevier

Vedlegg

Vedlegg 1: Readme

For kjøring av prosjektet må du laste ned alle servicene først:

Webapplikasjon, Kollektiv service, Email Service og User service

De kjører på følgende porter

Webapplikasjon: 3000

Kollektiv service: 8083

Email Service: 8081

User service: 8082

I alle servicene må du passe på at du er i rett mappe, det kan hende du må endre det.

for eksempel i User service er startmappen user service du må da flytte deg inn til project: «cd project»

Før du kan starte servicene må du passe på at du har alle eksterne bilbiotek installer, dette kan du gjøre ved å skrive kommandoen pip install -r requirements.txt i alle servicene. Vi anbefaler å bruke et virtuelt miljø for å installere og kjøre koden. Informasjon om dette finnes her: https://www.pythonforbeginners.com/basics/how-to-use-python-virtualenv

I React må du bytte til webapplikasjon mappen og kjøre kommandoen npm install.

Når du har byttet mappe og installert alt i requirements.txt kan du starte servicene, under er listen på hvordan du skal starte hver service:

User service: py manage.py runserver 8082

Kollektiv service: py manage.py runserver 8083

mailService: py manage.py runserver 8081

Webapplikasjon: npm start

Servicene skal nå være oppe og går og du kan sjekke ut prosjektet på http://localhost:3000.

Vedlegg 2: Kildekode

Android app: https://gitlab.com/kollektivet/androidApp.git

Email service: https://gitlab.com/kollektivet/mailService.git

Kollektiv service: https://gitlab.com/kollektivet/kollektivservice.git

User service: https://gitlab.com/kollektivet/userservice.git

Webapplikasjon: https://gitlab.com/kollektivet/WebApp.git