Norwegian University of
Science and Technology

Reviewsers

Forfattere
Haakon Reiss-Jacobsen
Håkon Legernæs
Maciej Piatkowski

Bachelor i programvareutvikling
20 ECTS
Institutt for Datateknikk og Informatikk
Norges teknisk-naturvitenskapelige universitet,

16.05.2018

Veileder Øivind Kolloen

Sammendrag av Bacheloroppgaven

Tittel:	**Reviewsers**
Dato:	16.05.2018
Deltakere:	Haakon Reiss-Jacobsen Håkon Legernæs Maciej Piatkowski
Veiledere:	Øivind Kolloen
Oppdragsgiver:	Pappagallo Partners AS
Kontaktperson:	Øivind Kolloen, oeivind.kolloen@ntnu.no, 61135218
Nøkkelord:	Android, Java, JavaScript, REST, API, Backend
Antall sider:	78
Antall vedlegg:	7
Tilgjengelighet:	Åpen

Sammendrag:	Prosjektet går ut på å lage et forbrukerdrevet sosialt nettverk, hvor brukere kan lese og skrive anmeldelser for konsumentprodukter. Under arbeidet utarbeidet vi en ny måte å strukturere anmeldelser på, kalt ”Progressive Review System”. Løsningen består av en frontend og backend del: en androidapplikasjon, webapplikasjon og et API. Vi har i prosjektet hatt fokus på bruk av smidig utviklingsmetodikk gjennom bruken av Scrum rammeverket.

Summary of Graduate Project

Title:	**Reviewsers**
Date:	16.05.2018
Authors:	Haakon Reiss-Jacobsen Håkon Legernæs Maciej Piatkowski
Supervisor:	Øivind Kolloen
Employer:	Pappagallo Partners AS
Contact Person:	Øivind Kolloen, oeivind.kolloen@ntnu.no, 61135218
Keywords:	Android, Java, JavaScript, REST, API, Backend
Pages:	78
Attachments:	7
Availability:	Open

Abstract: This project is about creating a consumer-driven social network, where users can read and write reviews for consumer products. During the project we created a new way to structure reviews, called ”Progressive Review System”. The project consists of a front-end and back-end part: an Android application, Web application and an API. We have focused on the use of agile development throug the use of the Scrum framework.

Forord

For det første ønsker vi å takke vår veileder Øivind Kolloen for god oppfølging og tilbakemelding i løpet av prosjektet. Vi vil også takke vår oppdragsgiver, Per og Lars fra Reviewsers AS, for sitt engasjementet og den tette oppfølgingen. Deres innspill og ressurser var veldig nyttig under utviklingen av prosjektet. Til slutt vil vi også takke Simon McCallum for innspill til anmeldelsessystemet vi utviklet i løpet av prosjektet, og alle deltakere av brukertestingen som ga oss konstruktiv kritikk og innspill.

Innhold

Forord . . . . . **iii**
Innhold . . . . . **iv**
Figurer . . . . . **viii**
Tabeller . . . . . **x**
List of Listings . . . . . **xi**
Ordliste . . . . . **xii**
1 Introduksjon **1**
1.1 Bakgrunn 1
1.1.1 Reviewsers 1
1.1.2 Hva vi skal levere 2
1.2 Brukeranmeldelser 3
1.2.1 Diskusjon rundt anmeldelser 3
1.2.2 Progressive Review System 4
1.3 Rammer 4
1.4 Avgrensning 5
1.4.1 Mobil 5
1.4.2 Webapplikasjon 5
1.4.3 Produkter 5
1.4.4 Internasjonalisering 5
1.5 Målgrupper 5
1.5.1 Mobilapplikasjonen 5
1.5.2 Webapplikasjonen 6
1.5.3 Rapport 6
1.6 Hvorfor vi valgte oppgaven 6
1.7 Gruppemedlemmene 6
1.8 Øvrige roller 6
1.8.1 Rapport 7
2 Utviklingsprosess **8**
2.1 Utviklingsmodell 8
2.1.1 Alternativer 8
2.1.2 Argumentasjon 9
2.1.3 Scrum utvidelser 9
2.2 Gjennomføring 10
2.2.1 Arbeidsplan 10
2.2.2 Planlegging av prosjektet 11

2.2.3 Møter . . . 11
2.2.4 Arbeidsmetodikk . . . 12
2.2.5 Problemer under utvikling . . . 13
2.3 Sprintoversikt . . . 13
2.3.1 Arbeid før sprint 1 (15. januar - 23. januar) . . . 13
2.3.2 Sprint 1 (23. januar - 5. februar) . . . 13
2.3.3 Sprint 2 (5. februar - 20. februar) . . . 14
2.3.4 Sprint 3 (20. februar - 5. mars) . . . 14
2.3.5 Sprint 4 (5. mars - 19. mars) . . . 14
2.3.6 Sprint 5 (19. mars - 16. april) . . . 14
2.3.7 Sprint 6 (16. april - 23. april) . . . 15
2.3.8 Sprint 7 (23. april - 30. april) . . . 15
2.3.9 Sprint 8 (30. april - 7. mai) . . . 15
2.3.10 Sprint 9 (7. mai - 16. mai) . . . 15
3 Kravspesifikasjon . . . 16
3.1 Funksjonelle krav . . . 16
3.1.1 Use-Case Diagram . . . 16
3.1.2 Høy-Nivå Use-Case . . . 17
3.1.3 Utvidet Use-Case . . . 20
3.1.4 Domenemodell . . . 20
3.2 Øvrige krav . . . 21
3.2.1 Produktdatabase . . . 21
3.2.2 Internasjonalisering . . . 23
3.2.3 Sikkerhet og personvern . . . 23
4 Valg av teknologi . . . 25
4.1 Android SDK . . . 25
4.2 Backend . . . 25
4.2.1 Node.js . . . 25
4.2.2 Express . . . 26
4.2.3 Database . . . 26
4.3 Webapplikasjonen . . . 27
4.3.1 JQuery . . . 27
4.4 Technical memo: Valg av produktkatalog tilbyder . . . 27
4.4.1 Bakgrunn . . . 27
4.4.2 Diskusjon . . . 27
4.4.3 Konklusjon . . . 28
4.5 Technical memo: Android eller kryssplattform . . . 28
4.5.1 Bakgrunn . . . 28
4.5.2 Diskusjon . . . 28
4.5.3 Konklusjon . . . 29

5 Design . . . 30
5.1 Arkitektur . . . 30
5.2 Backend . . . 30
5.2.1 Valg av arkitektur . . . 30
5.2.2 REST-arkitekturen . . . 31
5.2.3 Database . . . 31
5.3 Android . . . 33
5.3.1 MVC . . . 33
5.4 Progressive Review System . . . 36
5.4.1 Utgangspunkt . . . 36
5.4.2 Hvordan PRS strukturerer en anmeldelse . . . 37
5.4.3 Hvordan PRS viser fram anmeldelser . . . 39
5.5 Brukergrensesnitt . . . 40
5.5.1 Fragments og Activity . . . 40
5.5.2 Brukergrensesnitt Android . . . 41
6 Implementasjon . . . 44
6.1 Utviklingsmiljø . . . 44
6.1.1 Android . . . 45
6.1.2 Backend . . . 45
6.2 Backend . . . 46
6.2.1 Import av produktkatalog . . . 46
6.2.2 Kommunuikasjon med klient . . . 47
6.2.3 Kommunikasjonspipeline . . . 48
6.2.4 Produktsøk . . . 53
6.2.5 Paginering . . . 54
6.3 Android . . . 55
6.3.1 HomeActivity . . . 55
6.3.2 ApiRequest . . . 56
6.3.3 Filtrering av anmeldelser . . . 58
6.3.4 Visning av bilder . . . 59
6.3.5 Dekoding av strekkoder . . . 60
6.3.6 Legge til nye produkter . . . 60
6.4 Webapplikasjon . . . 62
6.4.1 Navigasjon . . . 62
6.4.2 Anmeldelser . . . 62
7 Testing og kvalitetssikring . . . 66
7.1 Kodekvalitet . . . 66
7.1.1 Sonarqube . . . 66
7.2 Unit tester . . . 67
7.3 Dokumentasjon av API . . . 68

7.4 Brukertesting . . . 69
7.4.1 Tilbakemelding . . . 70
7.4.2 Forandringer som følge av brukertesting . . . 70
8 Utrulling . . . 72
8.1 Backend server . . . 72
8.1.1 Nødvendig programvare . . . 72
8.1.2 Pm2 . . . 72
8.1.3 Sikkerhet . . . 73
8.1.4 Anbefalning . . . 73
8.2 Mobilapplikasjon . . . 73
8.2.1 Anbefalning . . . 74
8.2.2 Webløsningen . . . 74
9 Konklusjon . . . 75
9.1 Drøftinger . . . 75
9.1.1 Læringsmål . . . 75
9.1.2 Resultatmål . . . 75
9.2 Alternative valg . . . 76
9.3 Framtidig Arbeid . . . 76
9.4 Evaluering av arbeid . . . 77
9.5 Avslutning . . . 78
Bibliografi . . . 79
A Prosjektplan . . . 86
B Prosjektavtale . . . 108
C Kode . . . 112
D Gradle fil . . . 114
E Notater . . . 116
F Spørsmål til brukertesting . . . 118
G Bilder av androidapplikasjonen . . . 119
H Bilder av webapplikasjon . . . 125
I Gjennomgang av sprinter under utvikling . . . 126

Figurer

1 Eksempel på strukturen til en tradisjonell brukeranmeldelse (fra Amazon.com [1]) . . . 3

2 Eksempel på Kanban i bruk[2] . . . 9

3 Gjennomgang av sprint 1 . . . 12

4 Gjennomgang av sprint 4 . . . 12

5 Use case diagram . . . 17

6 Domenemodell for Reviewsers . . . 21

7 Statistikk over Android plattformdistribusjon per 16. April 2018[3] . . . 22

8 Arkitektur for Reviewsers . . . 30

9 Databasemodell for Reviewsers . . . 32

10 Oversikt over MVC-arkitekturen . . . 33

11 Utkast fra oppdragsgiver for design av anmeldelser . . . 36

12 Progressive Review System strukturering av anmeldelser. Bilder i modellen er tatt ifra web-applikasjonen . . . 37

13 Delanmeldelse. Bilde tatt ifra web-applikasjonen . . . 38

14 Hvordan PRS viser fram anmeldelser på mobil . . . 39

15 Bilder fra androidapplikasjonen . . . 41

16 Bilder fra androidapplikasjonen . . . 42

17 Viser pipeline for en request til API . . . 48

18 Visning av anmeldelsestagger . . . 64

19 API-dokumentasjon med Swagger.io `https://app.swaggerhub.com/apis/Reviewsers/Reviewsers/1.0.0` . . . 69

20 Utvalgte spørsmål fra brukertesting . . . 70

21 Forandringer gjort som følge av tilbakemeldinger . . . 71

22 Bilder fra androidapplikasjonen . . . 119

23 Bilder fra androidapplikasjonen . . . 120

24 Bilder fra androidapplikasjonen . . . 120

25 Bilder fra androidapplikasjonen . . . 121

26 Bilder fra androidapplikasjonen . . . 121

27 Bilder fra androidapplikasjonen . . . 122

28 Bilder fra androidapplikasjonen . . . 122

29 Bilder fra androidapplikasjonen . . . 123

30 Nytt produkt 2 . . . 123

31 Nytt produkt 4 124
32 Login web 125
33 PRS web 125
34 Product View Web 125
35 Sprint 1 126
36 Sprint 2 126
37 Sprint 3 126
38 Sprint 4 127
39 Sprint 5 127

Tabeller

1 Tidstabell for møter 10
2 Oppgaver i Sprintene 13
3 Sammenligning av CRUD operasjoner i HTTP og SQL 31
4 Sonarqube rapporter for Android 67

List of Listings

1 Utdrag fra scriptet som importerer produktkatalogen fra Icecat 46
2 Eksempel på respons fra API 47
3 Utdrag fra filen *routes.js* 49
4 Håndhever autentisering i *routes.js* 50
5 Funksjon som autentiserer en JWT-token 50
6 Utdrag fra funksjonen *verifyToken* 51
7 Utdrag fra filen *validator.js* 51
8 Utdrag fra filen *routes.js* 52
9 Funksjonen *getProductByID* 53
10 Funksjonen *paginate* 55
11 Funksjonen *showContent* 56
12 Viser en forespørsel til API før refaktorering 57
13 Viser en forespørsel til API etter refaktorering 58
14 Funksjonen *filterReviews* 59
15 Hvordan bilder ble hentet inn før refaktorering 60
16 Hvordan bilder ble hentet inn etter refaktorering 60
17 Hvordan vi bruker kamera i Android API'et 61
18 Utdrag fra funksjonen loadPage 62
19 Utdrag fra funksjonen addNewSubreview 63
20 Funksjonen *updateShownTags* 64
21 Utdrag fra scriptet som importerer databasen 113

Ordliste

API eller Application Programmable Interface er verktøy som tilbyr funksjonalitet for andre programmer. Et API inneholder metoder utviklere kan kalle fra sin egen kode[4]. viii, xi, xii, 16, 22, 24–28, 30, 31, 46–48, 50, 51, 56, 61, 68, 69, 72, 76

backend er den delen av programvareløsningen som har direkte tilgang til og er nærmest databasen. Blir ofte brukt som en betegnelse på serverdelen av programvaren[5]. xii, 1, 2, 4, 6, 7, 17–20, 22–25, 30, 45, 47, 48, 52, 53, 56, 66, 72–74

callback er funksjoner (eksekverbar kode) som sendes til annen kode, hvor det forventes at funksjonen på et senere tidspunkt skal kalles av koden som mottar denne funksjonen[6]. 46, 48, 49, 52, 56, 57

CRUD er en forkortelse for *Create, Read, Update, Delete*, og er fire standardoperasjoner som et webbasert API bør støtte for å være komplett[7]. x, 31

EAN-13 eller European Article Numbering er strekkoder som brukes til å merke produkter verden over. Består av 13 sifre med koder for land, produsent, varetype[8]. 16, 18, 21, 27, 46

GANTT Et Gantt-skjema er et type søylediagram som illustrerer et prosjekts tidsplan. Gantt-skjemaet illustrerer datoer for start- og sluttidspunkt for oppgavene i prosjektet og viser et sammendrag av prosjektet[9]. 11

Icecat er en tilbyder av produktkataloger og produktinformasjon. Har millioner av produkter i sine kataloger og brukes blant annet av nettbutikker[10]. xi, 27, 28, 32, 45, 46, 54, 60, 61, 73, 77

native Begrepet native applikasjon beskriver en applikasjon som kjører direkte på en enhetens operativsystem. 28

open-source eller på norsk ”åpen kildekode” vil si at programmets kildekode er gjort fritt tilgjengelig for alle[11]. 22, 25, 44–46, 50, 51, 72

pipeline er i denne sammenhengen en serie distinkte prosedyrer en forespørsel til backend går gjennom en etter en. 48–50, 52, 53

PRS eller ”Progressive Review System” er betegnelsen på det nye anmeldelsessystemet vi har designet for løsningen. En lengre forklaring på systemet finnes i seksjon 5.4. 4–7, 14, 29, 36–40, 42, 69–71, 75, 76

SDK står for *Software Development Kit* og er et sett med verktøy for utvikling av programvare for en bestemt løsning[12]. 22, 25

SSH står for *Secure Shell* og er en nettverksprotokoll som lar en administrator aksessere kommandolinjen på en server gjennom en kryptert tilkobling[13]. 45

SSL eller *Secure Socket Layer*, er en nettverksprotokoll for å kryptere kommunikasjonen mellom klient og server[14]. 24, 73

token er et begrep vi bruker når vi snakker om brukerautentisering. En *token* er en tidsbegrenset ”billett” brukeren får når dem først logger inn. Denne brukes så for autentisering i påfølgende innlogginger, slik at brukeren slipper å skrive inn e-post og passord hver gang. 18, 24, 30, 49, 56

1 Introduksjon

1.1 Bakgrunn

I februar 2017 var en av våre oppdragsgivere med på et seminar som omhandlet framtiden for digitale teknologier og trender i markedet. En av foredragsholderne snakket om en ny forretningsmodell kalt *influence-marketing* og hvordan den vil kunne endre verden. Ideen bak modellen er å bruke sosiale medier for å skape et forhold mellom forbruker, *influencers*, og markedsførere. En *influencer* kan være hvem som helst med kraften til å påvirke forbrukeren. Dette kan være alt fra en blogger, kjendis eller en populær atlet.

I samme periode hadde oppdragsgiverne vært i utlandet hvor de hadde blitt presset av forskjellige hotelleiere til å legge igjen gode anmeldelser på forskjellige reisesider. En ansatt ved resepsjonen hadde sagt at mange positive anmeldelser var lagt ut av ansatte, og at dette var vanlig praksis blant hotellene.

Basert på disse erfaringene diskuterte de sammen hvordan den nye modellen ville kunne påvirke verdikjeden[15] som er beskrevet og popularisert av økonomen Michael Porter[16]. Dette ettersom de fleste bedrifter bruker verdikjeden aktivt for å skape konkurransedyktige fordeler. En av ulempene med verdikjeden er at den tar lite hensyn til forbrukeren og fokuserer mer på selve bedriften og dens struktur.

Det er denne diskusjonen som utviklet seg til ideen ved å snu verdikjeden opp ned å sette brukeren i styringen av prosessen. Ideen var å skape et sosialt nettverk for anmeldelser, hvor alle anmeldelser skrives og leses på samme sted. Dette vil gjøre det lettere for brukeren å bli informert rundt et kjøp, basert på andre brukeres erfaringer, imotsetting til *influencer'e* eller falske anmeldelser.

En annen stor inspirasjon for denne ideen er applikasjonen *Vivino*, hvor brukere av applikasjonen kan ta bilde av en vinflaske, og få i retur brukeranmeldelser om vinen[17]. Dette var en svært suksessfull applikasjon, og oppdragsgiveren ønsket en mer generell løsning for flere typer konsumentprodukter.

Oppdragsgiver opprettet så bedriften *Pappagallo Partners AS*[18] og kom fram til at oppgaven passet godt som en bacheloroppgave. De kontaktet NTNU i Gjøvik hvor de foreslo oppgaven. Bedriften bytter nå navn til *Reviewsers AS*, og vi vil referere til selskapet ved det nye navnet i resten av rapporten.

1.1.1 Reviewsers

Reviewsers er ideen om et sosialt nettverk for anmeldelser for alle type konsumentvarer. Dette vil bety et brukersamfunn av anmeldere hvor alle anmeldelser finnes på ett sted. Tanken er at brukeren skal kunne lese, skrive og dele sine anmeldelser i brukersamfunnet. Applikasjonen skal utvikles for web og mobil, hvor en egen backend med produktdatabase og tilhørende anmeldelser vil legges til. Brukeren vil kunne finne fram til produkter ved å gjøre et tekstsøk, hvor man på mobil også kan skanne produktets strekkode. Etter søk eller skanning skal en produktside vises fram hvor anmeldelser kan vises og skrives. Hvis et produkt ikke ligger inne i databasen skal brukeren ha muligheten

til å legge til produktet selv.

Det ferdige produktet er ment til å sikte seg inn på det internasjonale markedet.

1.1.2 Hva vi skal levere

Vi skal levere en mobilapplikasjon som vil fungere som en prototype for applikasjonen oppdragsgiver har beskrevet. Applikasjonen vil utvikles for Android med en mulig iOS versjon etter oppgaven er levert. Samtidig skal vi utvikle en webapplikasjon som vil brukes til å demonstrere et nytt anmeldelsessystem kalt *Progressive Review System* (se seksjon 5.4). Dette er et egetutviklet system vi selv kom opp med før utarbeidelse av prosjektplanen. Etter implementasjon i webløsningen skal systemet implementeres også i Android applikasjonen. De to applikasjonene vil kobles opp mot en egenutviklet backendløsning, med tilhørende database hvor all data vil lagres. Oppgaven går også ut på å finne en tilbyder av produktkataloger som kan brukes i vår egen produktdatabase. Databasen skal kunne utvides ved at andre brukere legger inn produkter.

Funksjonalitet androidapplikasjon

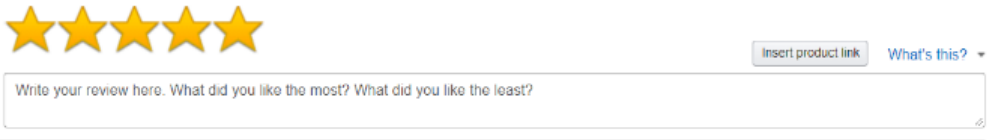
- Bruker skal kunne søke etter et produkt
- Skrive og lese anmeldelser for forskjellige produkter
- Legge til nye produkter i databasen
- Skanne en strekkode
- Dele applikasjon eller anmeldelser med andre utenfor applikasjonen
- Se alle sine anmeldelser
- Lese alle anmeldelser fra en annen bruker
    - Hvis produktet finnes i databasen, vil en produktside med tilhørende anmeldelser vises
    - Hvis produktet ikke finnes i databasen, vil brukeren ha muligheten for å legge til dette produktet, eller legge til strekkoden til et produkt som allerede finnes (et produkt kan ha flere strekkoder)
- Bruker må registrere en bruker for å kunne bruke applikasjonen
- Applikasjonen skal ha støtte for å legge inn flere språk

Funksjonalitet webapplikasjon

- Et system for å kunne legge inn anmeldelser med vårt nye system
- Kunne vise en produktside for å vise fram hvordan anmeldelser vil se ut i en ferdig applikasjon
- Ha et enkelt system for å legge til ny bruker og logge inn for å bruke web applikasjonen
- Webapplikasjon skal være laget for å kunne videreutvikles til en ferdig applikasjon, hvor de funksjonelle kravene kommer til å være det samme som mobilapplikasjonen

Implementere vår ide for et anmeldelsessystem som passer for løsningen

1.2 Brukeranmeldelser



Figur 1: Eksempel på strukturen til en tradisjonell brukeranmeldelse (fra Amazon.com [1])

En brukeranmeldelse har i dag en veldefinert struktur. Denne strukturen består som regel av en vurdering hvor det blir brukt en tallverdi sammen med en skrevet del (se figur 1). Dette vil til sammen utgjøre en anmeldelse. Eksempler på slike verdier kan være en til fem stjerner, et tall fra en til ti eller et terningkast o.l. hvor jo høyere verdien er jo bedre vurdering har anmeldelsen. Den skrevne delen av anmeldelsen består som regel kun av en tekstboks, hvor anmelderen kan skrive begrunnelse for vurderingen og gi en forklaring om hva anmelder mener om produktet. Det finnes små variasjoner av denne strukturen. Spesifikt er det vanlig at mer spesialiserte steder (f.eks. en elektronikkbutikk) lar deg gi en vurdering på noen av de generelle aspektene for produktene de selger. Brukeranmeldelser av produkter skrives som regel via butikken produktet ble kjøpt hos. Anmeldelsene skrives som regel på butikkens nettside, hvor noen krever at brukeren må ha en registrert kvittering som bekrefter kjøp.

Det vil være feil å si at denne formen for brukeranmeldelser representerer alle utbredte anmeldelsesstrukturer, men vi mener det gir et godt bilde for systemet de fleste er vant med. Det gir oss også en god representasjon for hva vi kan gå ut ifra når vi selv skal implementere vårt eget anmeldelsessystem.

1.2.1 Diskusjon rundt anmeldelser

Første gang gruppen møtte med oppdragsgiver, var anmeldelser noe som sto i sentrum. Det ble diskutert en god del problemer vi og oppdragsgiver hadde erfart. Det kom raskt fram at at oppdragsgiver delte våre syn på viktigheten i å få fram problemene med den ordinære strukturingen av anmeldelser. Ut ifra disse diskusjonene valgte vi å ta en nærmere analyse av anmeldelser og strukturen vi har diskutert, før vi utarbeidet en prosjektplan.

- Ved å bruke et verdisystem som stjerner eller liknende til vurdering skaper man et subjektivt system [1]. Det har blitt vist at hva forskjellige forbrukere forbinder med forskjellige verdier er forskjellig. En person vil kunne si at så lenge det ikke er noe direkte galt med produktet vil det bety at produktet er tilnærmet perfekt. En annen vil derimot kalle det gjennomsnittlig. Begge har samme oppfatning av de forskjellige aspektene av produktet men har en forskjellig konklusjon når du skal gi en poengsum som vurdering
- Vanskelig å skille vurdering ut ifra poeng og tekst. Hvor det ofte kun fokuseres på den førstnevnte
- Det er lite til ingen struktur i tradisjonelle anmeldelser, hvor brukere ofte lager egne stukturer i den skriftlige delen

- På grunn av at anmeldelser ikke blir delt mellom nettsidene de skrives på vil det ofte være begrenset for hvor mange anmeldelser hver bedrift vil ha. Dette fører til at det i mange tilfeller ikke er tilstrekkelig med brukerinformasjon for forbrukeren til å ta et informert valg om kjøp av produkt
- Falske anmeldelser, lagt inn av produsentens egne ansatte

Gjennomgående i rapporten vil vi komme tilbake til den tradisjonelle anmeldelsesstrukturen, hvor det vil drøftes rundt problemene denne strukturen har og hvordan vi har valgt å løse det.

1.2.2 Progressive Review System

Ideen om en ny type anmeldelsessystem ble et tema tidlig i planleggingsfasen. Ettersom brukeranmeldelser var en sentral del av oppgaven valgte vi å ta en større analyse av anmeldelser, hvilke strukturer som ble brukt for å anmelde og problemene de medfører. Det vi kom fram til ble et utkast for et alternativt anmeldelsessystem.

Vi tok utgangspunkt i vår analyse av brukeranmeldelser og problemstillingen ut i fra oppgaven, som krever at vi skal lage et system for anmeldelser som skal kunne fungere for alle type produkter. Løsningen har fått navnet *Progressive Review System* (PRS). Dette er et anmeldelsessystem vi selv har kommet opp med, hvor vi også har navngitt løsningen. Oppdragsgiver ønsket seg en internasjonal løsning så vi valgte på grunnlag av dette å navngi systemet på engelsk. Navnet baserer seg på hvordan produkter over tid vil kunne defineres ut ifra dets anmeldelser. Dette ved å kunne beskrives ut ifra sine egne egenskaper og funksjoner. Denne oppdelingen vil gjøres ved introduksjonen av *anmeldelsestagger*. Når en bruker skriver en anmeldelse skal brukeren kunne dele den opp i delanmeldelser. Hver delanmeldelse er definert ut ifra en ”anmeldelsestagg”. Man kan tenke på en ”anmeldelsestagg” som ett aspekt eller egenskap ved produktet. Hvis man for eksempel snakker om en mobiltelefon kan en ”tagg” være *skjerm*, *batteri* eller *byggekvalitet*. Ettersom vi ikke har tilgang til de unike delene hvert produkt består av, lar vi brukere selv definere disse. Alle ”tagger” for et produkt vil defineres av brukerne selv og vil hjelpe med å få fram de unike delene av produktet. Vi snakker mer om PRS og hvordan vi kom fram til løsningen kan leses i kapittelet 5.4.

1.3 Rammer

Sammen med oppdragsgiver diskuterte vi rammene rundt prosjektet. To av punktene det var ekstra fokus på, var å støtte så mange androidtelefoner som mulig og en backendløsning som enkelt kunne flyttes over til oppdragsgiverens egen server. Støtte for så mange som mulig androidtelefoner krever at vi definerer et hensiktsmessig minimum *API-nivå* for applikasjonen (se seksjon 3.2.1. **Android**

- Applikasjonen må støtte så mange Android-versjoner som mulig, så lenge det er hensiktsmessig (se seksjon 3.2.1).

Backend

- Backend må kunne flyttes over til oppdragsgiver sin egen server etter oppgaveslutt

General Data Protection Regulation

- De nye lovene fra EU rundt behandling av brukerdata (GDPR) [19] må følges. I

vår applikasjon vil det si at en bruker må kunne slette all sin data vi har lagret om brukeren, og at vi bearbeider retningslinjene til applikasjonen så bruker lett forstår hvilken informasjon vi lagrer

1.4 Avgrensning

1.4.1 Mobil

Vi begrenser oss til å kun utvikle for Android. Oppdragsgiver hadde i utganspunktet ønske om både en Android og en iOS applikasjon. Vi ga dem to scenarier, enten kunne vi lage en mer komplett prototype for Android, eller vi kunne fokusere på å lage to mobilapplikasjoner for både Android og iOS med noe mindre funksjonalitet. Oppdragsgiver konkluderte med at de heller ønsket at vi fokuserte på å få en mest mulig komplett prototype av applikasjonen. Vi vil fokusere på å lage en prototype som inneholder all grunnleggende funksjonalitet. Det er da viktig at prototypen enkelt skal kunne videreutvikles.

1.4.2 Webapplikasjon

På grunn av begrenset tid valgte vi å kun implementere PRS i webapplikasjonen og finne en løsning for hvordan å best vise fra anmeldelsene på en produktside. Opprinnelig ønsket vi å lage en komplett webapplikasjon av samme skala som mobilapplikasjonen, men vi satt det heller som noe vi eventuelt kunne gjøre hvis vi hadde mer tid.

1.4.3 Produkter

Vi begrenser oss til produkter som selges i butikk. Dette ekskluderer matprodukter og liknende produkter som konsumeres i den spiselige forstand. Hovedsakelig tar denne beslutningen utgangspunkt i hvordan oppdragsgiver så for seg løsningen, men har også å gjøre med hvor lokale slike produkter kan være og den totale mengden unike produkter det vil tilføre.

1.4.4 Internasjonalisering

Den ferdige applikasjonen skal lanseres internasjonalt, det vil si at applikasjonen må støtte flere språk og at retningslinjer for behandling av brukerinformasjon må i noen tilfeller behandles på et land-til-land basis. Etter en diskusjon med oppdragsgiver kom vi fram til at vi i dette prosjektet vil begrense oss til å fokusere på å ligge innenfor de nye retningslinjene fra EU om behandling av brukerinformasjon1.3. For språk vil vi legge inn engelsk som språk, med støtte for å kunne legge inn flere.

1.5 Målgrupper

Prosjektets målgruppe kan deles inn i to grupper: brukere av applikasjonen, og lesere av denne rapporten.

1.5.1 Mobilapplikasjonen

Vår hovedmålgruppe er brukere av mobilapplikasjonen. Dette fordi i motsetning til webapplikasjonen, vil dette være en prototype som vil ivareta den grunnleggende funksjonaliteten den ferdige applikasjon skal ha. Målgruppen er alle personer som kjøper og anmelder produkter, men også personer som ikke allerede i dag leser anmeldelser.

1.5.2 Webapplikasjonen

Målgruppen for webapplikasjonen er personer som kan teste ut og gi tilbakemelding på PRS. Etter testing og viderutvikling blir målet å legge inn denne strukturen på mobilapplikasjonen.

1.5.3 Rapport

Vår målgruppe for rapporten er alle personer som vil vite hvordan vi jobbet med prosjektet fra et utviklingsstandpunkt og hvordan vi kom fram til de løsningene vi valgte. Dette vil først og fremst være sensor, oppdragsgiver, veileder og potensielle oppdragsgivere i framtiden.

1.6 Hvorfor vi valgte oppgaven

Da vi valgte oppgaven var det kun en i gruppen som hadde møtt oppdragsgiver fysisk og som hadde vist interesse for oppgaven. Via vår studieveileder fikk vi vite at vi alle søkte etter en trepersons gruppe, og vi bestemte oss for å slå oss sammen. Ut ifra de forskjellige oppgavene vi hadde sett på valgte vi ut Reviewsers.

Det er to viktige punkter til hvorfor vi valgte Reviewsers som oppgave. Oppgaven var spennende og nytenkende samtidig som den ville gi oss stor kreativ frihet til å komme opp med egne løsninger innenfor oppgaven. Det å ha noen harde rammer rundt prosjektet, men stor frihet innenfor oppgaven var veldig viktig for både oppdragsgiver og oss, og det var enighet i at det ville kunne skape det beste resultatet. Annet enn det syntes vi prosjektet passet godt som en bacheloroppgave, både med tanke på kompleksitet og tidsrom. Vi så det som realistisk å kunne lage en prototype basert på oppgavebeskrivelsen, hvor vi kunne bruke mye av vår eksisterende og ny kunnskap til å utarbeide en løsning.

1.7 Gruppemedlemmene

Alle gruppemedlemmene har gjennomgått studiet Bachelor i Programvareutvikling (BPU) og har derfor det samme utgangspunktet når det kommer til vår faglige kunnskap fra studiet. Her er det lite som skiller oss annet enn noen valgemner som ikke er relevant for løsningen. Håkon har fra egne prosjekter en god del erfaring med Linux og backendutvikling, men er også noe de andre har verdt igjennom via studiene.

Gjennom studiet har vi opparbeidet oss kunnskap relevant til oppgaven. Dette inkluderer systemutvikling, utvikling i programmeringspråkene Java og C++. Vi har i webutvikling lært oss HTML, CSS og JavaScript. Vi har brukt denne kunnskapen på å lage applikasjoner på web, mobil og Windows/Mac, hvor vi også har laget backendløsninger ved bruk av SQL og PHP. Vi mener vår utdanning gir et godt grunnlag for å utarbeide en god løsning for denne oppgaven.

1.8 Øvrige roller

Haakon R-J. ble valgt til prosjektleder og scrummaster. Denne beslutningen ble tatt på bakgrunn av at han hadde hatt mest direkte kontakt med oppdragsgiver og var den som hadde best oversikt over hva vi skulle lage. Senere i utviklingen ble han også referent i gruppemøter for å bedre dokumentere diskusjoner. Han hadde også hovedansvar for booking av rom til møter.

Håkon L. var hovedansvarlig for utvikling av backendløsningen, dette fordi han hadde mest erfaring med Linux-miljøet og serveroppsett. Mye av valgene underveis ble derimot gjort sammen, og utvikling av backend og frontend kjørte ofte parallelt, hvor begge avhengte av hverandre.

På webapplikasjonen jobbet Haakon R-J. med implementasjonen av PRS, mens Maciej jobbet med produktsiden hvor anmeldelser skulle vises fram. Alle gruppemedlemmer jobbet sammen på androidapplikasjonen, hvor vi delte opp arbeidet ut fra hvor vi var i utviklingsprosessen.

Vår veileder var Øivind Kolloen, som vi hadde faste møter med annenhver mandag. Øivind sin rolle har vært å gi oss råd og veilede oss i løpet av oppgaven. Han har gitt oss tilbakemeldinger og kommentarer på beslutninger vi har tatt og kommentert på utkast til rapporten.

Oppdragsgiveren for oppgaven vår var oppstartsselskapet Reviewsers, som har to ansatte. Gruppen hadde faste møter med selskapet på starten av hver sprint. Dette ble gjort annenhver mandag hvor vi da kunne definere hver sprint mer basert på tilbakemelding fra oppdragsgiver og veileder.

1.8.1 Rapport

I rapporten har vi valgt å bruke en del engelske begreper fra fagmiljøet vårt. Det er også noen ord som ikke oversettes like godt til norsk. Engelske ord og andre begreper vil ha en klikkbar link som vil føre til en ordliste som forklarer ordene. Liknende linker blir også brukt når det refereres til andre kapitler, kilder eller figurer.

Kildekoden av lagt til ved innlevering av rapporten som en ZIP-fil.

Struktur

Vi har valgt å strukturere rapporten slik at det blir en logisk sekvens, hvor vi bedre kan redgjøre for hvordan vi har jobbet, og hva vi faktisk har utviklet. Rapporten er delt opp i ni kapitler og omhandler følgende hovedelementer

- Utviklingsprosess og arbeidsmetodikk
- Design og implementasjon av løsningen
- Testing og dokumentasjon
- Oppsummering og konklusjon samt anbefaling for videre arbeid

2 Utviklingsprosess

Dette kapittelet beskriver utviklingsprosessen i løpet av prosjektet. Her vil vi gjøre rede for hvordan vi planla arbeidet ved start av oppgaven og hvordan vi har valgt å arbeide underveis.

2.1 Utviklingsmodell

Det finnes mange måter å jobbe på, og enda flere måter å fordele arbeidet som skal gjøres. En systemutviklingsmetode er et rammeverk med forskjellige verktøy som har som oppgave å strukturere en utviklingsprosess [20]. I programvareutviklingsfeltet finnes det et bredt spekter av slike rammeverk. Noen er mer fast strukturerte mens andre gir mer frihet til utviklere. For utvikling i en bacheloroppgave som ikke er helt definert kan det være risikabelt å velge en utviklingsmetode som går hardt steg for steg gjennom utviklingen som f.eks *Rational Unified Process*[21]. Dette på grunn av at hvis det dukker opp problemer midt i utviklingen vil det være vanskelig å gjøre forandringer i en mer rigid utviklingsmodell. Det er derfor bedre å velge en mer smidig modell som gjør det lettere å løse problemer som dukker opp underveis, men som også vil kunne gi oss en god struktur å jobbe etter. To av de store utviklingsmetodene som vi valgte å se på er Scrum[22] og Kanban[23].

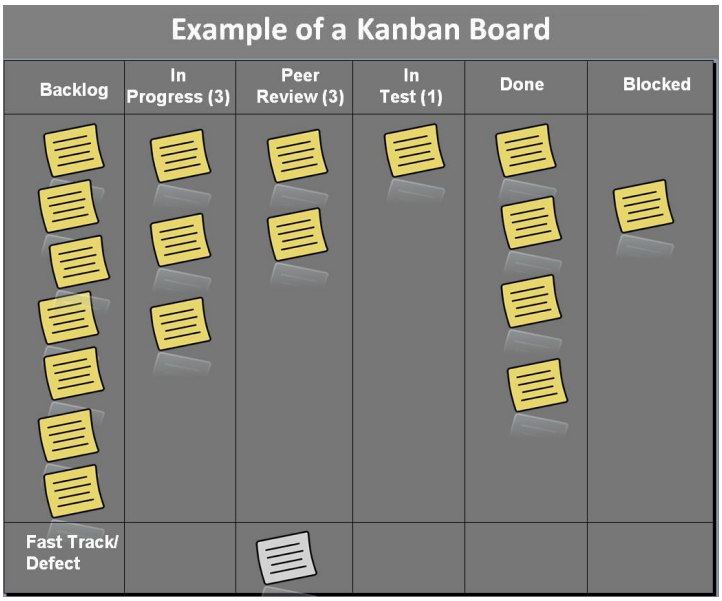
2.1.1 Alternativer

Den første systemutviklingsmetoden vi vurderte å bruke i prosjektet vårt er Scrum. Utenom noen grunnprinsipper i metoden som faste sprinter er Scrum veldig rik i forskjellige utvidelser man kan velge å ta i bruk. Det er for eksempel daglige møter, forskjellige måter for gruppeorganisering, eller løsninger for å håndtere sosiale problemer på arbeidsplassen, med for eksempel ”Snack Shrine”[24]. I prosjektet var det mest relevant med daglige møter, og med sprinter ville vi ha en bedre oversikt over status på prosjektet. Ettersom vi bare er en gruppe på tre var det ikke veldig relevant med mange av utvidelsene, men en fast sprint for levering av forskjellige oppgaver er veldig relevant når det er en hard frist når det gjelder levering av rapport.

Andre deler av Scrum som ville være veldig nyttig å ha med under utviklingen, er en såkalt *Product Backlog*. Det er en stadig oppdaterende liste av oppgaver som skal utføres gjennom hele utviklingsprosessen. I Scrum er det også noe som kalles *Sprint Backlog*, som er en oppgaveliste over hva som må gjøres i den pågående sprinten. Muligheten til å kunne legge på flere oppgaver, samt kunne holde styr på hva som er ønsket og hva som må gjøres til en nær frist, gjør det blant annet mulig å kunne ta innspill fra oppdragsgiver uten at det forstyrrer utviklingsprosessen.

Den andre metoden som ble vurdert er Kanban. Metoden baserer seg på å gi utviklere en stor del frihet, ved å la de selv velge hvilke oppgaver de skal gjøre. De eneste begrensningene som blir satt på utviklingsgruppen er en forhåndsdefinert begrensning på hvor mange oppgaver som skal kunne gjøres samtidig i de forskjellige stegene av utviklingen. For eksempel, på følgende eksempel kan man bare jobbe på tre ting samtidig 2. På pro-

sjektet vårt ville det bety at hvert gruppemedlem ville ha en eller to oppgaver å jobbe på om gangen, når disse ble gjort ferdig ville de velge en ny oppgave.



Figur 2: Eksempel på Kanban i bruk[2]

På eksempelet over kan man se hvordan utvikling er organisert med Kanban. Oppgaver blir satt opp i en backlog på en lignende måte som Scrum, men i stedet for å fordele oppgavene over en bestemt periode, kan man bare ha et visst antall oppgaver under utvikling om gangen. Disse kan da flyttes videre til neste utviklingssteg som testing, for å gjøre plass for andre oppgaver.

2.1.2 Argumentasjon

Det store skillet mellom Scrum og Kanban er den mer faste arbeidsrytmen og strukturen fra Scrum, i forhold til større frihet i Kanban. I Kanban står utviklere frie til å velge egne oppgaver, som gjøres i lengre perioder. Scrum derimot setter opp oppgavene for alle i begynnelsen av en såkalt ”sprint” som må gjøres innen et angitt tidsrom. Den tilbyr også mange små utvidelser man kan benytte seg av, som kan tilpasses utviklingsmodellen ettersom det trengs avhengig av for eksempel størrelse på utviklingsgruppen.

Etter diskusjon endte vi opp med å velge Scrum som vår utviklingsmodell. Det viktigste argumentet var å ha muligheten til å ha faste frister for deler av oppgaven som ville gi oss klar tilbakemelding på hvordan vi ligger an i forhold til prosjektplanen. Ettersom det er en absolutt frist for innlevering av rapporten må man sette opp frister som bør holdes, for å komme til et forventet mål i utviklingen. I tillegg er det mulig å utvide Scrum med forskjellige utvidelser som kan tilpasse prosjektet ytterligere.

2.1.3 Scrum utvidelser

Bortsett fra det mest grunnleggende i Scrum, som en backlog og en hardt definert lengde på sprinter, er det opp til hver utviklingsgruppe å velge hva som er nyttig å bruke under utvikling av sitt prosjekt. Vi har valgt å bruke to utvidelser ifra Scrum.

Daily Scrum

Blant utvidelser som Scrum tilbyr har vi valgt å benytte daglige scrum møter, som generelt skulle holdes fysisk på universitetet. Ettersom gruppemedlemmene ikke har jobbet mye med hverandre før, var det risiko for at noen i gruppen ville bli forsinket med jobben, og kanskje skape andre problemer for gruppen. For å holde oversikt og oppdage slike problemer raskt er det nyttig å møte fysisk ofte, siden sosialt press fører til at personer mer sannsynlig vil holde fristene. I tillegg er det mulig å dele sine meninger og oppdage problemer raskt, slik at det er mulig å forminske skadene.

Story Points

En annen utvidelse fra Scrum som vi bestemte å bruke under planleggingsfasen er såkalte ”story points”. Hver oppgave er annerledes og forskjellige oppgaver kan variere veldig når det gjelder kompleksitet og tid til utføring. Story points forenkler planleggingsprosessen ved å vurdere hver oppgave etter hvor mye tid den er forventet til å ta, og dermed ha bedre grunnlag i tildeling av oppgaver. Dersom det blir oppdaget at noen oppgaver er mer krevende enn andre kan man tildele antall oppgaver ut fra mengde, slik at arbeidsfordelingen blir lik mellom gruppen.

2.2 Gjennomføring

2.2.1 Arbeidsplan

Under planleggingen av prosjektet satt vi opp en ukeplan som baserte seg på at vi skulle møte daglig, og være forberedt på å jobbe sammen om nødvendig. Følgende er en tabelloversikt over den vanlige arbeidsuken for gruppen:

Mandag	Tirsdag	Onsdag	Torsdag	Fredag
9:00-11:00 (Annenhver)	**9:00-9:15**	**9:00-9:15**	**8:15-12:00**	**9:00-9:15**
Planning Meeting	Daily Scrum	Daily Scrum	Mobile Prog.	Daily Scrum
11:00-11:30	**9:15-16:00**	**9:15-16:00**	**12:00-12:15**	**9:15-12:00**
Veileder	Arbeid	Arbeid	Daily Scrum	Arbeid
11:30-16:00			**12:15-16:00**	**12:00-14:00**
Planleg. og Arbeid			Arbeid	Mobile Prog.
16:00 (Annenhver)				**14:00-16:00**
Oppdragsgiver				Prof. Prog.

Tabell 1: Tidstabell for møter

Annenhver mandag ble det holdt Planning Meeting for å bestemme hva som skulle gjøres i løpet av de to neste ukene. Dette møtet var satt til å vare i to timer, hvor det etterpå ble holdt møte med veileder for å blant annet diskutere planen videre. Møtene med veileder ble holdt hver uke og ble brukt stort sett til å få råd om hvordan utviklingen av prosjektet foregikk fra et mer utenforstående synspunkt. Etter møte med veileder gikk gruppen sammen og diskuterte problemer og temaer som ble tatt opp under møtet for å planlegge resten av uken i større detalj. Dersom Planning Meeting ikke ble ferdig før møte med veileder ble den ferdiggjort i den tiden også. Etter planen ville det neste være å jobbe sammen til kl. 16. Annenhver mandag, som regel samme mandag som det ble holdt Planning Meeting, ble det også holdt møte med oppdragsgiver for å diskutere prosjektet samt ønsker fra gruppemedlemmer og oppdragsgiver. Planen ville da bli jus-

tert om nødvendig for å ta med tasks basert på ønsker fra oppdragsgiver. På slutten av hver fredag, så vi nærmere på arbeid gjort under sprinten. Vi gikk gjennom gjenstående oppgaver og hvordan noen av oppgavene hadde blitt fullført, med kodegjennomgang på storskjerm hvis nødvendig.

Etter oppsett av plan og ferdigstilling av de ukentlige møtene på mandager ble resten av uken satt av til rent arbeid. Hver dag kl. 9 var det planlagt et daglig scrum møte, hvor hvert gruppemedlem presenterte kort hva de har gjort den forrige dagen. Dersom det ble oppdaget problemer ville de bli diskutert på møtet. Resten av dagen ble så satt av til arbeid med prosjektet.

Avvik i arbeidsplanen ble også bestemt, avhengig av forelesningstimer i de to emnene som gruppemedlemmene hadde. På grunn av oppgaver i Mobile Programming, ble Daily Scrum flyttet til kl. 12 på torsdager. Fra denne tiden ble det ingen felles arbeid grunnet forelesning i begge emnene rett etter hverandre.

2.2.2 Planlegging av prosjektet

Den første dagen med formell planlegging skjedde den 15. januar, etter at vi var på det første lynkurset for bachelorprosjekter og hadde hatt møte med veileder. Gjennom en uke med arbeid utarbeidet vi en plan for utviklingen av prosjektet, satt opp kommunikasjonskanaler og gjorde diverse annet forberedelsesarbeid. Planen vi utarbeidet, og det opprinnelige GANTT-diagrammet som gir en enkel oversikt over vårt planlagte utviklingstempo, er vedlagt i vedlegg A. Denne planen ble veldig nyttig når det gjelder oversikt over når visse deler av prosjektet stort sett skulle være ferdig. Prosjektplanen ble også mye brukt under planlegging av sprinter, som et grunnlag på hvilke oppgaver som skulle utføres på hvilket tidspunkt.

2.2.3 Møter

Planning Meeting

Planlegging for hver sprint ble foretatt på et såkalt *Planning Meeting*, som markerer startpunktet for hver sprint. På disse møtene ble det diskutert status etter forrige sprint, hvilke mål som var nådd, og hva som skulle gjøres de neste to ukene ifølge prosjektplanen. Lengden på disse møtene var forventet til å variere. Ettersom møtet kunne vare lengre enn to timer, og vi hadde planlagt møte med veileder to timer etter Planning Meeting, var det satt opp en fortsettelse av Planning Meeting etter møtet med veileder. Dette var også ment til å gi oss rom til forandring ved bruk av innspill fra veileder.

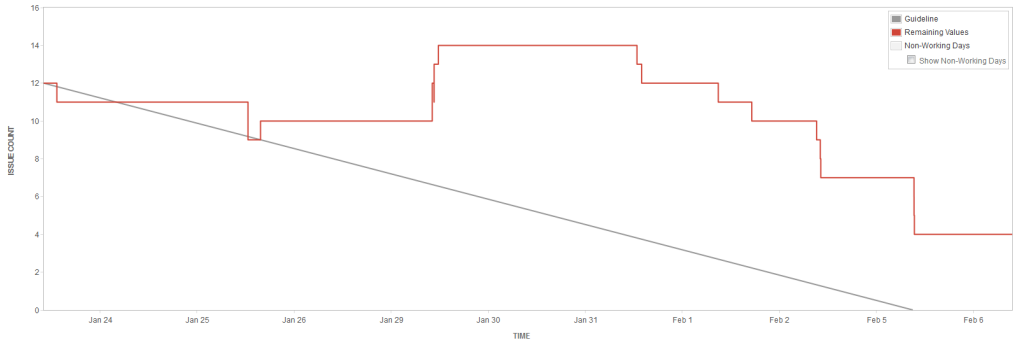
Møter med veilder

Sammen med gruppemøter var det satt opp regelmessig kontakt med veileder og oppdragsgiver. Ukentlige møter med veileder var ment til å gi oss muligheten til å spørre om mulige problemer med arbeidstempo eller arbeidsmengde, og spørsmål om hva vi burde fokusere på i prosjektet. Det kom blant annet frem at den opprinnelige prosjektplanen ikke var bra nok, og vi fikk en mengde tips om hvordan den kunne forbedres. Takket være det fikk vi på plass en bedre prosjektplan som sammen med andre råd hjalp oss mye i utviklingsprosessen.

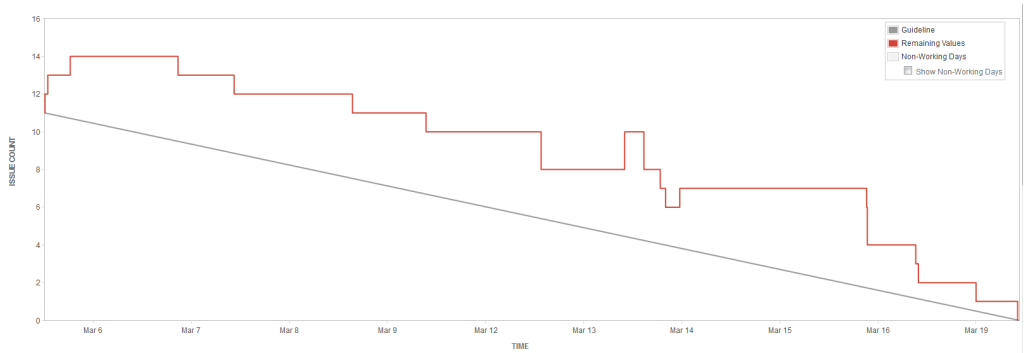
Møter med oppdragsgiver

Møter med oppdragsgiver ble bestemt til å holdes fysisk på campus hvis mulig, med minst en av to oppdragsgivere fysisk til stede på hvert møte. Når en av oppdragsgiverne ikke kunne møte fysisk, benyttet vi Skype som kommunikasjonsverktøy i stedet for å flytte møtetidspunktet, slik at kontakt med oppdragsgiver ble holdt jevnlig. Dessuten ble det satt opp kontakt med oppdragsgiver i sanntid gjennom Slack [25], som er det samme kommunikasjonsverktøyet vi selv benyttet under utvikling. Bruken av Slack til både utvikling og kommunikasjon med oppdragsgiver var ment til å gjøre problemløsing av de forskjellige problemene som kunne komme opp underveis enklere.

2.2.4 Arbeidsmetodikk



Figur 3: Gjennomgang av sprint 1



Figur 4: Gjennomgang av sprint 4

Bortsett fra planlegging og møter kommer det reelle arbeidet. Vi valgte å møte sammen og jobbe på campus, målet med dette var å gi flere muligheter for å jobbe sammen, og arbeide med oppgavene som ble bestemt under Planning Meeting. Dette var viktig fordi flere deler av løsningen var avhengig av hverandre og det var derfor viktig at vi hele tiden var på samme side.

Resultatet var at mesteparten av jobben som skulle gjøres ble gjort i tide. Når det oppstod problemer var andre i gruppen tilgjengelige, og rask hjelp var tilgjengelig dersom noen av gruppemedlemmene hadde spørsmål om andres kode e.l. I tillegg ble noe av arbeidet gjort hver for oss. Under oppstart av prosjektet var det mange oppgaver som krevde tett samarbeid, men når strukturen på prosjektet kom på plass, ble det lettere

å dele arbeidet inn i mer spesifikke oppgaver som kunne bli gjort hjemme. Selv om vi etter arbeidsplanen skulle være tilgjengelige helt til ettermiddagen, var det sjeldent at arbeidet varte lengre enn til kl. 14. Etter hvert som oppgavene ble mer definert og klare, begynte flere av de daglige møtene å avslutte med at gruppen gikk hjem med arbeidsoppgaver satt opp for resten av dagen. Hvis gruppen hadde større arbeidsoppgaver som ville ta lenger tid, ville det daglige møtet i noen tilfeller utsettes til dagen etter, for å se gjennom arbeidet.

2.2.5 Problemer under utvikling

Rundt påsketiden oppstod det problemer med organisering og forstyrrelse av arbeidsrytmen. Etter planen var det ikke mye arbeid som skulle gjøres i løpet av påsken, men dette tomrommet forskjøv offisiell avslutning av sprint fem frem til 16. april. Selv om arbeidet fortsatte i perioden etter påske ble det ikke holdt et skikkelig planleggingsmøte før da. Dette problemet var delvis forventet, men har fortsatt satt dokumentasjon og planlegging ned i prioritetslisten i en periode.

2.3 Sprintoversikt

Under utviklingsfasen hadde vi fem sprinter på rundt to uker hver, med unntak av den siste som havnet midt i påskeferien. Etter påskeferien fortsatte arbeidet med det som ble bestemt og vi forberedte oss på omstilling av arbeidsrytmen for å skrive rapport, ved å se på arbeidet som ble gjort og ta en del forberedelser hvor mulig.

Tabellen 2 viser en oversikt over sprintene som vi har hatt i løpet av utviklingsprosessen.

Sprint	Åpnet (etter start)	Lukket	Ikke gjort
1	14 (3)	10	4
2	20 (7)	14	6
3	13 (5)	10	3
4	17 (6)	17	0
5	25 (2)	19	4

Tabell 2: Oppgaver i Sprintene

2.3.1 Arbeid før sprint 1 (15. januar - 23. januar)

Første arbeidsmøte ble holdt den 15. januar etter å ha først holdt møte med veileder uken før. Hele uken ble brukt på planlegging samt mandag uken etter, dermed ble starten på sprint 1 forsinket med en dag.

2.3.2 Sprint 1 (23. januar - 5. februar)

Sprint 1 ble som nevnt forsinket med en dag. De første tre dagene ble brukt på design og oppsett av databasen. Når det ble gjort ferdig ble flere oppgaver fordelt mellom gruppemedlemmene og arbeid på Androidløsningen begynte. Blant disse var REV-40 som handlet om implementasjon av strekkodelesing i Android, og REV-77 som handlet om å implementere tilgang til produkter via vår server.

Det har også vært en oppgave å finne en produktkatelog med strekkoder som man kunne bruke som en start for databasen, dette har vært en oppgave underveis i de tidlige sprintene. Ofte ville oppdragsgiver gi komme opp med nye løsninger underveis, som vi

satt av tid til å vurdere.

2.3.3 Sprint 2 (5. februar - 20. februar)

Sprint 2 begynte offisielt den 5. februar, men grunnet forsinkelsen fra forrige sprint ble planleggingen av sprint 2 forskyvet til den 6. februar. Det meste av grunnleggende brukerkonto funksjonalitet som skulle komme på plass i løpet av sprint 1, ble forsinket og ferdig rundt 7. februar.

Som hovedoppgave i sprint 2 ble det satt opp et mål om å ha på plass det meste av den grunnleggende funksjonaliteten i webapplikasjonen. Det var da fokus implementasjon av PRS både på skriving og visning av anmeldelser. Dette målet ble delvis nådd, men design på produktside ble ikke ferdig i løpet av sprint 2 og ble forandret i løpet av de neste sprintene ut ifra tilbakemeldinger på brukertesting.

I sprint 2 ble det også startet forsøk på testing og oppsett av kvalitetssikringsverktøy som Sonarqube. Verktøyene ble satt opp, men bruk av integrerte tester har ikke blitt fulgt opp grunnet flere integrasjonsproblemer og tid som ville ha vært nødvendig for å sette opp testene, som uansett trolig ville måtte forandres kontinuerlig på under utvikling.

2.3.4 Sprint 3 (20. februar - 5. mars)

Som konsekvens av den originale forsinkelsen, ble start av sprint 3 også forskyvet med en dag. Sprint 2 var mye større i arbeidsmengde enn vi forventet, og den første uken ble brukt til å fortsette med grunnleggende funksjonalitet for webløsningen.

Den opprinnelige oppgaven for sprint 3 var å begynne en større brukertest av webløsningen, noe som vi planla å bruke store deler av den første uken på. Det endte opp med at vi gjennomførte brukertesting av familie og venner gjennom helgen midt i sprint 3.

Etter brukertestingen ble det gjort videreutvikling av webapplikasjonen basert på tilbakemeldinger fra testingen. Videreutviklingen av PRS tok resten av sprinten. Dette resulterte i en ferdig implementasjon av PRS som et konsept.

2.3.5 Sprint 4 (5. mars - 19. mars)

Ved start av sprint 4, nådde vi det planlagte midtpunktet i utviklingsfasen. Grunnet problemer med å få på plass grunnleggende funksjonalitet for brukertestene, konkluderte vi med at en fullverdig versjon på både Android og Web var urealistisk. Valget var mellom å få på plass en mer komplett Android- eller webløsning, eller ha to mye mindre løsninger. Vi bestemte at utviklingen av en mer komplett versjon ville være mer lærerikt enn å utvikle for to plattformer og bruke tid på å gjøre en oppgave to ganger.

Med det mer fokuserte målet på plass, satt vi opp en oppgaveliste som skulle gjennomføres i løpet av de to neste ukene, og klarte å nå målet i sprinten for første gang.

2.3.6 Sprint 5 (19. mars - 16. april)

Sprint 5 endte opp som en fire uker lang sprint, grunnet dårlig kommunikasjon om hvordan påskeferien skulle brukes, og det aktuelle arbeidet som ble gjort i påsken. Oppgaver som ble fordelt i starten av sprinten ble jobbet med frem til den siste møtedagen, så ble det arbeidsstans frem til etter påske. Etter påske opplevde gruppen problemer med å finne arbeidsrom på universitetet og å bli ferdig med oppgavene sine, noe som ble bestemt til å bli gjort ferdig innen slutten av uken etter påske.

Arbeid satt opp for sprint 5 var stort sett alt av den resterende funksjonaliteten, mens

sprint 6 ble tenkt til å bli brukt stort sett til å forbedre den eksisterende koden. Dette gjorde tapet av tiden mindre kritisk enn det kunne vært, spesielt siden vi hadde stoppet seriøs utvikling av webløsningen på starten sprint 4.

Den siste uken ble brukt på planlegging av rapporten og arbeid i de siste ukene, samt forbedring av kode.

2.3.7 Sprint 6 (16. april - 23. april)

Fra og med 16. april ble det bestemt å kutte ned lengden på sprintene til en uke hver. Fordelt på tre personer ville det ikke ta to uker å gjøre en del av det første utkastet til rapporten ferdig, og økning av arbeidshastigheten ville bidra til bedre endelig versjon av rapporten. Kapitlene og delkapitler ble fordelt mellom gruppemedlemmene. Arbeidet foregikk for det meste felles. Dette gjorde at vi lett kunne spørre hverandre om hjelp, samtidig som det hjalp oss å passe på at alt hang sammen.

2.3.8 Sprint 7 (23. april - 30. april)

Etter den første uken med rapportskriving klarte vi å skrive det meste av de tre første kapitlene. Bruken av Jira for å holde styr på utviklingen av rapporten viste seg å ikke være den beste løsningen, ettersom det eneste vi brukte den for var å skrive ned kapitlet som hver av oss skulle skrive. Vi valgte derfor å droppe dette. De fleste av de resterende kapitlene ble satt opp til å skrives i Sprint 7, med unntak av konklusjonen som skulle skrives i fellesskap når alt annet var ferdig.

2.3.9 Sprint 8 (30. april - 7. mai)

Etter to uker har det meste av rapporten kommet på plass. I starten av uken ble de siste delkapitlene avsatt til skriving, som ble gjort fram til midten av uken. Fra midten av uken begynte vi å gå gjennom det vi har skrevet, for å fikse skrivefeil og inkonsistens i teksten. Det første utkastet av rapporten ble også sendt til veileder for tilbakemelding.

2.3.10 Sprint 9 (7. mai - 16. mai)

Den siste sprinten ble i all hovedsak brukt på rapportskriving. Det ble ikke like mye ren skriving som forrige sprint, men mer gjennomgang og forbedring. Vi skrev ned punkter for forbedring og jobbet med disse. De siste dagene hadde vi litt ekstra tid, og benyttet denne tiden til å forbedre brukergrensesnittet i androidapplikasjonen.

3 Kravspesifikasjon

I dette kapittelet skal vi beskrive funksjonelle og øvrige krav som må imøtekommes for at applikasjonen skal fungere i henhold til hvordan den er beskrevet i tidligere kapitler.

3.1 Funksjonelle krav

Kort sagt skal vi lage en brukergenerert plattform for anmeldelser, hvor brukere skal ha tilgang til verdifulle produkterfaringer fra andre brukere. Brukere skal kunne anmelde og vurdere konsumentvarer fra en rekke ulike kategorier og få informasjon og brukeranmeldelser om ønsket produkt.

En bruker må registreres før applikasjonen kan tas i bruk. Dette fordi vi har brukergenerert innhold i løsningen, og trenger å ha kontroll over potensielt misbruk fra brukere. Hvis en bruker av applikasjonen ikke er logget inn, vil de bli møtt med en registreringsside hvor de kan lage en ny bruker, eller logge inn med en eksisterende bruker.

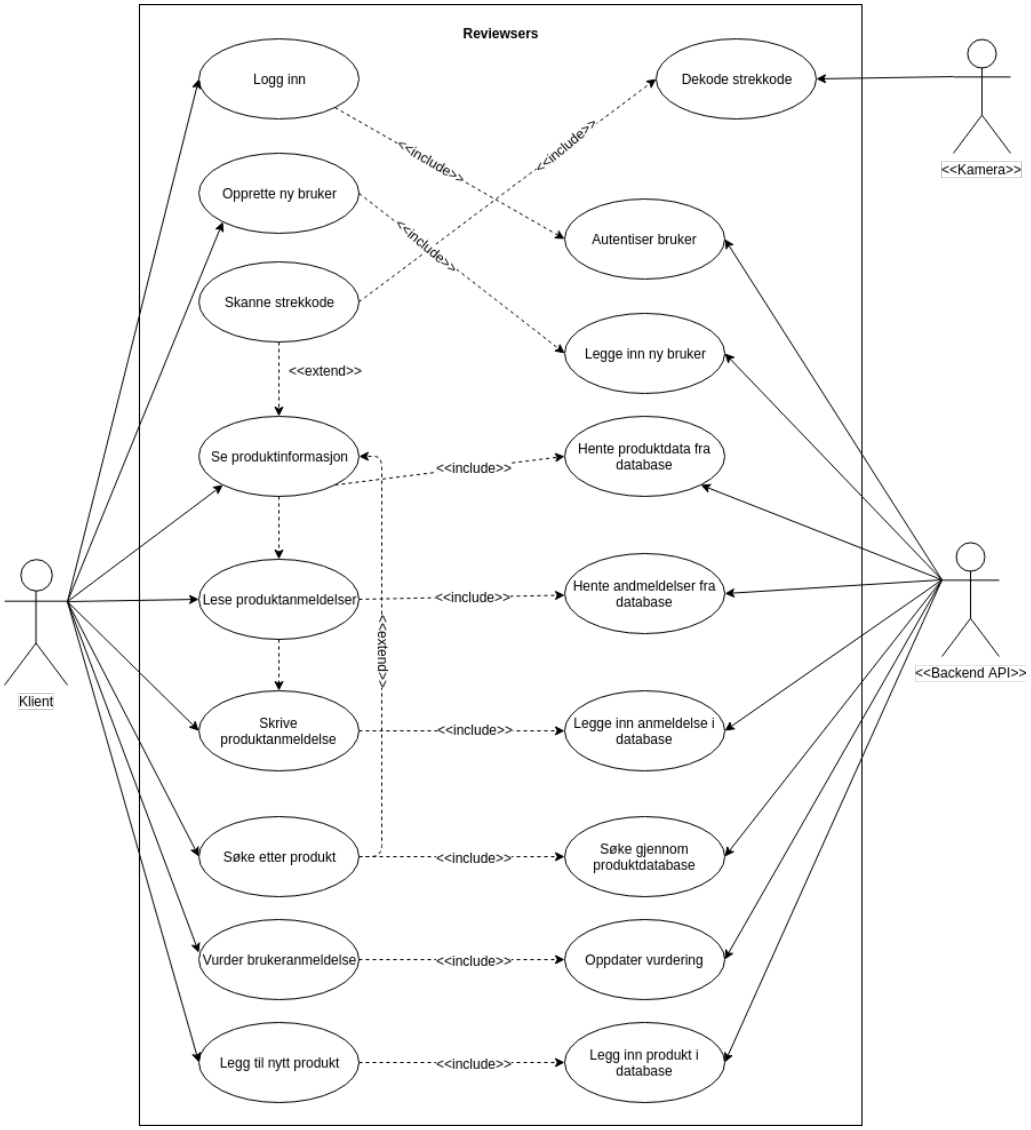
Innloggede brukere av applikasjonen skal kunne benytte telefonens kamera til å skanne inn en strekkode og få i retur produktinformasjon med tilhørende anmeldelser. For å kunne muliggjøre dette må vi benytte oss av et API eller bibliotek som kan dekode bilder av EAN-13 strekkoder til numerisk format slik at systemet kan benytte seg av dem. Alternativt skal brukere kunne søke opp ønsket produkt med et søkeord og få i retur samme informasjon. Dette krever at vi implementerer eller bruker en søkealgoritme som gir resultater rangert etter relevans.

Det skal være mulig for en innlogget bruker å skrive en ny anmeldelse for et produkt. Anmeldelsene er delt opp i ”delanmeldelser” med hver sin ”tagg” (enkelt forklart en egenskap av produktet). Denne strukturen er beskrevet nærmere i seksjon 5.4. Brukere skal også kunne rangere en anmeldelse etter nytteverdi, hvor to verdier kan velges mellom (hjelpsom, ikke hjelpsom). Dessuten bør også brukere kunne legge til produkter manuelt, dersom det ikke allerede eksisterer i databasen vår. Dette fordi vi forventer at tredjepartsløsningen som tilbyr oss produktkatalogen ikke vil være komplett.

3.1.1 Use-Case Diagram

Vi har illustrert et use-case diagram ved hjelp av applikasjonen Draw.io (se figur 5).

Vi har valgt å bruke aktøren ”Klient” for både web- og mobilapplikasjonen for å gjøre diagrammet mer leselig. Felles use-caser for web og mobil er lesing og skriving av anmeldelser, opprettelse av bruker og innlogging av bruker. Resten av use-casene er kun for mobilapplikasjonen. En ferdig webapplikasjon vil derimot i utgangspunktet ha samme funksjonalitet som mobilapplikasjonen, med unntak av skanning av strekkoder.



Figur 5: Use case diagram

3.1.2 Høy-Nivå Use-Case

Under vises en mer detaljert oversikt over hvert enkelt use-case.

Use Case	Logg inn (Mobil og web)
Aktør	Klient
Beskrivelse	Brukeren åpner applikasjonen for første gang, uten å ha logget inn på en stund, eller brukeren har valgt å logge ut tidligere. Brukeren blir møtt med en påloggingsskjerm, fyller inn e-post addresse og passord. Dette blir sendt til backend for autentisering av brukeren. Brukeren blir logget inn.

Use Case	Autentiser bruker
Aktør	Backend
Beskrivelse	Backend mottar en forespørsel fra en klient for innlogging av bruker. Det sjekkes at en bruker med angitt e-post adresse eksisterer. Tilsendt passord krypteres, og det sjekkes at output fra *hashing-funksjonen* stemmer med kryptert passord i databasen. En ny token sendes i retur til klient.

Use Case	Opprette ny bruker (Mobil og web)
Aktør	Klient
Beskrivelse	Brukeren åpner applikasjonen for første gang, og blir møtt med en påloggingsskjerm. Brukeren trykker registreringsknappen, fyller ut brukerinformasjon og registrerer seg som ny bruker.

Use Case	Skanne strekkode (Mobil)
Aktør	Klient
Beskrivelse	Bruker åpner applikasjonen og velger ”skann produkt”. Mobilens kamera starter opp og brukeren retter kameraet mot strekkoden på produktet. Strekkoden dekodes og sendes til backend. Bruker får i retur anmeldelser og produktinformasjon. Hvis produktet ikke eksisterer vil bruker kunne legge til produktet i databasen. Et scenario for dette use-caset er når brukeren er i butikken og er interessert i å vite mer om et produkt.

Use Case	Dekode strekkode (Mobil)
Aktør	Kamera
Beskrivelse	Mobilens kamera mottar et bilde av en EAN-13 strekkode og dekoder informasjonen til numerisk format.

Use Case	Se produktinformasjon (Mobil)
Aktør	Klient
Beskrivelse	Bruker har skannet en strekkode eller søkt etter og valgt et produkt. Applikasjonen ber om produktinformasjon fra backend. Brukeren får i retur produktinformasjon, som vises i applikasjonen.

Use Case	Hente produktdata fra database
Aktør	Backend
Beskrivelse	Backend mottar en forespørsel for produktinfo. Backend ser etter produktet i databasen og sender informasjonen i respons til klient.

Use Case	Lese produktanmeldelser (Mobil og web)
Aktør	Mobilbruker
Beskrivelse	Brukeren er inne på en produktside og blar ned for å se på produktanmeldelser. Bruker ber om anmeldelser fra backend og disse vises i applikasjonen.

Use Case	Hente anmeldelser fra database
Aktør	Backend
Beskrivelse	Backend mottar en forespørsel for anmeldelser for et spesifikt produkt eller bruker. Backend henter anmeldelser fra database og sender disse i respons til klient.

Use Case	Legge inn anmeldelse i database
Aktør	Backend
Beskrivelse	Backend mottar en anmeldelse for et produkt fra en bruker. Backend validerer at anmeldelsen er i korrekt format og legger denne inn i databasen. Respons sendes til klient om at anmeldelse ble lagt inn.

Use Case	Søke etter produkt (Mobil)
Aktør	Klient
Beskrivelse	Bruker velger å søke etter produkter i applikasjonens ”app bar”. Brukeren skriver inn et søkeord og får opp en liste med produkter som matcher ordet, sortert etter relevans.

Use Case	Søke gjennom produktdatabase
Aktør	Backend
Beskrivelse	Backend mottar en søkeforespørsel fra en bruker. Backend bruker søkeordet til å matche produkter i produktdatabasen og sender disse i respons til klient.

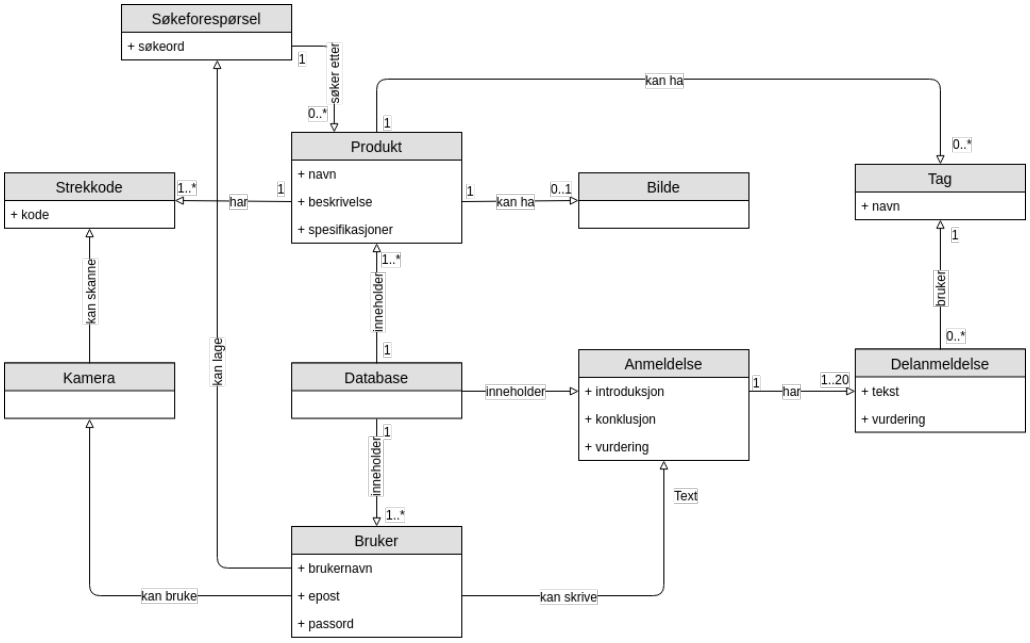
Use Case	Vurder brukeranmeldelse
Aktør	Klient
Beskrivelse	Brukeren har lest en anmeldelse, trykker på anmeldelsen, og velger ”vurder anmeldelse”. Brukeren velger ”hjelpsom” eller ”ikke hjelpsom” avhengig av vurdering av lest anmeldelse.

3.1.3 Utividet Use-Case

Use Case	Skrive produktanmeldelse (Mobil og web)
Aktør	Klient
Flyt	1. Brukeren har erfaringer med et produkt som ønskes delt med andre brukere. 2. Brukeren søker etter eller skanner inn strekkode for dette produktet og produktsiden vises. 3. Brukeren trykker ”skriv anmeldelse”. 4. Brukeren skriver en introduksjon for anmeldelsen, eller lar den stå tom. 5. Brukeren trykker ”legg til tagg”. 6. Liste med tilgjengelige tagger/egenskaper for produktet vises, hvor brukeren kan velge eller legge til nye tagger. 7. Brukeren velger en tagg, og et nytt skrivefelt vises. 8. Brukeren skriver sin anmeldelse om denne taggen/egenskapen av produktet, og rangerer denne delen av produktet med 3 mulige verdier: negativt, nøytralt eller positivt. 9. Brukeren foretar steg 5-8 fra 0-20 ganger til. 10. Brukeren skriver en konklusjon for anmeldelsen, eller lar den stå tom. 11. Brukeren rangerer hele erfaringen med produktet. 12. Brukeren trykker ”post anmeldelse”.
Beskrivelse	Bruker har erfaringer om et produkt som ønskes delt med andre brukere. Brukeren velger ”skriv anmeldelse” på produktsiden og skriver erfaringer innen utvalgte deler/tagger av produktet. Brukeren sender anmeldelsen til backend.

3.1.4 Domenemodell

For å lettere få en oversikt over systemet har vi modellert en domenemodell, som viser en abstraksjon av systemet og sammenhengen mellom dets komponenter og data. (Se figur 6).



Figur 6: Domenemodell for Reviewsers

3.2 Øvrige krav

3.2.1 Produktdatabase

For å kunne tilfredstille alle de funksjonelle kravene, trenger systemet vårt tilgang til en stor produktkatalog inneholdende informasjon om produkter som er tilgjengelig på markedet. Det er viktig at databasen holdes oppdatert når nye produkter lanseres og den må inneholde produkter fra alle de store merkevarene. Produktene i databasen må inneholde den tilhørende EAN-13 koden (eller koder) som brukes av butikkene slik at produktet kan identifiseres når brukeren skanner en strekkode, og annen type data som produktnavn og produktbeskrivelse. Den bør helst også ha produsentbilder og produktspesifikasjoner. Løsningen vår er utviklet for en rekke produktkategorier, som elektronikk og klær, så databasen bør inneholde varierte typer av produkter.

Vi har utarbeidet følgende krav til produktdatabasen:

- Produkter i databasen må inneholde grunnleggende produktinformasjon som navn, helst også spesifikasjoner, og må inneholde produktets EAN-13 strekkode.
- Produktdatabasen må holdes oppdatert med nylig lanserte produkter.
- Det bør være informasjon om hvilket land produktet selges i, for å tilpasse applikasjonen for forskjellige nasjonaliteter.
- Kategorier er ikke absolutt nødvendig, men en stor fordel.

Plattform

Frontend

Applikasjonen skal utvikles for en mobilplattform. Her er det en rekke plattformer å velge mellom, men vi har valgt å avgrense oss til Android. Det skal også utarbeides en enkel prototype for webløsningen, som kan fungere som en erstatning for øvrige plattformer.

Backend

Backendløsningen skal utvikles for servere som kjører Linux operativsystem. Siden open-source programvare brukes, er det sannsynlig at løsningen også vil fungere på andre operativsystem, men dette vil ikke testes.

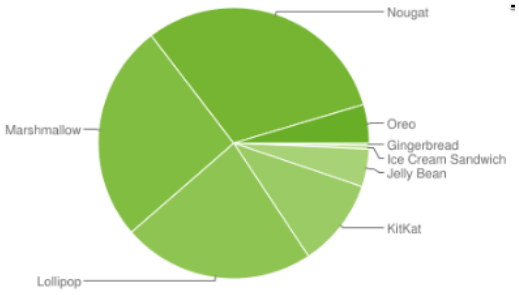
Kompatibilitet

Når man skal utvikle for Android-plattformen, er det viktig å ta valget om hvilket minimum *API-nivå* man velger å utvikle for. Dette valget påvirker hovedsakelig to ting: hvor stor andel av brukere med Android-telefoner som kan kjøre applikasjonen, og hvor mye ny funksjonalitet fra Google sitt SDK man har tilgang til. Den første forbedres jo lengre ned man går i API-nivå, og den andre begrenses. De aller fleste har ikke de nyeste Android versjonene installert på telefonen sin, derfor er det viktig å finne en balanse mellom hvor lavt nivå man støtter med tanke på plattformdistribusjonen, og hvor mye funksjonalitet man tolererer å gå glipp av. Ifølge Google er det en god praksis å ta sikte på å støtte minst 90% av brukermassen for Android[26] når man utvikler en Androidapplikasjon.

Google publiserer en oppdatert oversikt over nåværende plattformdistribusjon på Android[3]. Basert på dataene publisert 16. April 2018 (vist i figur 7), valgte vi å utvikle Reviewsers for API-nivå 19 (Android 4.4). Begrunnelsen for dette valget var at vi ikke så på det som hensiktsmessig å gå lengre tilbake enn 19. Dette var fordi API 19, som vist i figuren, står for 10.5% av brukermassen. Går man lavere enn dette derimot er det lite å tjene, kun omtrent 5%. Ved å ha minimumsstøtte for API nivå 19, vil applikasjonen vår nå ut til ca. 94,8% av potensielle Android-brukere.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.4%
4.1.x	Jelly Bean	16	1.7%
4.2.x		17	2.2%
4.3		18	0.6%
4.4	KitKat	19	10.5%
5.0	Lollipop	21	4.9%
5.1		22	18.0%
6.0	Marshmallow	23	26.0%
7.0	Nougat	24	23.0%
7.1		25	7.8%
8.0	Oreo	26	4.1%
8.1		27	0.5%

Data collected during a 7-day period ending on April 16, 2018.
Any versions with less than 0.1% distribution are not shown.



Figur 7: Statistikk over Android plattformdistribusjon per 16. April 2018[3]

3.2.2 Internasjonalisering

Oppdragsgiveren hadde et klart ønske om at applikasjonen skulle sikte seg på en internasjonal brukerbase, ettersom de ønsker å nå så mange potensielle brukere som mulig. De ba oss derfor om å bruke engelsk som språk i første omgang. Det må likevel legges til rette for at applikasjonen lett kan utvides til å støtte flere språk senere. Tekststrenger som er synlig og brukes i applikasjonens brukergrensesnitt må derfor ikke hardkodes (skrives direkte inn), men nøkler med referanser til språkressurser må brukes istedenfor.

Android har god støtte for internasjonalisering, hvor en kan lage slike språkressurser med nøkkel-streng par i form av XML-filer[27]. Telefonens konfigurerte språk vil da bli brukt til å bestemme språket som brukes i applikasjonen.

I webløsningen fokuserte vi ikke på internasjonalisering, men valgte å hardkode norske tekststrenger. Dette fordi webløsningen hovedsakelig ville være en enkel prototype ment for brukertesting, hvor testerne ville være norske.

3.2.3 Sikkerhet og personvern

Et viktig prinsipp vi har hatt når det kommer til personvern var at vi kun skulle lagre brukerinformasjon som anses som helt nødvendig for å ivareta de funksjonelle kravene. På grunn av dette måtte vi i planleggingsfasen utrede hvilken brukerinformasjon vi må ha, kunne være foruten eller som kunne gjøres frivillig å fylle inn. Vi kom fram til at det var hensiktsmessig å kun be om navn, brukernavn, land, e-post adresse, og passord når en bruker registrerer seg. E-post adresse bruker vi for å gjøre det vanskeligere å lage flere brukere per person, da dette ikke er ønskelig med tanke på falske anmeldelser. I tillegg gjør det det mulig å sende viktige meldinger til brukere i framtiden dersom dette er nødvendig. Brukernavn er aliaset som vises for andre brukere når brukeren skriver en anmeldelse. Brukere kan også velge å fylle inn kjønn dersom de ønsker, som vil vises på brukerens profilside.

Ettersom vi ikke lagrer veldig sensitiv informasjon om brukerne, senker dette kravene til sikkerhet en viss grad. Men siden vi overfører passord og e-post adresse over internett til backend er det viktig at denne informasjon ikke kan fanges opp av angripere under overførselen. Overføringen må derfor foregå kryptert fra ende til ende.

Et annet viktig aspekt rundt sikkerhet er validering av data. Der data sendes fra klient til backend, er det en fare for såkalt *SQL injection* angrep. Denne typen angrep kan la en angriper manipulere databasen ved å sende over SQL-spørringer til backend (istedenfor vanlig tekst), dersom disse ikke valideres på forhånd[28]. Man kan aldri stole på informasjon som sendes fra klient, da informasjonen enkelt kan modifiseres av angripere[29].

GDPR

Kort tid etter prosjektet kommer ny personvernlovgivning fra EU, den såkalte *GDPR*, som trer i kraft senere i 2018[19]. Denne loven vil gjelde for alle EU og EØS land, så det er viktig at vi tar den nye lovgivningen i betraktning ved utarbeiding kravspesifikasjonen. Loven er svært omfattende, men vi vil her diskutere kort hovedforandringene i den nye personvernloven og hva vi må ta hensyn til for å følge den.

På Datatilsynets nettsider kan man lese om de nye rettighetene brukere vil få som følge av *GDPR*. Her finnes blant annet ”Retten til å bli glemt”, hvor brukeren nå vil ha rett til å få all personlig data som tjenesten lagret om seg slettet. Det er også ”Rettigheten

til å kreve begrensning" som går ut på at brukere kan kreve at opplysningene kun kan lagres, men ikke brukes til noe. Til slutt, har vi "Retten til dataportabilitet". Dette vil si at brukeren har rett på å få returnert alle opplysninger som holdes lagret om seg på et standard og maskinlesbart format[30].

Oppsumert

Vi har utarbeidet følgende sikkerhet- og personvernskrav for løsningen:

1. Det skal ikke lagres personlig informasjon utover det som anses som nødvendig for å ivareta funksjonaliteten.
2. En bruker tildeles en autentiserings token fra API'et etter innlogging som er gyldig for en viss tidsperiode. Alle foregående forespørsler til API'et skal inneholde denne, og vil sjekkes for gyldighet.
3. All kommunikasjon mellom klient/app og API på backend skal foregå gjennom SSL/HTTPS protokollen som vil sikre kryptering av trafikken.
4. Data som sendes fra klient/app til API'et må valideres for å sikre at angripere ikke kan manipulere databasen.
5. Det må tilbys funksjonalitet som lar brukeren enkelt slette all informasjon vi har på brukeren, inkludert anmeldelser.
6. Det skal være mulig for brukeren å eksportere all informasjon vi har på brukeren til maskinlesbart format. Vi velger her å bruke XML-formatet.

4 Valg av teknologi

4.1 Android SDK

Google sitt Android-SDK står sentralt når det kommer til utvikling på Android. En SDK består som oftest av en rekke systemutviklingsverktøy som brukes for å hjelpe til med å lage en applikasjon. Hvor noen SDK'er fungerer mer som et hjelpeverktøy i utviklingen er Android SDK en mer integrert del av utviklingen på Android. Android SDK inneholder de bibliotekene som kreves for å lage en androidapplikasjon samt ekstra funksjonalitet tilgjengelig i Android API'et. Det gir også tilgang til emulator, dokumentasjon, opplæring og en debugger for kode. Kort sagt er det en essensiell del av utviklingen på Android[31].

Hver gang Google slipper en ny versjon av Android slippes det også en ny Android SDK. Primært sett vil den nye SDK'en bringe med seg nye funksjonaliteter sammenliknet med den gamle i form av et nytt API. Utvikleren kan så implementere de nye funksjonalitetene i sin applikasjon.

Etter at valget falt for å utvikle en native androidapplikasjon, var det aldri diskusjon om å ikke bruke Android SDK. Det finnes måter å skrive deler av applikasjonen i C++ [32] ved bruk av Android NDK (Native Development Kit), som hovedsakelig brukes dersom man har veldig ytelseskrevende kode (f.eks 3d spill)[33]. Dette var derimot ikke relevant for oss å bruke, fordi vi ikke har noen spesiell ytelsessensitiv kode. Vi hadde heller ikke noen C++ biblioteker vi måtte bruke.

4.2 Backend

4.2.1 Node.js

Node.js er et open-source rammeverk som lar utviklere bruke JavaScript for å lage applikasjoner på serversiden. Det er et relativt nytt rammeverk sammenlignet med ande populære alternativer som PHP[34], men skiller seg ut ved at den bruker en *event-drevet* og asynkron modell. Dette gjør at Node.js yter spesielt godt der det er mye blokkerende kall (typisk IO, skriving og lesing fra disk) fordi Node.js kan utføre andre operasjoner imens den venter på at disse operasjonene blir ferdig[35].

I planleggingsfasen brukte vi noe tid på å velge hvilket serversidespråk som skulle brukes for backendløsningen. Her var det to språk vi vurderte som aktuelle: PHP og Node.js. PHP var en sterk kandidat fordi det er et veletablert og mye brukt serversidespråk som også gruppemedlemmene har erfaring med blant annet fra tidligere emner (Database og WWW-teknologi).

Valget falt likevel på Node.js. Denne beslutningen tok vi da vi la særlig vekt på ytelse og effektivitet når det kom til backendløsningen. Vi ønsket at API'et skulle kunne takle så mange samtidige brukere som mulig, samtidig som applikasjonen skulle oppleves som responsiv for brukerne. Den primære funksjonen til backendløsningen ville være å svare på forespørsler fra en klient (i dette tilfellet Android- og webløsningen). I en undersøkelse utført av Kai Lei et al. konkluderes det med at Node.js egner seg mye bedre enn PHP i IO-intensive scenarier [36]. De fleste spørringer til vårt API'et vil resultere i spørringer

til en MySQL database, noe som er en IO-intensiv operasjon. For slik bruk viste Node.js å egne seg bedre enn PHP, ettersom det i tester viser seg å takle omtrent dobbelt så mange spørringer som PHP per sekund under samme testkonfigurasjon[36]. Vi vurderte derfor ytelsesfordelen som overveiende sammenlignet med den tidligere erfaringen vi hadde med PHP.

4.2.2 Express

Express er et webapplikasjon-rammeverk som brukes i form av en modul for Node.js. Prosjektet utvikles av blant annet IBM, og har bidragsytere gjennom GitHub[37]. Express baserer seg rundt såkalte *routes*, som egentlig bare er bestemte URL addresser eller stier til en bestemt destinasjon. Det som gjør Express nyttig for API-utvikling er at den lar utviklerne lage en eller flere funksjoner (kalt *middleware* i Express) som kalles når en spørring mottas og gir full kontroll over hva som skjer mellom en klients spørring til serveren, og hva som sendes i respons[38].

I Express kan man også utnytte de forskjellige HTTP verbene, slik som *GET*, *POST*, *PUT* og *DELETE* slik at en bestemt rute har forskjellig funksjon avhengig av hvilket slikt verb som brukes i spørringen til API'et[38]. Dette gjorde at vi kunne koble de forskjellige API-rutene tett opp mot spørringer til databasen vår, hvor ruter som *GET /user* betyr *hent bruker fra database* og *DELETE /user* betyr *slett bruker fra database* og lignende.

Express er også et såkalt *unopinionated* rammeverk, som betyr at den er veldig fleksibel med tanke på hvordan ting gjøres[38]. Dette er gunstig for oss da vi hadde spesifikke krav når det kommer til hvordan f.eks autentisering skal gjøres i løsningen vår (se seksjon 3.2.3).

4.2.3 Database

Under planleggingsfasen diskuterte vi hvorvidt vi skulle ta i bruk en tradisjonell relasjonsdatabase slik som SQL, eller ta i bruk de nye løsningene, også kjent som noSQL. NoSQL er betegnelsen på et bredt spekter av databaseløsninger som ikke baserer seg på relasjonsmodellen[39].

Et viktig argument som favoriserte SQL var at gruppemedlemmene hadde mye erfaring med dette fra før av, ettersom vi hadde lært om SQL i flere tidligere emner vi har hatt. Hvis vi skulle valgt en noSQL database, ville valget sannsynligvis blitt MongoDB, som bruker en JSON-struktur for databasen[40]. Vi drøftet fordelene og ulempene med de to løsningene, og hva som ville passet best for datastrukturen vår.

For det første har SQL-databaser veldig god støtte for *spørringer* mot tabellene, som tillatter veldig komplekse spørringer og såkalte *joins* som lar deg kombinere flere tabeller i datastrukturen. Dette er en forskjell fra noSQL som ikke har slike komplekse spørringer. En annen forskjell er at SQL-databaser, fordi de er relasjonsdatabaser, kan *normaliseres*, som sikrer at data ikke dupliseres og holdes i en tabell[41]. En noSQL database derimot trenger ikke noe struktur definert på forhånd, og kan derfor kreve mindre forarbeid.

Vi kom relativt raskt fram til valget om å bruke en SQL-basert database. Her valgte vi MariaDB, som er en forbedret videreutvikling av databaseløsningen MySQL[42]. Grunnen til dette var hovedsakelig at det ikke var ønskelig å måtte sette seg inn i noe nytt på databasefronten, da vi allerede hadde Node.js å lære. Selv om noSQL kunne krevd mindre forarbeid med tanke på struktur, ville dette sannsynligvis ikke hjulpet oss mye

da vi måtte brukt mye av tiden på å lære det. I tillegg til dette tok vi i betraktning at databasestrukturen kunne bli relativt kompleks, og tenkte derfor at vi kunne dra nytte av SQL sitt spørringsspråk. En annen grunn som også veide mot SQL var at det vil bli lettere å slette all data om en bruker. Dette er som nevnt i seksjon 3.2.3 et krav vi har til løsningen. Gjennom normalisering av databasen vil ikke brukerdataen dupliseres og det blir lettere å holde oversikt over hvor brukerdataen ligger.

4.3 Webapplikasjonen

4.3.1 JQuery

JQuery er et bibliotek for JavaScript som har lenge hatt som mål gjøre mer med mindre kode enn vanlig JavaScript. Det har derimot de siste årene blitt mindre sant ut ifra nyere oppdateringer av JavaScript, og flere alternativer som React og Angular begynner å bli mer populære. Vi har derimot valgt å bruke JQuery mest på grunn av at det var et rammeverk alle gruppemedlemmene var erfarne med og vi så ingen spesiell grunn til ikke å bruke det. JQuery er fortsatt nyttig og gir oss muligheten til å legge til animasjoner og ta bruk av mye eksisterende dokumentasjon. Noe JQuery fortsatt gjør bedre enn mange liknende løsninger er kompatibilitet mellom nettlesere[43], som gjør at vi kan forvente at koden vi tester ut i f.eks Google Chrome vil fungere likt i nettlesere som Mozilla Firefox eller Microsoft Edge.

4.4 Technical memo: Valg av produktkatalog tilbyder

4.4.1 Bakgrunn

For å kunne tilfredsstille det meste av de funksjonelle kravene til løsningen, måtte vi ha tilgang på en stor katalog av konsumentprodukter. Oppdragsgiveren hadde undersøkt dette på forhånd og merket seg at det finnes flere tredjepartsløsninger som tilbyr slike produktkataloger. Disse kommer i form av API'er som lar utviklere bruke katalogen deres, eller at man får direkte tilgang til katalogen ved å laste ned en maskinlesbar fil med produktdataene.

Vi undersøkte de forskjellige tredjepartsløsningene for å se hvilke som egnet seg mest for løsningen vår. En av disse var *EAN-search.org*, denne reklamerer med å ha "130 millioner" produkter i databasen[44]. Man får tilgang til produktkatalogen gjennom et API, og de har produktinfo som EAN-13 kode, navn og opprinnelsesland. En annen vi undersøkte var *upcitemdb.com*. På likhet med den andre påstår disse å ha "150 millioner" produkter i katalogen, og den aksesseres gjennom et API[45]. Den tredje tilbyderen vi undersøkte var Icecat[46]. Denne skiller seg litt ut fra de to andre. Icecat får mye av produktinformasjonen direkte fra produsentene[10], og bruker ikke et API, men tilbyr direkte nedlastinger av produktkatalogen i form av et XML dokument.

4.4.2 Diskusjon

Datakvalitet

Det var mye forskjell når det kommer til kvaliteten på dataene hos de ulike tilbyderne. Selv om *EAN-search.org* og *upcitemdb.com* hadde langt større antall produkter i katalogen enn Icecat, må også *datakvaliteten* tas i betraktning. For å undersøke dette gjorde vi noen produktsøk hos tilbyderne. Det vi kom fram til var at det hos de to førstnevnte var dårlig konsistens når det kom til kvalitet på dataen. Noen av produktene hadde eksempelvis

navn og bilde, imens andre ikke hadde det. Icecat hadde en langt mer standardisert produktkatalog, hvor hvert av produktene som regel ville ha samme type data.

API vs. importerbar fil

Kun en av løsningene, Icecat, kunne tilby direkte nedlasting av produktkatalogen. De andre løsningene hadde et API som en må sende forespørsler til for å kunne bruke produktkatalogen. Fordelen med å bruke en maskinlesbar fil er at hele katalogen kan importeres inn i egen database. På denne måten unngår man å ha en ekstern avhengighet (tilbyderens server), noe som vil gi dårligere responstid og mulig nedetid dersom API-serveren har problemer. Bruken av API er også mindre fleksibelt, ettersom vi ikke vil ha direkte tilgang til produktdataen. Ulempen er at vi må bruke tid på å skrive et script som importerer produktkatalogen, og strukturere databasen for å holde hvert produkt.

4.4.3 Konklusjon

Vi valgte til slutt å gå for Icecat. Hovedgrunnen til dette var at dataene var av høyere kvalitet, mer standardisert, og at vi kunne slippe å være avhengig av et eksternt API.

4.5 Technical memo: Android eller kryssplattform

4.5.1 Bakgrunn

Oppdragsgiveren ønsket seg en applikasjon for Android og iOS. I planleggingsfasen kom vi raskt fram til at to native applikasjoner for Android og iOS ville være urealistisk. Grunnen til dette var begrenset tid, erfaring og ingen lett tilgang på en maskin med MacOS, noe som er et krav for å kunne utvikle native applikasjoner for iOS. Vi endte derfor opp med å se etter en kryssplattformløsning, noe som krevde et rammeverk som React Native[47] eller Xamarin[48]. Alternativet til dette var å lage en mer fullverdig native applikasjon kun på Android. Dette ville gitt oss mer tid til å fokusere på ren funksjonalitet rundt løsningen. Det var også en diskusjon om å utvikle en Progressive Web App [49], men på grunn av usikkerhet rundt kompleksiteten til applikasjonen og hvilke funksjonaliteter på mobilen vi trengte tilgang til, så valgte vi å fokusere på en native applikasjon.

4.5.2 Diskusjon

Tidligere kunnskap

Haakon R. hadde allerede tidligere erfaring med utvikling av native androidapplikasjoner fra et emne fra studiet (Mobile Development). Resten av gruppen skulle i dette semesteret ha samme emne. Alle i gruppen hadde også erfaring med Java fra et annet emne som de hadde høsten 2016. Det var derimot ingen i gruppen som hadde noe erfaring med verken React Native eller Xamarin. Vi hadde altså måttet brukt mer forarbeid for å ha settet oss inn i de nye rammeverkene.

Tilgang til støtte

Ved å lage en native applikasjon for Android vil det være mer støtte å finne gjennom internett og dokumentasjonen, ettersom det er mer brukt enn kryssplattformrammeverk.

Oppdragsgiver

En viktig faktor i diskusjonen var oppdragsgivers ønsker. De ville helst ha en kryssplattform applikasjon, men kunne også se fordeler med å kun fokusere på Android.

Begrenset med tid

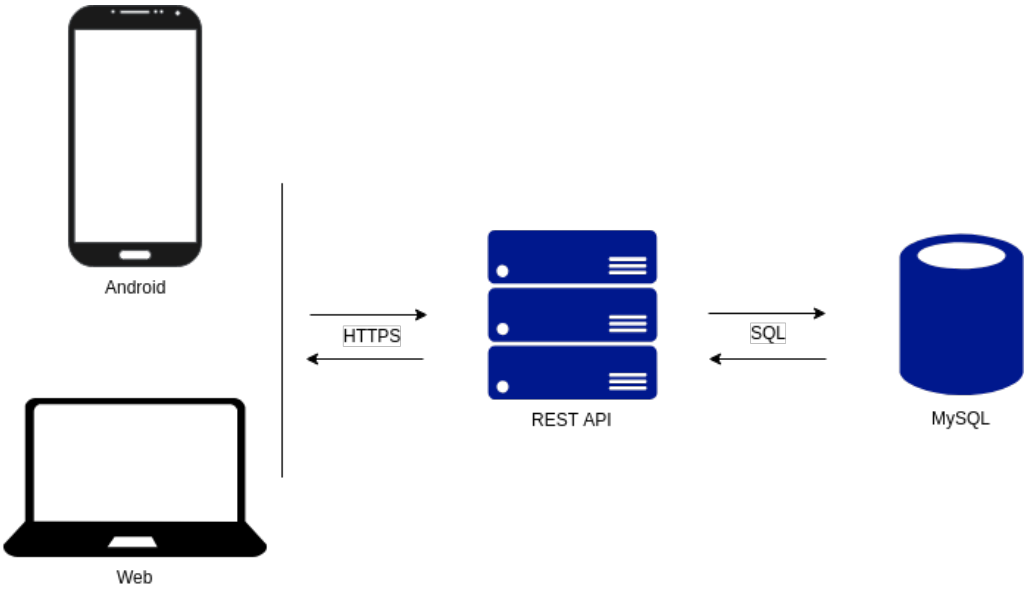
Mye av kravspesifikasjonen var utarbeidet da denne diskusjonen tok sted. Med andre ord visste vi for det meste hva som måtte være på plass til levering, men det var også en del usikkerhet rundt den fulle løsningen. Spesifikt visste vi ikke på dette tidspunktet hvilken tredjepart tilbyder vi kunne bruke for å skaffe en etablert produktdatabase, eller hvor vanskelig det ville bli å implementere PRS. Det vil derfor være mulig at vi må begrense bredden på oppgaven hvis vi går for å lage en kryssplatform applikasjon.

4.5.3 Konklusjon

Basert på punktene over valgte vi å utvikle en native applikasjon for Android i Java. Dette var mest på grunn av at oppdragsgiver ønsket en mer ferdigutviklet prototype som kunne omskrives til iOS på et senere tidspunkt. På grunn av veldig lav markedsandel[50] var det aldri snakk om å utvikle for Windows Mobile.

5 Design

5.1 Arkitektur



Figur 8: Arkitektur for Reviewsers

Den overordnede arkitekturen for Reviewsers består av tre lag. Den første er klienten, som kommuniserer med backend serveren for å hente ut og oppdatere data. Klientene i Reviewsers er Android- og webapplikasjonen som er ”tynne” klienter med tanke på at applikasjonene stort sett tar seg av det grafiske grensesnittet og visning av data, og hvor prosessering og behandling av data foregår på backend. Det lagres også lite data på klienten, med unntak av en token for brukerautentisering. Androidapplikasjonen kommuniserer med backend gjennom HTTP-protokollen ved bruk av HTTP-verb som *GET*, *POST* og *DELETE*. Backend kommuniserer på sin side med en MariaDB database, hvor all informasjon som klienten får fra backend hentes fra. Dette gir en separasjon mellom klient og database, som er viktig for å ivareta sikkerheten ved at den meste av forretningslogikken ligger på server, fri fra manipulasjon på klientsiden.

5.2 Backend

5.2.1 Valg av arkitektur

Innenfor API-utvikling er det i hovedsak arkitekturene REST og SOAP som blir brukt. SOAP er den arkitekturen som tradisjonelt sett har vært mest brukt, men i nyere tid har REST tatt over den posisjonen og er nå brukt av ca. 70% av web API’er [51]. Fordelen med REST er at det er en enklere arkitektur å forstå og lære seg, ettersom man i REST kun bruker et enkelt sett med operasjoner som støttes av HTTP-protokollen for å

aksessere ressurser. SOAP bruker et mer komplisert system basert på XML dokumenter. En annen forskjell er at forespørslene i REST er *stateless*, imens de i SOAP er *stateful*. Dette betyr at REST kan tilby bedre ytelse gjennom mellomlagring av dataene mottatt i forespørsler[51]. Vi har valgt å bruke REST-arkitekturen for API'et på backend. Vi valgte dette fordi det er en enkel arkitektur å implementere og forstå, og det gir oss ett standard sett med operasjoner (CRUD) for klienten mot API'et.

5.2.2 REST-arkitekturen

Representational State Transfer (REST) er en arkitektur for webtjenester som bruker HTTP-protokollen, først beskrevet i 2000 av Roy Fiedling[52]. Arkitekturen definerer et sett med prinsipper som lar klienter hente ut og oppdatere tekstbaserte ressurser fra en webtjeneste, og sikrer interoperabilitet på tvers av systemer. Alle operasjoner i en webtjeneste som bruker REST-arkitekturen (en *RESTful* tjeneste), er *stateless*. Det vil si at enhver forespørsel til server må inneholde all nødvendig informasjon for at forespørselen skal behandles, og forespørsler skal kunne utføres uavhengig av hverandre. Tilstanden til klientene må derfor holdes lagret på klientsiden.

Når REST-arkitekturen brukes for et web-API som vårt eget, er det viktig å ha støtte for fire generelle operasjoner som klienten kan bruke for å aksessere dataene på serveren. Disse operasjonene blir ofte kalt CRUD (Create, Read, Update, Delete). Hver av disse operasjonene kan korrespondere til *metoder* i HTTP-protokollen, hvor *Create* og *Update* er *POST*, *Read* er *GET*, og *Delete* er *DELETE* (samme som i CRUD)[7]. Fordelen med dette er at vi får et standard sett med operasjoner som tett kan "mappes" opp mot SQL spørringer i databasen. Det betyr f.eks at en API operasjon kalt *"GET user"* da vil kunne gjøres om til SQL-setningen *"SELECT * FROM user"*. I dette eksempelet korresponderer HTTP-verbet *GET* til operasjonen *SELECT* i SQL.

Her har vi laget en tabell som viser hvordan hver CRUD-operasjon korresponderer til ett HTTP-verb (hvordan klient kommuniserer med backend) og en SQL-setning (hvordan backend kommuniserer med database):

CRUD	HTTP	SQL
Create	POST	INSERT
Read	GET	SELECT
Update	POST	UPDATE
Delete	DELETE	DELETE

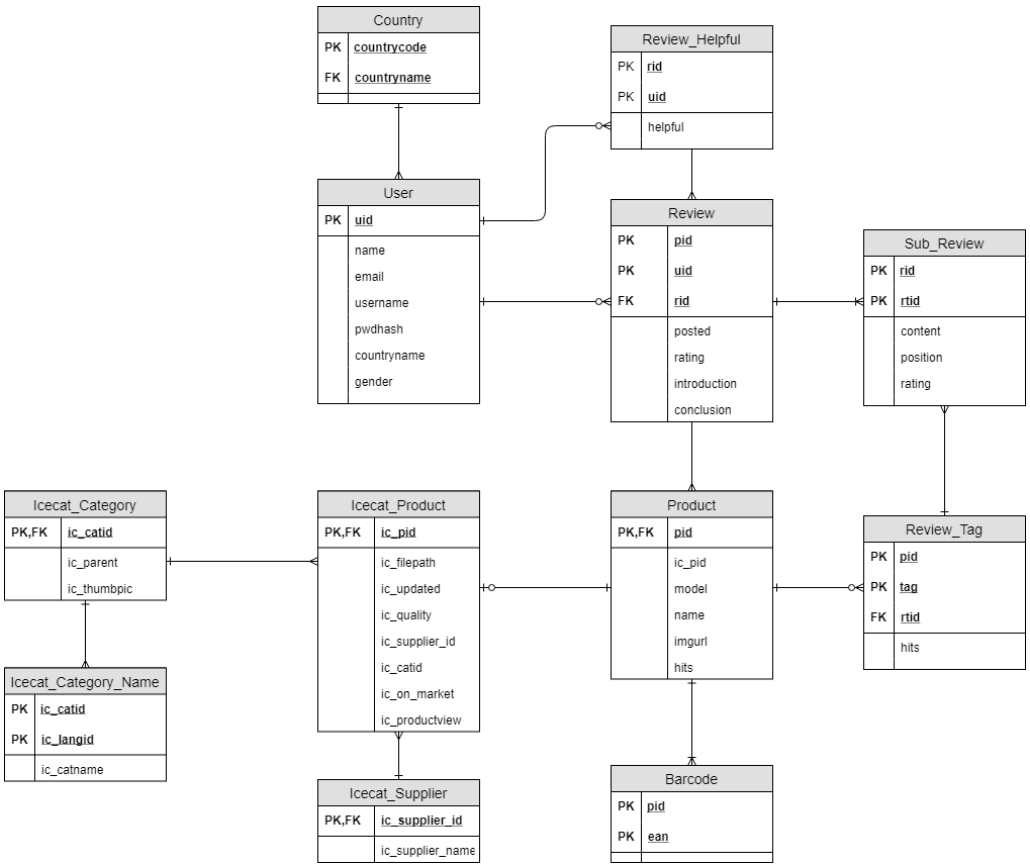
Tabell 3: Sammenligning av CRUD operasjoner i HTTP og SQL

5.2.3 Database

I denne seksjonen beskriver vi litt rundt hvordan vi valgte å modellere databasen. Som forventet består hovedsakelig databasen av tabeller som har med brukere, produkter og anmeldelser å gjøre.

Noe viktig å nevne her er at produkttabellene er delt opp i to forskjellige "deler". Den ene er hovedtabellen "Product". Alle produkter som ligger i databasen er registrert i denne tabellen, som inneholder enkel data som navn og produkt ID. Den andre delen er tabeller som starter med "Icecat_". Dataene i disse tabellene består av dataen vi

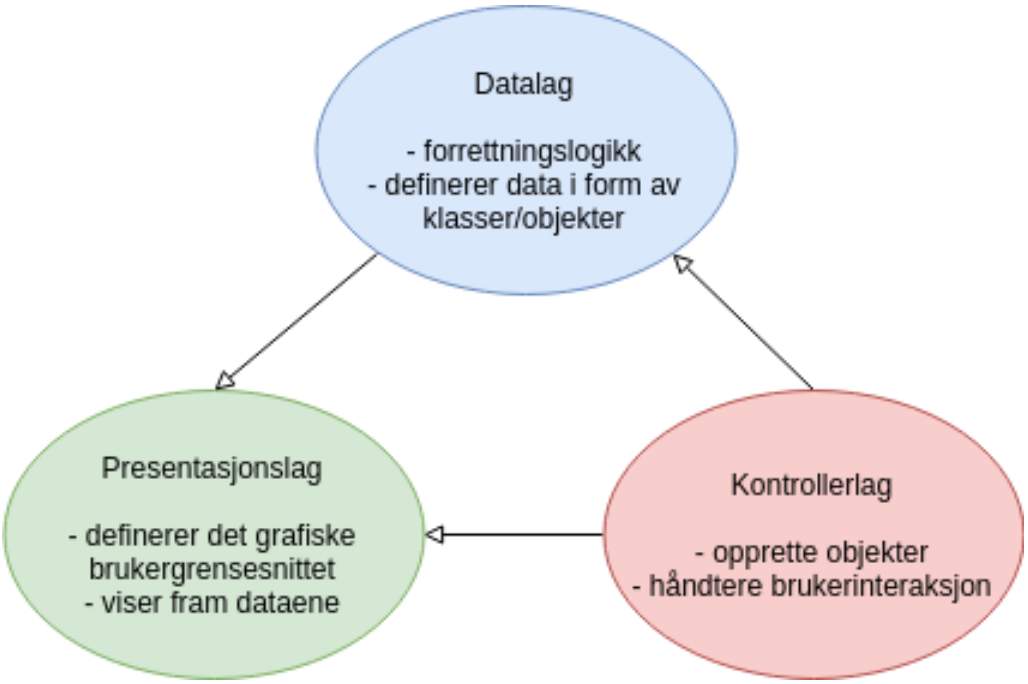
importerer fra Icecat. Ved å dele opp databasen slik, med en generell produkttabell og adskilte tabeller for produktkatalogen til Icecat, sikrer vi at det er lett å ekspandere produktdatabasen senere med produkter innlagt av brukere eller andre produktkataloger som importeres.



Figur 9: Databasemodell for Reviewsers

5.3 Android

5.3.1 MVC



Figur 10: Oversikt over MVC-arkitekturen

Utviklingsmiljøet i Android er lagt opp til at bruk av MVC (Model View Controller) arkitekturen skal være enkelt. I denne arkitekturen skiller man mellom datalaget *model* og presentasjonslaget *view*. I MVC er datalaget ansvarlig for å holde systemets *tilstand*, altså dataene, samt applikasjonens forretningslogikk (dvs. kode som behandler dataene). Presentasjonslaget er ansvarlig for alt som har med det grafiske brukergrensesnittet å gjøre. For å holde disse to lagene adskilt fra hverandre brukes et eget kontroller-lag, som er ansvarlig for å opprette objekter med data og håndtering av brukerinteraksjonen med grensesnittet, som når brukeren trykker på en knapp eller skriver inn tekst[53].

Grunnen til at denne arkitekturen egner seg for utvikling på Android er hvordan Android-arkitekturen er lagt opp. Presentasjonslaget kan enkelt defineres ved hjelp av *layouts*, hvor applikasjonens utseende konstrueres via XML, som ikke er veldig ulik måten nettsider lages gjennom HTML. Videre har Android konseptet om *activites* og *fragments*, som styrer brukerinteraksjonen i applikasjonen[54]. I Android kan *layouts* konstruert med XML ansees som presentasjonslaget, en *activity* som kontroller-laget, og annen Java-kode bortsett fra disse vil da fungere som datalaget.

Nå vil vi forklare kort hvordan vi har brukt MVC-arkitekturen i Android applikasjonen. Henholdsvis presentasjonslaget, datalaget, og kontrollerlaget.

Datalaget

Mesteparten av dataene som brukes i applikasjonen er definert som klasser med tilhørende datatyper som *integer*, *String* osv. Disse klassene har også såkalte *getter* og *setter*-metoder som andre klasser kan bruke for å aksessere dataene. Med slike metoder sikrer

man at dataene ikke aksesseres direkte, og metodene kan utvides med ekstra validering e.l senere dersom det er nødvendig. Her er et eksempel på hvordan en anmeldelse representeres med et utdrag fra klassen *ReviewModel*:

```
public class ReviewModel implements Serializable {
...
    private String username;
    private String date;
    private String introduction;
    private String conclusion;
    private int rating;
...
     public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }
...
}
```

Presentasjonslaget

Presentasjonslaget representeres gjennom *layouts* i Android som er XML-filer hvor applikasjonens GUI er definert[54]. Her er et utdrag fra filen *activity_product_view.xml* som er *layout*'en for grensesnittet som vises når brukeren ser på et produkt:

```
...
    <android.support.v7.widget.Toolbar
        android:id="@+id/default_toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        android:elevation="4dp"
        android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
        app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
        tools:ignore="UnusedAttribute" />

    <TextView
        android:id="@+id/product_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textAppearance="@android:style/TextAppearance.Large"
        android:text="@string/action_userinfo"
        android:padding="10dp"
```

```
19          app:layout_constraintLeft_toLeftOf="parent"
20          app:layout_constraintTop_toBottomOf="@id/default_toolbar" />
21  ...
```

Kontrollerlaget

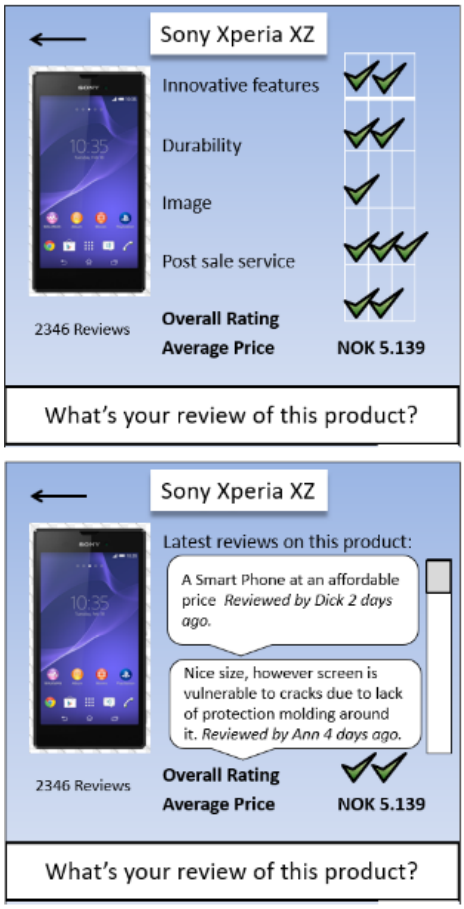
Kontrollerlaget i applikasjonen er koden som ligger i et *Activity*-objekt. Dette er inngangspunktet for applikasjonen (eller for deler av applikasjonen, hvis en *Activity* opprettes av en annen), og siden ett *Activity*-objekt eksisterer gjennom hele applikasjonens livssyklus, egner denne seg for å brukes som kontrollerlag. Her kan vi kalle på metoder i datalaget for å f.eks utføre en API-forespørsel.

5.4 Progressive Review System

5.4.1 Utgangspunkt

Vår oppgave er å lage et sosialt nettverk for anmeldelser. Da sier det seg selv at systemet som brukes for å skrive og lese anmeldelser blir en viktig del av oppgaven. Som nevnt i introduksjon valgte vi tidlig i prosjektet å se nærmere på strukturen av brukeranmeldelser 1.2 og problemer denne strukturen medfører.

Med utgangspunkt i hva oppdragsgiver viste fram utarbeidet vi et eget system for anmeldelser. Oppdragsgiver hadde før prosjektstart utarbeidet et grovt utkast av grensesnittet for hele applikasjonen og dets funksjonalitet.



Figur 11: Utkast fra oppdragsgiver for design av anmeldelser

Grensesnittet inkluderte visjonen deres for hvordan anmeldelser skulle se ut og fungere i applikasjonen. Filosofien var å hele tiden oppfordre brukeren til å skrive anmeldelser, hvor anmeldelsene selv skulle oversiktlig vises til bruker. Inspirasjonen var tatt ifra det tradisjonelle anmeldelsessystemet, hvor det istedenfor stjerner ble brukt haker. Denne hakevurderingen skulle også kunne gis til forskjellige deler av produktet som *innovative features* eller *durability* som kan sees i figuren 11. Systemet er en lignende løsning mer lik anmeldelsesstrukturen vi har diskutert tidligere, med mulighet til å gi noe mer kontekst rundt anmeldelsen ved å gi hakevurdering for flere aspekter av produktet. Det ble derimot tidlig klart for oss at et slikt system ikke ville fungere.

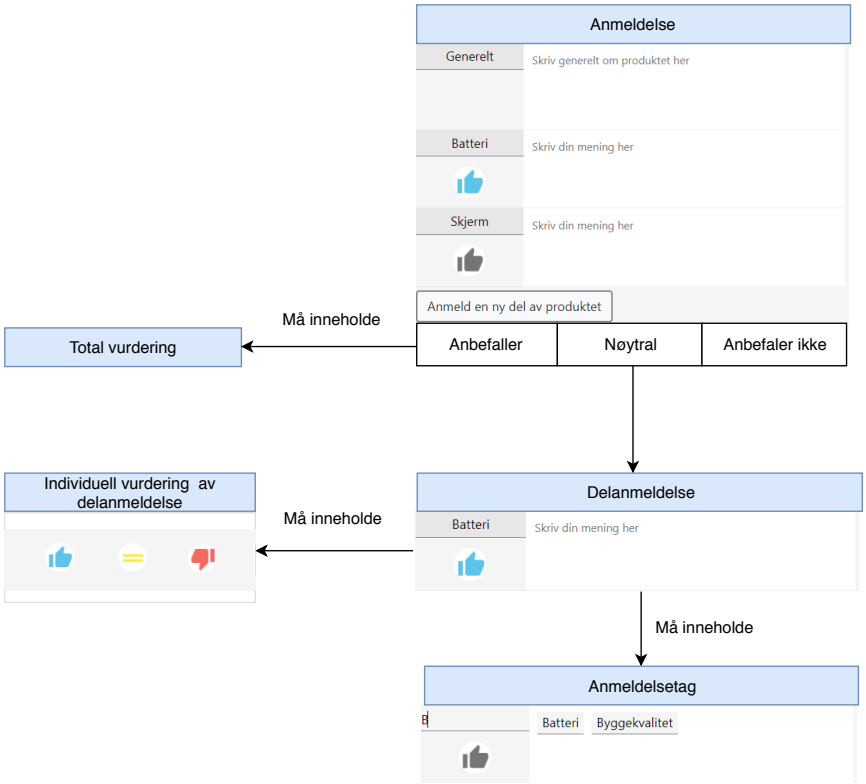
Utfordringen er oppgaven i seg selv. Løsningen skal fungere for alle eksisterende produkter. Dette kan være alt fra klær, elektronikk eller kjøretøy. Med andre ord vil det bli en stor variasjon av mengde og type produkter med hver sine unike karakteristikker. Denne variasjonen vil gjøre det vanskelig å lage et system hvor vi kan dele opp produkter i sine egenskaper slik som i figuren 11. Hvor et alternativ blir veldig generelle egenskaper som vil passe for mange nok produkter.

Ideen om PRS baserte seg på diskusjoner og samtaler vi hadde hatt oss imellom og med oppdragsgiver om anmeldelser, da spesifikt det tradisjonelle brukeranmeldelsessystemet. Et av gruppemedlemmene kom opp med ideen om å la brukere selv dele opp produktet i flere komponenter. Ideen var inspirert av brukeranmeldelser hvor anmeldere selv strukturerte deres egne anmeldelser ved å dele opp dem opp i flere seksjoner. Her ble det gitt en skriftlig vurdering gitt individuelt for de komponentene anmelderen valg-

te å skrive om. Anmelderen får forklart seg selv og sin vurdering, hvor leseren vil kunne bedre forstå de gode og dårlige egenskapene ved produktet. Det er derimot et fåtall som strukturer sine anmeldelser på denne måten, samtidig som det ofte resulterer i mye tekst som brukeren må lese igjennom. Da ideen ble introdusert foran oppdragsgiver var det lite diskujon, og det ble klart at det skulle implementeres.

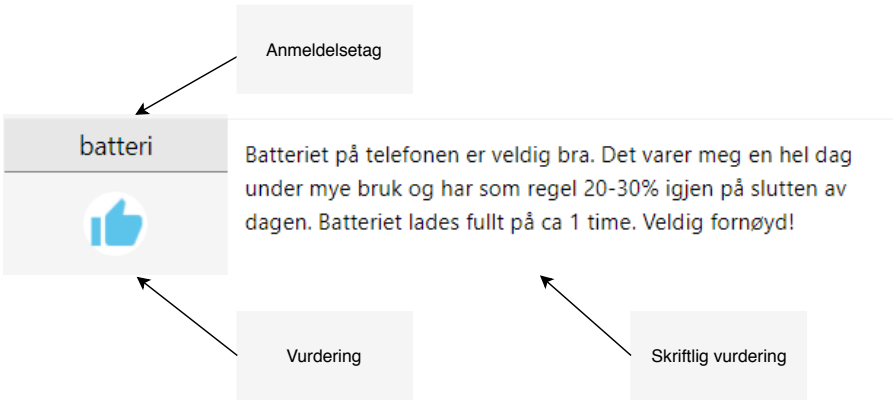
5.4.2 Hvordan PRS strukturerer en anmeldelse

PRS består hovedsakelig av to strukturer. Hvordan anmeldelser skrives og hvordan anmeldelser vises. La oss først snakke om hvordan PRS lar brukeren strukturere en anmeldelse.



Figur 12: Progressive Review System strukturering av anmeldelser. Bilder i modellen er tatt ifra web-applikasjonen

PRS bygger på prinsippet om å la brukeren strukturere sin egen anmeldelse. Systemet gir brukeren verktøyene til å enkelt kunne dele opp anmeldelsen i logiske deler basert på produktet som anmeldes. Dette gjør det også enklere for brukere som normalt sett ikke er erfarne med å anmelde å lage gode oversiktlige anmeldelser. Strukturen på PRS kan ses i figuren over 12. En anmeldelse er delt opp i tre deler. De tre delene er *Generelt*, *Delanmeldelser* og *Vurdering*. *Generelt* brukes til å kunne gi informasjon rundt produktet. Dette kan være informasjon om hvor lenge brukeren har brukt produktet, hvor det ble kjøpt og lignende. Denne delen av anmeldelsen har ingen vurdering, og eksisterer kun for å kunne gi konteksten rundt brukeren og produktet.



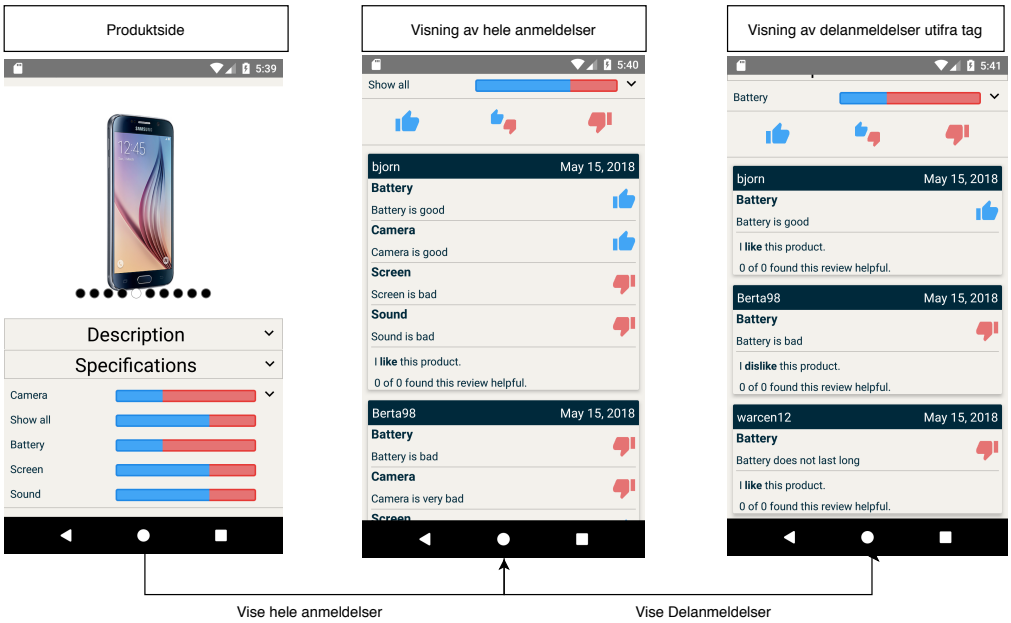
Figur 13: Delanmeldelse. Bilde tatt ifra web-applikasjonen

En anmeldelse består av ingen eller fler *delanmeldelser* . Delanmeldelser er kjernen i PRS og er komponenten som løser mange av problemene nevnt tidligere. En delanmeldelse er en anmeldelse for en spesifikk del av produktet. En del av et produkt kan være skjermen på en mobil, passformen på en skjorte eller bensinbruken på en bil. Det er denne delen som skaper en *anmeldelsestag* . *anmeldelsestag'er* er "taggene" som definerer produktene som blir vurdert. Brukere står fritt til å kunne lage egne tagger for å dele opp deres anmeldelser. Taggene vil da kunne dele opp produktet ut ifra dets egenskaper. Når en anmeldelse publiseres lagres nye "tagger" for produktet som ble vurdert. Dette betyr at når en bruker skriver en delanmeldelse kan bruker velge mellom tagger lagt til av andre anmeldere. Når "taggene" vises til bruker er de sortert etter popularitet. Det er på denne måten vi har valgt å løse problemet ved å kunne gi brukeren muligheten til å vurdere forskjellige deler av produktet, uten å forhåndsdefinere produkters unike egenskaper.

Hver delanmeldelse har sin egen vurdering i form av tekst og valgt vurdering, som kan være negativ, nøytral eller positiv. Konseptet med at hver delanmeldelse har en egen vurdering var opprinnelig kun for å gjøre det lettere for å få en bedre oversikt over lange anmeldelser. Det ble derimot en essensiell del av PRS, og ble med på å kjennetegne systemet. Vi snakker mer om dette i seksjonen under.

Hver anmeldelse må ha en egen vurdering. Dette er for å formidle brukerens konklusjon rundt produktet. Vurderingen kan gis i form av *anbefaler*, *nøytral* eller *anbefaler ikke*. En egen tilhørende konklusjon i form av tekst kan legges til på bunnen av anmeldelsen.

5.4.3 Hvordan PRS viser fram anmeldelser



Figur 14: Hvordan PRS viser fram anmeldelser på mobil

Alle anmeldelser for et produkt vises fram på en produktside. Hver anmeldelse vises som en separat enhet hvor generelt, hver delanmeldelse og konklusjon (tekst) utgjør hver del. Brukere kan vurdere anmeldelser som hjelpsomme eller ikke-hjelpsomme. Denne vurderingen blir gjort om til en poengsum som bestemmer rekkefølgen anmeldelsene blir vist på. Det som gjør PRS til en såpass kraftig løsning, er måten anmeldelsene kan deles opp på, og vises fram på en ny måte.

Når en bruker går inn på en produktside, vil det som standard vises hele anmeldelser med en vurderingsbar som viser prosentandelen av positive og negative anmeldelser. Det er her vurderingen på delanmeldelser kommer inn.*Anmeldelsestag'er* er lagret for hvert produkt. Hver tagg er koblet opp mot vurderinger fra brukere som har brukt taggen. Dette gjør det mulig å lage en nedtrekksmeny hvor alle brukerdefinerte tagger vises. Hver tagg har samme vurderingsbar som hele anmeldelser i menyen (se figur 14). Dette gir brukeren direkte oversikt over hvilke deler av produktet som brukere liker, og hvilke de ikke liker, uten å ha lest en eneste anmeldelse. Hvis brukeren trykker på en tagg blir det eksklusivt vist delanmeldelser for denne spesifikke delen av produktet. Hvis bruker trykker på taggen *Skjerm* vil det kun vises delanmeldelser med taggen *Skjerm*. Delanmeldelsene kan igjen filtreres etter vurdering. Med andre ord, hvis brukeren kun ønsker å se delanmeldelser med negativ vurdering, kan brukeren velge å kun vise negative anmeldelser for denne delen av produktet. Akkurat som hele anmeldelser kan delanmeldelser gis en hjelpsomhetsvurdering som er uavhengig av vurderingen gitt for hele anmeldelsen. Grunnlaget for denne funksjonen er at en anmeldelse med mange delanmeldelser kan være en god anmeldelse, men det betyr derimot ikke nødvendigvis at alle delanmeldelsene i anmeldelsen er hjelpsomme.

Vi føler at dette er en veldig god måte å vise fram anmeldelser på. Det løser problemene vi introduserte i starten av rapporten med mangel på struktur i brukeranmeldelser. Samtidig gir vi brukeren en enklere og mer presis måte å se den gjennomsnittlige vurderingen fra andre brukere.

PRS var under konstant utvikling i prosjektet, og det er mange avgjørelser og forandringer vi har gjort i utviklingsprosessen. Da det første utkastet av anmeldelsessystemet var implementert utførte vi en brukertesting av systemet. Det kan leses mer om valg vi tok underveis (se seksjon 7.4).

5.5 Brukergrensesnitt

Under utvikling av androidapplikasjonen har vi valgt å følge et standard designmønster. For Android er Material Design standardløsningen som blir fremmet og støttet av Google[55]. Ved å følge deres retningslinjer har vi avlastet valg om utseende og form fra oss, og heller fokusert på applikasjonen som en helhet. I tillegg har vi fått med oss en stor verktøykasse som vi kunne plukke fritt fra for å lage en løsning for PRS på Android uten større problemer.

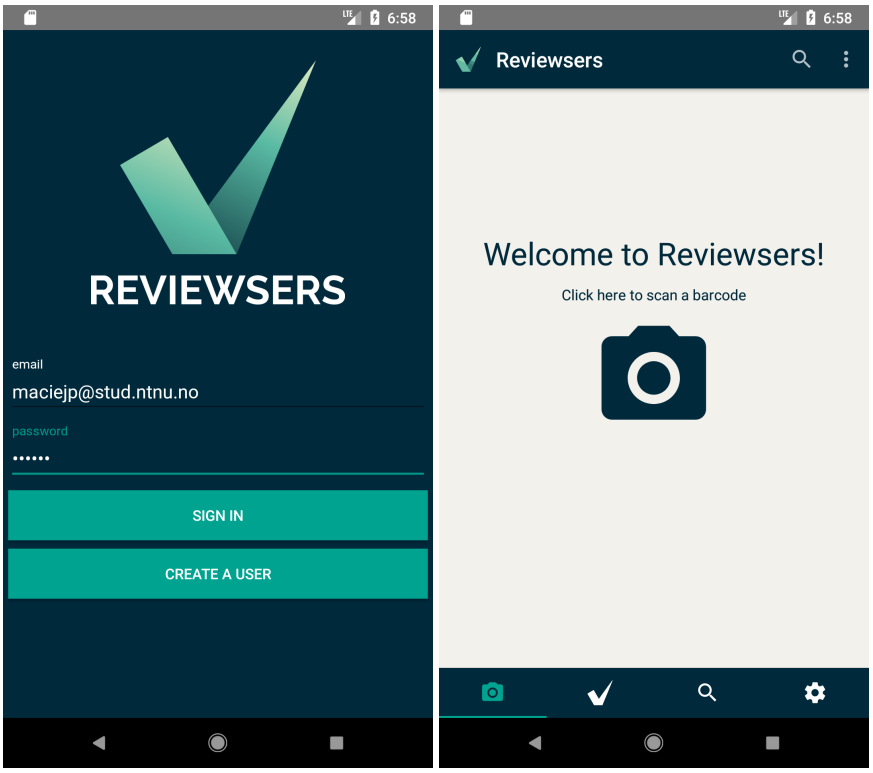
5.5.1 Fragments og Activity

Utvikling av en androidapplikasjon vil i stor grad handle om hvordan man velger å strukturere brukergrensesnittet. Alle androidapplikasjoner må ha minst *en* såkalt *Activity*[56], med tilhørende kode, logikk og brukergrensesnitt relatert til activity'en. *Activities* kan ikke kombineres med hverandre, noe som gjør gjenbruk av brukergrensesnittet vanskelig. For å løse dette, tilbyr Android også *fragments* , som er gjenbrukbare komponenter i en *activity*. Disse kan plasseres i forskjellige kombinasjoner[57]. Dette krever at fragment'et kan håndtere flere mulige plasser den kan være plassert på, men dersom man skal gjenbruke et mindre element flere plasser, slipper man unødvendig duplisering av brukergrensesnitt.

I applikasjonen har vi gått ut i fra at de aller fleste elementene vil være satt inn som fragments, som vil erstattes når brukeren trykker seg videre inn i applikasjonen. Det tillater oss å gjenbruke elementene andre plasser om det blir mulig. Under utviklingen har vi ikke tatt i betrakning layout for nettbrett eller horisontal visning, men bruk av fragments vil gjøre det mulig å vise to forskjellige deler av applikasjonen ved siden av hverandre. Det kan være for eksempel søk på produkter på venstre side, og produktinformasjon på høyre.

Oppdeling av applikasjonen gjør det også enklere å separere funksjonalitet slik at hver del er ansvarlig nettopp for det den skal gjøre, og ikke mer. I mange applikasjoner vil det man kaller "Main Activity", det første vinduet som kommer opp når man starter applikasjonen, trenger en god del kode for å håndtere overgang til de forskjellige andre delene, i tillegg til å ha sin egen funksjonalitet på plass i samme klasse. Ved å separere styringen av hvilke GUI-elementer som vises fra funksjonaliteten til Main Activity, kan man holde koden mer ryddig og isolert til bare et mål.

5.5.2 Brukergrensesnitt Android

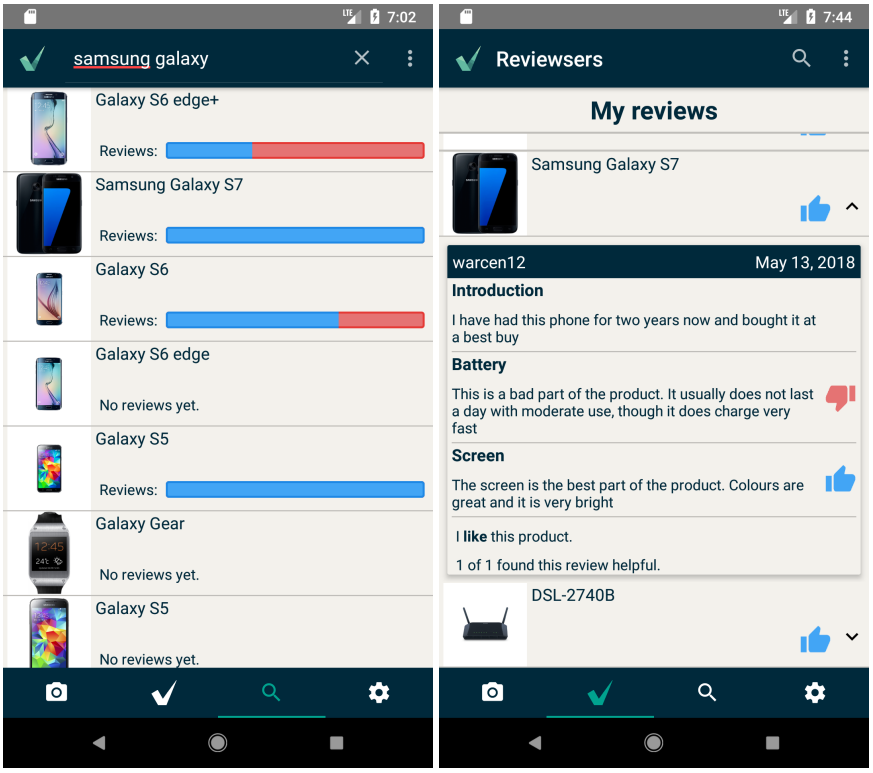


(a) Innloggingssiden (b) Hjemsiden

Figur 15: Bilder fra androidapplikasjonen

Selv om applikasjonen er tilgjengelig for alle til å laste ned, betyr ikke det at den gir like muligheter for bruk. Dersom fargevalget i applikasjonen er dårlig, kan det hende at noen brukere ikke klarer å se forskjellen på de forskjellige elementene. Når det skjer vil brukere få problemer med å bruke applikasjonen. Alle gruppemedlemmer har liten erfaring med valg av farger bortsett fra grunnleggende kunnskap om hvor viktig det er med gode kontraster. I dette tilfellet har oppdragsgiver stilt med en profesjonell spesialist i design som har utformet et utkast for fargepalett som vi kunne benytte i applikasjonen. Sammen med utkast for et fargepalett var det laget logoer for applikasjonen.

Pallettet var derimot basert på oppdragsgivers eget utkast for design, som var laget før oppgavestart. Fargevalget vi ble gitt fulgte ikke regler for Material Design, som spesifiserer et par farger man bruker rundt i applikasjonen, med mørke og lyse variasjoner av disse[58]. For å følge Material Design har vi da valgt å benytte noen av fargene vi fikk, som vi da lagde variasjoner av ved å bruke Googles verktøy[59]. Den mørkeblå fargen vi benytter er da primærfargen i applikasjonen, som er ment til å være en ramme rundt applikasjonvinduet, og en mer grå farge som blir sekundærfargen vi bruker til å vise frem innholdet i vinduet. I tillegg benytter vi en grønn farge på alle plasser hvor brukeren aktivt skriver inn tekst. Logoen oppdragsgiver ga oss har vi implementert i brukergrensesnittet på mobil som kan sees i bildene over (15).



(a) Søkesiden (b) Brukerens egne anmeldelser

Figur 16: Bilder fra androidapplikasjonen

For å gjøre applikasjonen enkel og intuitiv i bruk, har vi valgt å følge løsninger vi har sett bli brukt av andre populære applikasjoner. Søkeelementet holdes i en plass i verktøykassen på toppen av applikasjonen. Hovedløsningen for å finne konsumentvarer i applikasjonen er skanning av strekkoder, noe som er fremhevet på hjemsiden og som er enkelt tilgjengelig for brukeren. De forskjellige funksjonene vi tilbyr ligger også enkelt og intuitivt på bunnen av hjemsiden som "tabber", og kan enkelt byttes mellom. Når det gjelder implementasjon av PRS på Android, har vi laget en enkel løsning som implementerer konseptet vi utarbeidet før utviklingen, hvor vi også kom opp med ekstra funksjonalitet 5.4. Skjermbilder av hjemsiden og anmeldelsessystemet kan sees i figur 15. I tillegg ligger skjermbilder av det meste i applikasjonen lagt til i vedlegg G.

Dersom brukeren ikke klarer å forstå informasjonen som blir presentert, vil brukeren ikke få nytte av løsningen. For å løse dette har vi valgt å vise minimalt med overflødig informasjon til brukeren. Når brukeren åpner siden for et konsumentprodukt, vil beskrivelsen og listen over spesifikasjoner holdes gjemt bak knapper, som kan vise den ekstra informasjonen ved klikk. Dette fører til at brukeren ikke blir overbelastet med informasjon. Når det gjelder anmeldelser, kan disse enkelt filtreres som brukeren skulle ønske, med kun et tastetrykk. Hvis brukeren liker anmeldelsen, kan den åpnes ved å trykke på og velge visning av hele anmeldelsen.

Vi har aktivt også sett på måter vi kan standardisere og gjenbruke elementer vi bruker flere plasser. Vi valgte da å benytte fragments for det meste av brukergrensesnittet, som plasseres i en container. På denne måten kan for eksempel fragmentet som viser bru-

keranmeldelser gjenbrukes når man viser andre brukers anmeldelser i en annen activity. Slik gjenbruk har ført til standardisering av utseende i applikasjonen på de forskjellige sidene, og gjorde det enklere å forandre elementene flere steder samtidig.

6 Implementasjon

Dette kapittelet beskriver de mer tekniske detaljene når det kommer til implementasjon, som verktøy vi har valgt å bruke under utvikling, og eksempler med forklaring som viser hvordan vi implementerte funksjonaliteten i kode.

6.1 Utviklingsmiljø

For å holde koden organisert under utviklingen av de forskjellige løsningene, har vi brukt versjonskontrollverktøyet Git[60]. Det er en populær open-source løsning originalt utviklet av Linus Torvalds for bedre håndtering av Linux sin kildekode. På dette tidspunktet fantes det ingen løsning som kunne tilfredsstille hans krav for håndtering av kildekode[61]. Løsningen er benyttet i de fleste programmeringsemnene på NTNU i Gjøvik, og er det verktøyet som gruppen hadde mest erfaring med. Valget ble tatt som en selvfølge uten diskusjon.

I løpet av utviklingprosessen har det også raskt kommet opp problemer med såkalt ”merging”, som kan oppstå når forandringer har blitt gjort på samme sted i samme kodefil. Git vil da ikke automatisk kunne finne ut hvordan å kombinere forandringene. For å håndtere problemet har vi aktivt benyttet såkalte *feature branch*’er. Det vil si at vi lagde en egen *branch* for hver nye funksjonalitet som utvikles, istedenfor å utvikle direkte på *master*, som er hoved ”branchen”[62]. Dette minker problemet med *merging* ved å la hvert medlem i gruppen arbeide uforstyrret fra andre på gruppen med arbeidet sitt, for deretter å se på problemene sammen når vi hadde møte.

Når man bruker Git, har man kode som lagres på en felles server og utviklernes lokale arbeidsmaskiner. Det er mange tjenester som tilbyr plass på sine serverene for programmeringsprosjekter, men med mange er det lite mulighet for å gjøre koden privat gratis. Atlassian tilbyr gratis student-pakke på sin tjeneste ”Bitbucket”, som blant annet tillater studenter å holde koden sin privat[63]. Ettersom vi alle har brukt Bitbucket før i andre emner, og Atlassian-verktøy er aktivt brukt av NTNU, har vi tatt den løsningen i bruk.

Et annet Atlassian-verktøy som vi tok i bruk for arbeid på alle løsningene er ”issue-tracking” programvaren Jira. En ”issue-tracker” hjelper utviklere å organisere prosjektet ved å holde styr på oppgaver og dokumentasjon av problemer i prosjektet. Jira er en av de mest populære slike løsningene, ifølge VersionOne sin årlige rapport er Jira den mest brukte løsningen på markedet når det gjelder utviklingsprosess-verktøy, med 54% bruk[64]. Jira har også støtte for å sette opp Scrum, med tilhørende task’er og sprinter. Jira er en betalt løsning vi fikk tilgang til gjennom NTNU sin egen Jira server. Alle i gruppen hadde erfaring med Jira fra før, noe som gjorde det enkelt å ta i bruk under bacheloroppgaven.

For håndtering av notater har vi tatt i bruk Google Docs[65]. Verktøyet ble valgt foran diverse andre løsninger på grunn av at man trenger minimalt oppsett for å kunne dele notater aktivt med andre, og kunne se forandringene i sanntid. For rapportskriving har vi valgt å benytte ShareLaTeX[66]. På likhet med Google Docs tilbyr denne også redigering

i sanntid. Dette gjorde det enkelt å skrive rapporten felles ved å gi en oversikt over hva andre i gruppen arbeidet med.

6.1.1 Android

For utvikling av androidløsningen har vi tatt i bruk det integrerte utviklingsmiljøet Android Studio. Det er det offisielle [67] utviklingsverktøyet tilbudt av Google som har hovedansvaret for Android. Løsningen tilbyr mange verktøy som gjør utvikling for Android enklere. Blant disse som tidligere nevnt er en emulator som simulerer en Androidtelefon. Dette gjør det mulig å utvikle for Android uten å måtte bruke en Android enhet og vil gjøre det mulig å teste applikasjonen på eldre Android versjoner for å sikre kompatibilitet. Ettersom verktøyet også er benyttet i emnet "Mobile Development", som to på gruppen tar parallelt med bacheloroppgaven, var det naturlig å ta dette i bruk.

For å kunne utvikle en applikasjon for Android, må man også ta i bruk et byggeautomatiserings verktøy, ettersom mange verktøy rundt Android er hele tiden under utvikling. Et byggeverktøy automatiserer byggeprosessen ved å hente biblioteker fra internett, kompilerer, og holder prosjektet oppdatert for alle utviklerne. Gradle er standard byggeverktøyet for Android, og det var derfor ikke noe vurdering å velge noe annet. Android Studio setter automatisk opp Gradle for prosjekter, mens hovedalternativet Maven krever innsats fra brukere for å få på plass[68].

6.1.2 Backend

For koding av backendløsningen brukte vi koderedigeringsverktøyet Visual Studio Code. Dette er et open-source verktøy fra Microsoft som ikke er ment for et spesifikt programmeringsspråk, men som kan tilpasses mange forskjellige utviklingsscenarier ved hjelp av utvidelser. Noe som gjorde det gunstig å bruke er at den har innebygget støtte for automatisk kodefullføring for JavaScript og Node.js[69].

Testserver

For å kunne teste ut backendløsningen i et realistisk miljø over internett, trengte vi å ha tilgang på en testserver. I starten vurderte vi å bare teste med en lokal server kjørende på hver vår maskin, men fant fort ut at det var mer hensiktsmessig å ha en felles server som alle prosjektmedlemmene bruker, slik at vi brukte et felles testmiljø.

Oppdragsgiveren stile til disposisjon med en såkalt VPS (*Virtual Private Server*), som gir full administratortilgang til serveren via en kryptert tilkobling (SSH). Serveren det er snakk om kjører Linux operativsystem, med distribusjon Ubuntu 16.04.

Deployment Script

For å forenkle prosessen med utrulling til testserveren etter kodeforandringer, bestemte vi oss for å lage et eget script i bash som automatiserer prosessen. Dette gjorde vi for å spare tid hver gang vi gjorde en forandring på master-branchen, hvor vi måtte oppdatere backend til nyeste versjon på testserveren. Oppgaven til dette scriptet er å først slette den eksisterende databasen. Deretter opprettes en ny database med tilhørende bruker, og selve databasestrukturen (i form av SQL spørringer i filen *database.sql* importeres. Etter dette kjøres scriptet for å importere produktkatalogen til Icecat (se seksjon 6.2.1).

Prosedyren som måtte utføres etter en oppdatering av backend, ble da først å logge seg inn på serveren gjennom SSH, kjøre *git clone* for å hente nyeste versjon, og kjøre

dette scriptet. Scriptet er vedlagt i kodeeksempel 21.

6.2 Backend

6.2.1 Import av produktkatalog

Selve produktkatalogen med data vi laster ned fra Icecat kommer i form av et stort XML dokument. Vi måtte derfor lage et script som importerer hele denne produktkatalogen inn i databasen vår, slik at vi kunne få brukt dem i løsningen vår.

Et av problemene her var størrelsen på filen, som lå på over 600Mb. For å forstå hvorfor dette kan være et problem må en forstå at det finnes to forskjellige typer *parser*'e for XML. Med *parser* menes her programvaren som tyder filen. Den første typen er *DOM* (Document Object Model) basert. Disse kan bruke veldig mye minne dersom filen er stor, fordi den må lese inn hele filen i minnet og i tillegg holde styr på tilstanden om hvor pekeren er i dokumentet. Alternativet til dette er en såkalt *SAX* (Simple API for XML) basert parser. Dette er en svært enkel parser, som kun genererer *events* (hendelser) når pekeren kommer over en ny node i XML-dokumentet. Fordelen med å bruke en SAX-basert parser er at den kun trenger å lese en linje om gangen fra XML dokumentet inn i minnet. På denne måten bruker en slik parser nærmest ikke noe minne, og er også mye raskere. Ulempen er at SAX er vanskeligere å bruke enn DOM, fordi utvikleren selv må holde styr på tilstanden ved hjelp av egendefinert kode og variabler[70].

Vi fant fort ut at det ikke var noe reellt alternativ annet enn å velge en SAX-basert parser i vårt tilfelle. Dette var hovedsakelig på grunn av filstørrelsen. Testserveren hadde kun 1GB med minne, så det veide tungt mot SAX.

Scriptet som importerer produktkatalogen fra Icecat ble på likhet med API'et skrevet i Node.js. Vi brukte open-source modulen *sax-js*, som er tilgjengelig via GitHub, for parsing av produktkatalogen.

Et lite utdrag av scriptet vises under. Funksjonen *saxStream.on*, registrerer en callback funksjon, som kalles når pekeren kommer over en ny node i dokumentet. Hvis denne noden heter "file" (som er et produkt), kalles igjen funksjonen *insertProduct* med nodens attributter (som inneholder produktinformasjon). Denne funksjonen er ansvarlig for å legge til produktet i databasen. Variabelen *currentProduct* brukes for å holde styr på hvilket produkt som leses inn. Når pekeren kommer over noden "EAN_UPC" (EAN-13 koden for et produkt), kan vi da legge til strekkodene for produktet i databasen.

```
1 saxStream.on('opentag', (node) => {
2     if (node.name === 'file') {
3         insertProduct(node.attributes);
4         currentProduct = parseInt(node.attributes['Product_ID']);
5     }
6     if (node.name === 'EAN_UPC') {
7         insertBarcode(node.attributes, currentProduct);
8     }
9 });
```

Listing 1: Utdrag fra scriptet som importerer produktkatalogen fra Icecat

6.2.2 Kommunuikasjon med klient

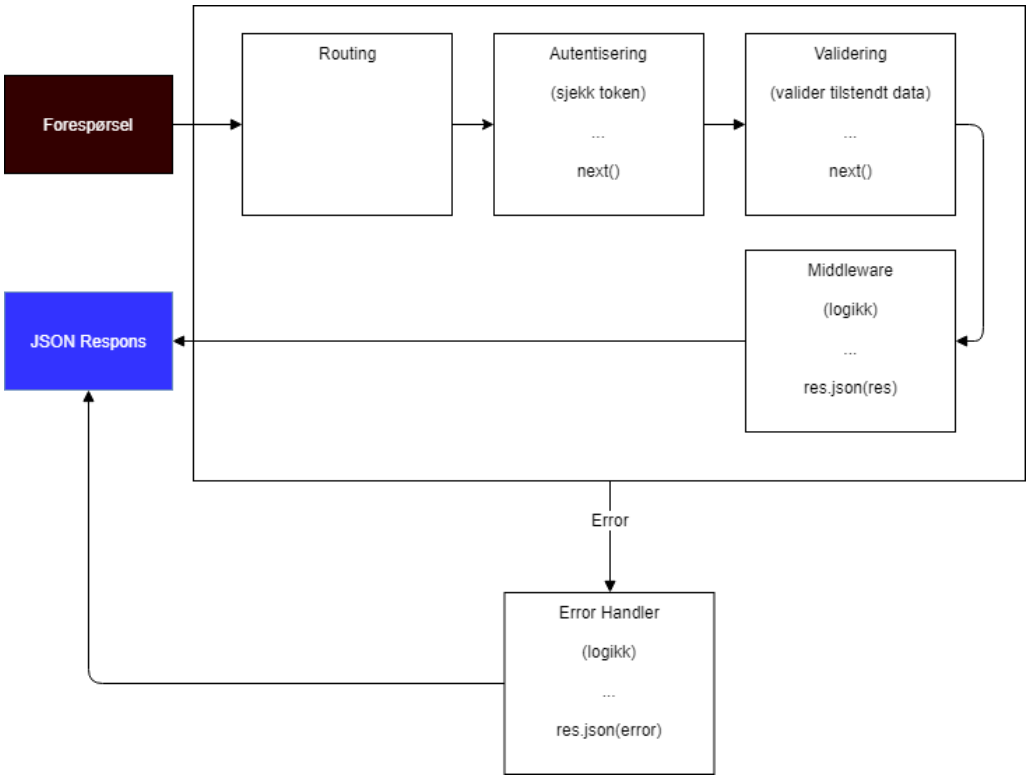
Formatet på dataene som blir sendt i forespørsler, og respons fra backend, er i form av JSON. På klientsiden er nøkkel-verdi parene sendt i en forespørsel som ikke er i JSON-format (vanlige HTTP-forespørsler), men gjøres om til JSON på serversiden ved hjelp av Express og modulen *body-parser*[71]. På klientsiden brukes *org.json*[72] biblioteket fra Android SDK'et for å håndtere en JSON respons.

Et eksempel på en respons fra backend vises under, hvor anmeldelser fra brukeren *Testbruker123* hentes:

```
{
    "pages": 1,
    "results": [
        {
            "username": "Testbruker123",
            "pid": 864029,
            "name": "Galaxy S6 edge+",
            "imgurl": "http://images.icecat.biz/img/gallery
            /40097813_9018818911.jpg",
            "posted": "2018-04-27T08:33:04.000Z",
            "rating": 1,
            "general": "Dette er generelt",
            "conclusion": null,
            "numHelpful": 0,
            "numRatings": 0,
            "sub_reviews": [
                {
                    "tag": "Skjerm",
                    "position": 1,
                    "rating": 1,
                    "content": "Mobilen har veldig bra skjerm"
                },
                {
                    "tag": "Ytelse",
                    "position": 2,
                    "rating": 1,
                    "content": "Jeg er fornøyd med ytelsen. Her går det unna."
                }
            ]
        }
    ]
}
```

Listing 2: Eksempel på respons fra API

6.2.3 Kommunikasjonspipeline



Figur 17: Viser pipeline for en request til API

Måten en forespørsel til backend behandles på kan beskrives som en *pipeline* (vist i figur 17, hvor forespørselen går gjennom flere steg før en respons blir sendt tilbake til klienten. Hvis et av disse stegene ikke passerer vil det istedet bli returnert en feilmelding som respons, som indikerer hva som gikk feil.

Routing

Det første steget som skjer når API'et mottar en forespørsel, er at forespørselen skal "rutes". Dette betyr at backend må finne ut (basert på URL-addressen), hvordan den skal svare på forespørselen. Dette er hovedoppgaven til Express-rammeverket, hvor man kan definere et sett med *routes* og tilhørende callback-funksjoner som kalles når en klient aksesserer en bestemt *route* på serveren[73]. Bruken av Express-rammeverket gjorde at vi ikke trengte å implementere selve nettverkslogikken selv, men heller tenke på resten av logikken, som databasespørringer og lignende. Dette sparte oss en del tid under utvikling av backend.

Når man skal definere *routes* i Express, kaller man metoder på Express sitt *router*-objekt. For å holde oversikt over de mange rutene vi hadde i API'et vårt, definerte vi alle rutene i en egen fil kalt *routes.js*. Et lite utdrag av denne filen vises her:

```
...
/**
 * REVIEW ROUTES
 */
router.get('/review/:productid', validation.getReviewsByProduct,
 ↪  checkValidation, review.getReviewsByProduct);
router.get('/review/user/:userid', validation.getReviewsByUser,
 ↪  checkValidation, review.getReviewsByUser);
router.get('/review/username/:username', validation.getReviewsByUsername,
 ↪  checkValidation, review.getReviewsByUsername);
router.post('/review', validation.createReview, checkValidation,
 ↪  review.createReview);
router.delete('/review/:productid', validation.deleteReview,
 ↪  checkValidation, review.deleteReview);
router.get('/review/tag/:productid', validation.getTags, checkValidation,
 ↪  review.getTags);
router.post('/review/tag', validation.createTag, checkValidation,
 ↪  review.createTag);
router.post('/review/rate', validation.rateReview, checkValidation,
 ↪  review.rateReview);
...
```

Listing 3: Utdrag fra filen *routes.js*

Metoden som kalles på router-objektet, som get(), post() og delete()[74] tar en URL addresse som parameter og en eller flere *middleware* (callbackfunksjoner) som sender forespørselen videre i pipeline'en. Disse rutene registreres da av Express, og blir gjenkjent når en klient prøver å aksessere disse.

Autentisering

Ettersom vi krever at brukere er innlogget mesteparten av tiden når applikasjonen er i bruk (bortsett fra når bruker registreres), måtte vi ha et system for autentisering. Vi bestemte i kravspesifikasjonen at dette skulle foregå via token-basert autentisering. Det vil si at alle brukere får en hemmelig streng som gir dem tilgang til systemet. Det er flere måter å gjøre dette på, men siden vi bruker REST-arkitekturen hadde vi noen spesifikke krav. Dette fordi REST krever at ingen tilstand om klienten skal holdes lagret på server men være inneholdt i forespørselen[52].

Etter å ha undersøkt mulige løsninger, fant vi fram til at JWT (JSON Web Token) ville være en enkel løsning. JSON Web Token er en åpen standard (RFC 7519)[75], og lar en server kryptografisk signere en token. Når en token signeres, genererer serveren en kryptografisk signatur av informasjonen inneholdt i token'et ved hjelp av en nøkkel som kun ligger på serveren og brukerne ikke har tilgang til[75]. På denne måten kan vi forsikre oss om at innholdet i en tilsendt token er korrekt, og ikke er forandret på etter at den var utsendt. En slik token består av en *header* som inneholder informasjon om hashingalgoritmen som er brukt, en *payload* med informasjon om utsteder, utløpstid, og annen meta-data. Til slutt er det en *signatur* som verifiserer dataenes integritet ved hjelp av hashingalgoritmen[75]. Dette token'et kan omkodes til en kompakt *base64* streng på formatet *xxxxx.yyyyy.zzzzz*[75]. *Base64* er en metode for å kode binær data om til en

tekststreng[76]. Dette gjør at den kan inkluderes i hver enkel spørring til API'et, som da vil gjøre at hele klientens *state* er holdt i forespørselen. I vårt tilfelle valgte vi å legge ved dette token'et i et HTTP header felt kalt *x-access-token*.

For å slippe implementasjon av JWT-standarden selv, brukte vi Node.js modulen *express-jwt-token* som tilbyr autentisering av JWT token'er for Express. Denne er open-source og er tilgjengelig gjennom GitHub[77].

For å forsike om at autentisering skjer som det andre steget i pipeline'en for alle API-ruter bortsett fra de som ikke krever innlogget bruker, la vi til denne kodesnutten i *routes.js*.

```
1 // Authenticate requests using JWT (JSON Access Token)
2 router.use(auth.token);
```

Listing 4: Håndhever autentisering i *routes.js*

Metoden use()[74] gjør at alle API-ruter som defineres under denne, må gå gjennom metoden *auth.token* som er en *middleware*- ansvarlig for å validere en JWT-token (se seksjon 6.2.3).

```
1  exports.token = function (req, res, next) {
2      // Try to retrieve token from body, query strings, or HTTP headers
3      const token = req.body.token || req.query.token ||
4      req.headers['x-access-token'];
5      if (!token) {
6          return next(errors.unauthorized('No authentication token provided'));
7      }
8
9      authToken.verifyToken(token, function(err, decoded) {
10         if (err) {
11             return next(err);
12         }
13         res.locals.token = decoded;
14         next();
15     });
16 }
```

Listing 5: Funksjon som autentiserer en JWT-token

Metoden henter token fra HTTP headeren i forespørselen, og kaller igjen en egen funksjon kalt *verifyToken* som bruker *express-jwt-token* til å verifisere at token er gyldig. Dersom denne passerer uten feil, vil metoden next()[74] kalles som sender forespørselen videre i pipeline'et. I motsatt fall vil en feil umiddelbart returneres til Express sin error-handler og forespørselen får ikke passere til neste steg.

```
...
// Verify the token signature
jwt.verify(token, config.token_secret, function (err, decoded) {
    if (err) {
        console.log(err);
        return cb(errors.unauthorized('Invalid authentication token'));
    }
    cb(null, decoded);
});
...
```

Listing 6: Utdrag fra funksjonen *verifyToken*

Validering

Ettersom vi bruker en MySQL database må vi særlig beskytte oss mot SQL-injection, som kan tillate angripere å manipulere databasen[28]. Validering bruker vi også for å begrense minimum og maksimum lengde på brukernavn, at e-post adresser er i korrekt format, sikre at passord har minst 6 tegn og lignende.

Med det første vurderte vi å implementere funksjoner som validerer og sjekker format på dataene selv. Dette viste seg imidlertid raskt å være upraktisk og tidkrevende. Vi begynte derfor å se etter ferdige løsninger som kunne tas i bruk. Vi fant fram til en open-source modul for Node.js kalt *express-validator*, som er tilgjengelig gjennom GitHub[78]. Denne modulen tilbyr validering for Express, ved at den har metoder som kan brukes for å validere tilsendte parametre i en forespørsel. Disse metodene er basert på en annen modul kalt *validator.js*, som også er tilgjengelig på GitHub[79].

For å holde orden på forskjellige valideringsprosedyrer for hver API-rute, opprettet vi filen *validation.js*. Denne inneholder funksjoner som korresponderer til en *route* vi bruker i APIet. Et typisk eksempel på en slik funksjon vises her:

```
...
    check('name').isAlpha('nb-NO').isLength({min: 2, max: 32}),
    check('email').isEmail().isLength({min: 3, max: 64}),
    check('username').isAlphanumeric('nb-NO').isLength({min: 3, max: 32}),
    check('password').isLength({min: 6, max: 72}),
    check('country').isAlpha().isUppercase().isLength({min: 2, max: 2}),
    check('gender').optional().isAlpha().isUppercase(),
...
```

Listing 7: Utdrag fra filen *validator.js*

Resultatet av valideringen hentes ved hjelp av *validationResult*[78], som forteller hvilke tilsendte parametre som ikke passerte validering. Dersom denne ikke er tom, vil en feil umiddelbart returneres til Express sin error handler. Brukeren får i retur en melding om hvilke felter det er feil med, ved hjelp av *validationResult* sin mapped() metode (se API [78].

Den definerte *registrer bruker*-ruten i *routes.js* vil da se slik ut:

```
1 router.post('/user', validation.createUser, checkValidation, user.createUser);
```

Listing 8: Utdrag fra filen *routes.js*

Denne kodelinjen vil håndheve at ruten først må gå gjennom valideringsprosedyren for *createUser*, så funksjonen *checkValidation*, som er en felles funksjon som sjekker at valideringen gikk gjennom. Til slutt kommer funksjonen for logikken som faktisk registrerer brukeren.

Middleware

Middleware er callback-funksjoner i Express-rammeverket som kan inneholde all mulig kode. Navnet kommer av at koden eksekveres mellom en forespørsel til serveren, og før en respons sendes tilbake. En kan derfor si at koden lever *mellom* en forespørsel. En middleware-funksjon er altså en funksjon som eksekveres først når en klient aksesserer en bestemt *route* på serveren. Bruken av middleware er det som definerer Express-rammeverket, og hvert steg i pipeline'et beskrevet over er egentlig en egen middleware-funksjon. Det kan være flere middleware-funksjoner som brukes for en *route*, noe som muliggjør en slik pipeline vi har beskrevet. Disse kaller da metoden next() som vil sende kontrollen til neste middleware i kjeden[80].

Noe annet som kjennetegner en middleware-funksjon er at den får tilsendt to objekter som parametre: et *request* og et *response* objekt (forkortet med req, res)[80]. Med *request*-objektet får man tilgang til egenskapene for en forespørsel, inkludert URL, cookies, og verdiene som er tilsendt fra en klient i en GET eller POST forespørsel som nøkkel-verdi par[74]. *Respons*-objektet brukes for å kontrollere hva som blir sendt tilbake til klient i gjeldende forespørsel. Dette objektet inneholder metoder som send()[74] som sender en streng tilbake til klienten, og json()[74] som sender et JSON-objekt som respons.

Det siste steget en forespørsel går gjennom i backendløsningen etter autentisering og validering er passert, er da selve logikken i form av en middleware. Disse vil alltid være siste steg i pipeline'et og resulterer i at en respons sendes til klienten. Som regel består logikken av databasespørringer, som å hente noe eller oppdatere data i databasen. Et typisk eksempel er use-caset *se produktinformasjon* beskrevet i seksjon 3.1.2, hvor mobilapplikasjonen må gjøre en forespørsel til backend for å hente ut produktinformasjon fra databasen. Implementasjonen av denne vises under.

```
exports.getProductByID = function(req, res, next) {
    queryDb('SELECT pr.model, pr.name, pr.imgurl, pr.ic_pid,
     ↪  ipr.ic_filepath, iprn.ic_catname FROM product AS pr ' +
            'LEFT JOIN icecat_product AS ipr ON pr.ic_pid = ipr.ic_pid '
             ↪  +
            'LEFT JOIN icecat_category_name AS iprn ON ipr.ic_catid =
             ↪  iprn.ic_catid ' +
            'WHERE pr.pid = ? AND iprn.ic_langid = 1;' +
            'UPDATE product SET hits = hits + 1 WHERE pid = ?',
             ↪  [req.params.productid, req.params.productid],
             ↪  function(err, rows) {
        if (err) {
            return next(errors.internal());
        }
        res.json(rows[0]);
    });
}
```

Listing 9: Funksjonen *getProductByID*

Databasespørringen er litt kompleks, fordi vi har flere forskjellige produkttabeller (forklart i seksjon 5.2.3). Oppgaven til denne funksjonen er å hente ut produktinformasjon ut fra produkt ID. Produkt ID'en blir sendt med i en forespørsel fra brukeren, og verdien hentes ut gjennom *request*-objektet sitt *params* objekt. Verdien brukes så i databasespørringen, hvor vi har funksjonen *queryDb* som utfører spørringen og returnerer resultatet. Dersom *queryDb* returnerer uten feil, vil metoden json() kalles på *response*-objektet og resultatet av spørringen returneres i respons til brukeren.

Det er verdt å merke seg at denne funksjonen egentlig inneholder to databasespørringer, hvor den andre spørringen inkrementerer attributtet *hits* for produktet med en. Dette gjøres slik at produkter rangeres etter popularitet i produktsøk. Merk også at i denne middleware-funksjonen er det et *next* objekt i tillegg til req og res, selv om denne funksjonen er sist i pipeline'en. I dette tilfellet brukes next() for å kalle express sin error-handler[81].

6.2.4 Produktsøk

Muligheten til å søke etter produkter er en sentral del av løsningen. Det bør derfor være slik at når en bruker søker etter et produkt, er resultatene rangert etter relevans i forhold til søkeord. Når vi startet å implementere produktsøk bestemte vi oss først for å lage en enkel løsning som kun sammenligner produkter etter navn. Den mer avanserte utgaven som tar flere variabler enn dette i betraktning ble tatt senere i utviklingsprosessen, da vi fokuserte på å implementere et enkelt og fungerende produktsøk først. På denne måten kunne vi implementere produktsøk i Androidapplikasjonen raskt istedenfor å vente på at backendløsningen var ferdig.

Vi implementerte derfor først enkel søkefunksjonalitet som bruker en funksjonalitet fra MySQL kalt FTS (Full-Text Search)[82]. En annen vanlig måte å søke i en MySQL-database på er å bruke setningen *LIKE*, som sammenligner ord. Problemet med denne teknikken er at den sammenligner ord nøyaktig, noe som ikke er veldig praktisk i virke-

ligheten. Dette problemet løser FTS til en viss grad ved at den sammenligner ord som er mer likt søkeordet. En annen ulempe med å bruke *LIKE*, er at ytelsen er mye dårligere. Dette er fordi FTS bygger kontinuerlig opp en indeks, som den slår opp i når den leter etter lignende rader i tabellen[83]. Med potensielt millioner av produkter i databasen ville det sannsynligvis blitt et ytelsesproblem å ikke ha en form for indeks.

Den enkle løsningen med FTS fungerte, men det viste seg å være en del problemer når det kom til relevansen på resultatene. Det største problemet var at når noen søkte på eksempelvis søkeordet *iPhone*, så kom det opp hundrevis av tilbehør som etui og kabler for telefonen. Selve telefonen var derimot langt ned på søkeresultatet. Vi valgte derfor å se nærmere på informasjonen tilgjengelig i Icecat sin produktkatalog. I produktkatalogen til Icecat bruker de et hierarki av kategorier. Dette fungerer på den måten at overordende produkter, som mobiltelefoner, bærebar PC og lignende er i øvre kategorier, med tilbør til produktene som underkategorier. Derfor modifiserte vi script'et vi bruker til å importere produkter fra Icecat (se seksjon 6.2.1) til å telle dybden på hver enkelt kategori, hvor øverste kategori starter på null, og økes med en for hver underkategori. Vi la så til attributtet *depth* i tabellen *Icecat_Category* (vist i seksjon 5.2.3). Dette løser problemet ved å gi produkter en lavere dybde enn dets tilbehør, og vil derfor rangeres høyere i søkeresultatene.

Til slutt la vi til attributtet *hits* for et produkt (tabellen *Product*). Denne økes med en hver gang en bruker ser på et produkt (vist i 6.2.3). På denne måten kan vi også sortere produkter etter popularitet i produktsøk.

Produktsøk endte da opp med å fungere slik:

1. Sammenlign ord med FTS (Full-Text-Search)[82].
2. Sorter resultater etter *hits* i synkende rekkefølge.
3. Sorter resultater etter dybde for produktets kategori i stigende rekkefølge.

6.2.5 Paginering

Ettersom resultatsettet for et produktsøk kan inneholde flere tusen rader, og det kan være hundrevis av anmeldelser for et produkt, var det nødvendig å ha et system for å dele opp resultatene i egne sider. Dette er nødvendig for å få en responsiv brukeropplevelse. Man ønsker ikke å laste ned for mye data om gangen da det fører til lange ventetider og mye minnebruk.

For dette systemet ville vi heller ikke gjenbruke kode, da vi visste at vi trengte funksjonaliteten flere steder. Måten vi løste det på var ved å lage et system som kan gjenbrukes gjennom funksjonen *paginate* vist under. Denne mottar en SQL-spørring, verdier for spørringen, og et sidetall som starter på en. Den legger så til to nye verdier for spørringen: *min*, som forteller MySQL nummeret på den første raden som skal hentes, og *max* som forteller den siste raden. Vi bruker 20 resultater per side, så med sidetall nummer en blir *min* da null og *max* blir 20. Funksjonen forandrer på den opprinnelige SQL spørringen ved å legge til spørringen *CALC_FOUND_ROWS* som returnerer det totale antall radene i *hele* resultatsettet[84]. Resultatene som returneres til klienten blir deretter begrenset med setningen *LIMIT*. Det som returneres til klienten er et JSON objekt med nøkkelen *pages* som forteller hvor mange sider det er totalt i resultatsettet (som er totalt antall resultater / resultater per side), og en JSON array kalt *results* som inneholder resultatsettet for siden med maks 20 resultater.

Når klienten utfører et produktsøk, eller henter produktanmeldelser fra server, må en ekstra verdi legges til i forespørselen kalt *page*. En forespørsel for å hente anmeldelser fra produkt ID en og sidetall to vil da se slik ut:

```
GET /review/1?page=2
```

```
var paginate = function(sql, values, page, cb) {
    // Limit min,max
    var min = config.results_per_page * (page - 1);
    var max = min + config.results_per_page;
    values.push(min);
    values.push(max);

    // Change SQL query
    sql = sql.replace("SELECT", "SELECT SQL_CALC_FOUND_ROWS")
        .concat(" LIMIT ?,?; SELECT FOUND_ROWS() AS count");

    queryDb(sql, values, function(err, rows) {
        if (err) {
            cb(err, null);
        }

        var totalPages = Math.ceil(rows[1][0]['count'] /
        config.results_per_page);
        var res = {'pages':totalPages, 'results':rows[0]};
        cb(null, res);
    });
}
```

Listing 10: Funksjonen *paginate*

På klienten vil paginering være nesten usynlig for brukeren. I androidapplikasjonen er paginering håndtert ved at klienten automatisk henter inn en ny side når brukeren blar seg nedover og når slutten på "siden". På denne måten er alle resultatene "på samme side" fra brukerens perspektiv. Dette er et designmønster kjent som *infinite scroll*[85].

6.3 Android

6.3.1 HomeActivity

Måten vi har brukt *Activities* og *Fragments* på varierer. Vi har brukt *fragments* der det er hensiktsmessig og *activities* hvor dette passer bedre. I hoved *activity*'en kalt *HomeActivity*, har vi en egen container som holder en fragment, og denne byttes ut når brukeren trykker på en tab nederst på skjermen. På denne måten beholder vi deler av grensesnittet, men bytter ut kun det som er relevant. Her blir også en *backstack* for fragments brukt. Denne kan lagre fragmentenes tilstand på en stack slik at de kan gjenopprettes senere, eksempelvis når brukeren trykker på tilbakeknappen.

```
1     private void showContent(Fragment fragment, String tag) {
2         getSupportFragmentManager().beginTransaction().
3                 setCustomAnimations
4                 (android.R.anim.slide_in_left,
5                 android.R.anim.slide_out
6                 _right,
7                         android.R.anim.slide_in_left,
8                         android.R.anim.slide_out_
9                         right).
10                replace(R.id.mainContent, fragment, tag).
11                addToBackStack(tag).commit();
12    }
```

Listing 11: Funksjonen *showContent*

6.3.2 ApiRequest

All form for kommunikasjon med backend skjer ved å bruke klassen *ApiRequest*. *ApiRequest* er i seg selv en utvidelse av *AsyncTask*[86]. Det er en kodesnutt som kjører for seg selv uavhengig av hva som vises for brukeren. For å gjøre et kall til backend, er det mulig å bruke *ApiRequest* klassen direkte, men det har også blitt laget en forenklet klasse kalt *SimpleApiRequest*, som er en "wrapper"-klasse rundt *ApiRequest* som er enklere å bruke.

Kobling til backend

For å koble applikasjonen opp mot backend, brukes det flere forskjellige forespørselsmetoder som *GET*, *POST* og *DELETE*. Navnet på metoden sendes til *ApiRequest* i første parameter som streng. Backend API'et tilbyr flere ruter for å sende tilbake til brukeren ønsket informasjon. Denne settes som andre parameter. Dersom ruten bruker metoden *GET*, settes ønsket data etter ruten ved bruk av standard URL syntaks, som for eksempel skjer når man ønsker å få opp en ny side med anmeldelser fra et produkt. I tillegg er det nødvendig i de fleste kallene å legge på en autentiserings-token. Denne er også lagret og sendt rundt i applikasjonen som en streng.

I de fleste tilfellene er det nødvendig å sende med ekstra informasjon til backend, om det er for å spesifisere hva brukeren ønsker eller for eksempel å registrere en bruker. For å håndtere dette sender vi med en liste bestående av to strenger som nøkkel-verdi par. Dette fører til at vi kan sende med til backend et variabelt antall datasett, avhengig av hva vi ønsker å gjøre. Til slutt er det nødvendig å spesifisere et *RequestCallback* objekt, som er en callback som kalles når *ApiRequest* er ferdig. Dette objektet må implementere de to funksjonene *onSuccess* og *onError*, som kalles avhengig av om backend kunne håndtere forespørselen.

Feilhåndtering

ApiRequest klassen er ganske usikker, og håndterer dårlig problemer som skulle oppstå dersom noe galt skjer. Dessuten er den veldig unaturlig å bruke, krever at man lager et nytt objekt for parametre samt vet hva type den er, og til slutt krever en ordrik definisjon av funksjoner som skal kjøre etter at *ApiRequest* er ferdig.

Klassen *SimpleApiRequest* tar seg av de ekstra parametere som skal sendes med i *ApiRequest* ved å tilby en enkel metode kalt *setParam* istedenfor å sende med disse direkte.

I tillegg implementeres det en forenklet callback som gir respons eller feilmelding. Gjennom forening av disse to funksjonene, er det mulig å konvertere den ordrike definisjonen av callback til et enkelt *lambda*-uttrykk, noe som vi mener gjorde koden enklere å lese, samt holder koden som skal kjøres etter *ApiRequest* i samme *scope* som koden før og ellers i funksjonen. På den måten fjerner man unødvendige deklarasjoner av objekter med final flagg, bare for å ha tilgang til disse etter *ApiRequest*.

Ved å ta i bruk både *SimpleApiRequest* og *ApiRequest*, kan vi splitte oppgavene som skal utføres mellom disse to, og nå er *ApiRequest* ansvarlig for all slags nettverkskommunikasjon og *SimpleApiRequest* brukes til forenkling og data validering. Det er for eksempel i *SimpleApiRequest* som sjekker om brukeren er tilkoblet internett, og hvis ikke, avbrytes forespørselen *ApiRequest* som ellers ville gjort at applikasjonen krasjer.

På følgende kodesnutt kan man se den gamle koden som ble brukt til å lage en ny bruker, fulgt av den nye.

```
...
ArrayList<Pair<String, String>> params = new ArrayList<>();
params.add(Pair.create("username", username.getText().toString()));
// Fem flere params

final Activity thisActivity = this;
new ApiRequest("POST", "user", null, params, new RequestCallback<String>() {
    @Override
    public void onSuccess(String res, int status) {
        Intent intent = new Intent(thisActivity, LoginActivity.class);
        intent.putExtra("user_created", getString(R.string.user_created_text));
        startActivity(intent);
    }

    @Override
    public void onError(String err, int status) {
        Log.d("RESPONSE", err);
    }
}).execute();
...
```

Listing 12: Viser en forespørsel til API før refaktorering

```
SimpleApiRequest createUserRequest = new SimpleApiRequest(
    this, "POST", "user", null);

createUserRequest.setParam("username", username.getText().toString());
// Fem flere params

createUserRequest.execute((error, res, status) -> {
    if (error) {
        Messages.showError(
        getApplicationContext(),
        this.getString(R.string.create_user_failed), null);
        return;
    }

    Intent intent = new Intent(this, LoginActivity.class);
    intent.putExtra("user_created", getString(R.string.user_created_text));
    startActivity(intent);
});
```

Listing 13: Viser en forespørsel til API etter refaktorering

6.3.3 Filtrering av anmeldelser

Filtrering av anmeldelser skjer i hovedsak på klienten. Dersom brukeren ønsker å kun se positive anmeldelser, eller delanmeldelser 5.4.2 for en *anmeldelsestag* 5.4.2 kan brukeren enkelt finne fram til mer spesifikk informasjon ved å filtrere anmeldelser ut ifra eget ønske.

For å enkelt sortere anmeldelser som ønsket, er anmeldelsene lagret i to forskjellige matriser. Matrisen *allItems* inneholder alle anmeldelser som har blitt lastet inn fra backend i den rekkefølgen de hentes. Matrisen *items* starter som en svak kopi av *allItems*. Det betyr at den peker til de samme objektene som *allItems*. Denne matrisen er også delt med adapteren til listen som håndterer visning av anmeldelsene, og alle forandringer på *items* speiles i adapteret. Ved å utnytte grunnleggende funksjonalitet i objekter, oppbevarer vi bare en kopi av hver anmeldelse, og velger ut hva brukeren ser ved å justere på pekere, ikke data.

Bruker kan filtrere anmeldelser på to forskjellige måter. Det er mulig å se hele anmeldelser eller delanmeldelser basert på anmeldelsestagg'er, som begge kan filtreres ut ifra vurdering. Den korrekte funksjonen for filtrering av anmeldelsesobjektet kalles ut ifra brukerens valg. Når anmeldelsen ikke har den utvalgte tagg'en, blir hele anmeldelsen fjernet fra *items* matrisen.

Inne i anmeldelsesobjektene brukes samme håndtering av sortering som i fragment'et 5.5.1 som vises til brukeren. Alle delanmeldelser holdes lagret i *subReviewsAll*, og hver gang man sorterer anmeldelser fjernes den gamle sorteringen og erstattes med en kopi av *subReviewsAll*. Unntaket er generelt og konklusjon, som holdes utenfor dette systemet og gjemmes når brukeren sorterer etter hvilket som helst tagg.

Til slutt i funksjonen gir man beskjed om at datasettet, i dette tilfellet *items* er forandret og utseende må oppdateres. Anmeldelsen er filtrert basert på valg fra bruker, og resultatet blir synlig umiddelbart, uavhengig av internett tilkobling.

```
private void filterReviews() {
    ReviewListFragment.this.items.clear();
    ReviewListFragment.this.items.addAll(ReviewListFragment.this.allItems);

    String filterTag = this.reviewTagsFilter.get(this.selectedTag);

    for (int i = 0; i < this.items.size(); i++) {
        ReviewModel review = this.items.get(i);
        //4-way if tree, based on whether searching
        for tag, rating, both, or none
        //TRUE = show all
        if (filterTag.equals(reviewTagsFilter.get(0))) {
            if (this.selectedRating == REVIEW_RATING_SHOW_ALL) {
                review.showAll();
            } else {
                review.showOnly(this.selectedRating);
            }
        } else {
            if (this.selectedRating == REVIEW_RATING_SHOW_ALL) {
                review.showOnly(filterTag);
            } else {
                review.showOnly(filterTag, this.selectedRating);
            }
        }
        if (review.nrOfSubReviews() < 1) {
            this.items.remove(i);
            i--;
        }
    }
    ReviewListFragment.this.rla.notifyDataSetChanged();
}
```

Listing 14: Funksjonen *filterReviews*

6.3.4 Visning av bilder

Måten den gamle koden fungerte på, var at vi først forsøkte å hente bilder fra *cache* gjennom en egenimplementert klasse. En *cache* er et sted for mellomlagring av data på minne som er mye raskere tilgjengelig enn den opprinnelige lagringsplassen. I vårt tilfelle er cache det lokale minnet på enheten, mens den opprinnelige lagringsplassen ligger på nett. Dersom bildet fantes i cache gjenbrukte vi den, hvis ikke, forsøkte vi å koble til bildets URL-adresse for å lese inn data direkte. Resultatet var at bildene veldig ofte endte opp på feil plass og ble veldig raskt erstattet med andre bilder fra andre produkter, eller bare ikke lastet inn. Problemene oppsto trolig på grunn av måten søkefragment'et fungerete i bakgrunnen, men det har tvang oss uansett til å lete etter en annen måte å gjøre det på.

Etter litt søking etter et eksternt bibliotek, endte vi opp med å erstatte metoden vår med *Glide*[87]. Dette er et bibliotek egnet nettopp til det vi ønsket, som hadde ingen av de problemene vi opplevde med vår egen metode. Forskjellen mellom vår egen løsning og *Glide* er best illustrert i dette eksemplet:

```
// Get the image from the GetBitmap task, and place it in the imageview
if (!TextUtils.isEmpty(imgUrl)) {
    new GetBitmap(imgUrl, new RequestCallback<Bitmap>() {
        @Override
        public void onSuccess(Bitmap res, int status) {
            searchImg.setImageBitmap(res);
        }

        @Override
        public void onError(String err, int status) {
            // Not used
        }
    }).execute();
}
```

Listing 15: Hvordan bilder ble hentet inn før refaktorering

```
if (!TextUtils.isEmpty(imgUrl)) {
    Glide.with(convertView).load(imgUrl).into(searchImg);
}
```

Listing 16: Hvordan bilder ble hentet inn etter refaktorering

Etter denne forandringen justerte vi ytterligere på løsningen, som ved å legge til et standard bilde for når bilde for produkt ikke er tilgjengelig. Det vi har lært av å implementere denne delen er at man bør se aktivt etter andre løsninger og vurdere om de egner seg før man prøver å løse problemet selv. I vårt tilfelle lot det oss erstatte en kodesnutt på 11 linjer som var vanskelig å lese med kjente problemer, med en linje som klart og tydelig løser problemet uten trøbbel, er det verdt det.

6.3.5 Dekoding av strekkoder

For å håndtere lesing av strekkoder måtte vi finne et passende eksternt bibliotek. Ettersom vi utvikler applikasjonen for Android har vi valgt å benytte Googles eget Mobile Vision bibliotek til å lese strekkoder[88]. Vi var her nødt til å benytte oss av et eksternt bibliotek, ettersom egenutvikling av et slikt system ville tatt altfor lang tid. Ved å benytte dette biblioteket kan vi støtte mange strekkodetyper uten at vi trenger å bruke tid til å utvikle vår egen løsning som trolig ikke ville være like bra, noe vi har erfart før som beskrevet i 6.3.4. Dessuten er fokuset i oppgaven ikke å utvikle en strekkodeleser, men et sosialt nettverk for produktanmeldelser.

Når det gjelder vår implementasjon, har vi valgt å benytte store deler av løsningen i eksemplet for bruk av Mobile Vision[89].

6.3.6 Legge til nye produkter

En viktig del av oppgaven er at en bruker skal kunne legge til et nytt produkt i databasen. Grunnlaget for dette er at Icecat ikke har tilgang til alle eksisterende produkter på markedet. Dette vil føre til situasjoner hvor bruker prøver å finne anmeldelser for et produkt, hvor vi ikke har produktet i databasen. Produkter kan også ha mer enn kun en strekkode

og det er ikke alltid Icecat har tilgang til alle disse. Det vil derfor oppstå situasjoner hvor bruker skanner et produkt som ligger i databasen, hvor det ikke vil komme noe resultat. Med andre ord må det i tillegg til en løsning for at bruker skal kunne legge til et nytt produkt, også være en løsning hvor strekkoder kan legges til for eksisterende produkter.

Vi har valgt å avgrense på noen forskjellige områder. Det tas ikke til rette for at løsningen skal gjøre noe med at brukere vil kunne legge til feil produkter. Dette på grunn av mangel på en administrativ løsning i prototypen. Løsningen er ment som et grunnlag for hvordan produkter kan legges inn og er ikke ment som en ferdig løsning. Produkter kan også kun legges til etter at bruker har skannet en strekkode. Dette er for å hele tiden ha et unikt element assosiert med produktet, noe som vil hjelpe til å hindre duplisering av produkter.

Prosessen starter med at brukeren skanner en strekkode på et produkt. Hvis strekkoden ikke finnes i vår database, vil det dukke opp en melding som lar brukeren legge til produktet i databasen. Strekkoden som ble skannet lagres i en variabel og sendes med til *AddProductActivity* hvor produkt eller strekkode kan legges til. Bruker kan her skrive inn navnet på produktet og vil fortløpende få forslag til produkter med liknende navn. Her blir det gjort et søk på samme måte som ved home. Eneste forskjell er at anmeldelser er fjernet fra hvert produkt i resultatslisten ettersom de ikke er nødvendige. Hvis bruker trykker på et av produktene vil et fragment (se seksjon 5.5.1) av produktsiden fylle skjermen. Dette gjøres for å gi brukeren muligheten til å finne produktet som ble skannet og gir så muligheten til å legge til strekkoden til det eksisterende produktet. Om brukeren ikke finner produktet kan brukeren velge å legge til et nytt produkt. Dataene som må legges til er navn, modellnavn og bilde av produktet ved bruk av telefonens kamera.

...

```
1  if (ContextCompat.checkSelfPermission(
2              Objects.requireNonNull(getActivity()), Manifest.
3              permission.CAMERA)
4          != PackageManager.PERMISSION_GRANTED) {
5      // Request permissions
6      ((AddProductActivity) getActivity()).requestCameraPermission();
7  } else {
8      // Permissions already granted
9      ((AddProductActivity) getActivity()).dispatchTakePictureIntent();
10 }
11 ...
```

Listing 17: Hvordan vi bruker kamera i Android API'et

Sjekk for tilgang til kamera må håndteres hver gang den brukes, ettersom det fra Android 6.0 er mulig for brukere å nekte tilgang til enkelte rettigheter mens applikasjonen er aktiv[90]. Dersom tilgang allerede er blitt gitt, kan vi sende en forespørsel til mobilen om å ta bilde, hvor brukeren da kan bruke kamera med telefonens egen kameraapplikasjon.

Problemet med løsningen

Som sagt er det problemer med denne måten å håndtere nye produkter. Det største problemet er kvaliteten mellom informasjon ifra Icecat i forhold til bruker. Icecat gjør at vi

får tilgang til mye informasjon om produktet. Her er det først og fremst bilder, forklaring, spesifikasjoner og kategorier som er informasjon vi bruker. Av disse er kategorier sentral ved at den hjelper oss å sortere resultater i søk og vil gjøre det mulig i framtiden å implementere en type for filtrering i produktsøk. Dette kan i teorien løses ved å la brukeren selv legge inn manglende informasjon, hvor det største problemet blir å sikkre kvaliteten på informasjonen.

6.4 Webapplikasjon

6.4.1 Navigasjon

Vi har valgt å implementere webapplikasjonen som en såkalt *Single Page Application*. Dette vil si at nettleseren kun laster inn en enkelt side først, og andre sider blir dynamisk lastet inn i en *container*. AJAX forespørsler brukes i JavaScript til å dynamisk hente data fra server, som fører til en raskere og mer responsiv side fordi det ikke trengs å lastes inn like mye data når man henter inn en ny side. Dette fordi man kun henter inn markup og kode som er spesifikk for den ene siden.

Måten dette fungerer på er svært enkelt. Vi har en tom *div* i index.html (hovedsiden) kalt *content*. Når brukeren ber om en ny side, kalles funksjonen *loadPage* som bruker JQuery sin *.load()* funksjon, som henter dataen fra server og setter det inn i *div*'en.

```
1 ...
2     $content.load(url, function() {
3         $content.fadeIn();
4         console.log('Loaded', url);
5 ...
```

Listing 18: Utdrag fra funksjonen loadPage

6.4.2 Anmeldelser

En anmeldelse består av et hierarki med HTML-elementer. Dette hierarkiet starter med en *container* som ligger rundt alle elementene i anmeldelsen. Elementene består av *container*'e for "generelt", delanmeldelser og konklusjon. Vi vil i denne seksjonen snakke mer om funksjonalitetene rundt de forskjellige komplekse elementene i anmeldelsen.

Delanmeldelse

Hver delanmeldelse er et større HTML-element bestående av mindre elementer. Hver gang brukeren vil legge til en ny delanmeldelse lages det en ny *container*, hvor alle underliggende elementer *føyes på*. Delanmeldelsen føyes til en egen *container* for delanmeldelser, som lar brukeren stokke om delanmeldelsene. Implementasjonen av hvordan disse elementene bygges opp kan sees i kodesnutten under.

```
...
    function addNewSubreview() {
    // Function html elements
    var subreviewContainer = document.createElement('li');
    var removeSubReview = document.createElement('button');
    var subreview = document.createElement('div');
    var titleDiv = document.createElement('div');
...


...
    // appending elements to subreview
    $(subreviewControlPanel).append(titleDiv);
    $(titleDiv).append(titleTag);
    $(titleDiv).append(input);
    $(subreviewControlPanel).append(ratingDiv);
...

...
    // Giving class to ratingDiv
    ratingDiv.className = "rating-div";
    ratingButton.className = "rating-button"
    ratingButton.setAttribute("tabindex", 0);
    // This is for enabling "focus" on the div element
...
```

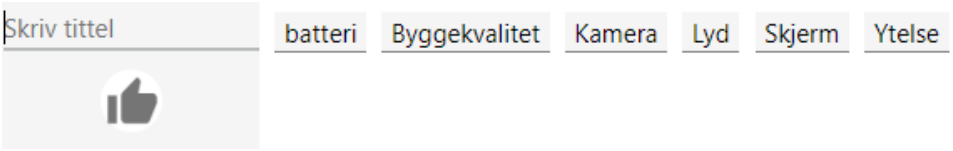
Listing 19: Utdrag fra funksjonen addNewSubreview

Her ser man hvordan funskjonen *addNewSubReview* oppretter de nye HTML-elementene ved bruk av *document.createElement()*. Elementene legges til variabler for å kunne brukes senere i funksjonen. Delanmeldelsen bygges så opp ved at elementene blir lagt til i et hierarki i delanmeldelsen, ved å føye dem til deres *parent*. Data blir så lagt til lenger ned i funksjonen. Denne måten å bygge opp delanmeldelser på, lar oss ha alt på ett sted. Det ble derimot klart jo mer delanmeldelsen vokste i funksjonalitet at dette var feil måte å gjøre det på.

Problemet er at for hvert nytt element som legges til i en delanmeldelse så øker antallet elementer i funksjonen. Dette var ikke et problem i starten av utviklingen da hver delanmeldelse ikke var spesielt kompleks. Etter en del videreutvikling endte det opp med mange elementer for hver delanmeldelse, noe som gjorde det vanskelig å finne fram til hvert element i funksjonen.

En mer optimal løsning ville vært å lage egne funksjoner som tar seg av mindre komplekse elementer i delanmeldelsen. Disse funksjonene kan så kalles i *addNewSubReview*, hvor elementene kan legges sammen til en delanmeldelse. Dette ble ikke gjort på grunn av tidsmangel til refaktorering av kode.

Anmeldelsestagger



Figur 18: Visning av anmeldelsestagger

Hvor vurdering og tekst er mer selvforklarende er tagger et generelt nytt konsept i forhold til en vanlig anmeldelse. Tagger definerer delanmeldelsen og det var derfor viktig å implementere dem på en slik måte at brukeren lett forstår hva de gjør. Vi lar derfor brukere skrive taggen for en delanmeldelse som om det er en type tittel eller overskrift for delanmeldelsen. Ut ifra hva bruker skriver vises det eksisterende tagger mest lik tittelen i et vindu hvor den skriftlige delen ligger (se figur over). I kodesnutten under vises funksjonen som filtrerer taggene ut ifra hva bruker skriver.

```
1  function updateShownTags(text, newTagBtn, tagsDiv) {
2      var noTagsEqual = true;
3
4      // For each registered tag
5      $(tagsDiv).children('.tag').each(function(t, obj) {
6          // Nothing in title, show all tags to user
7          if(text.length == 0) {
8              visible(obj);
9              noTagsEqual = false;
10             $(tagsDiv).show();
11         } else {
12             // If the text matches part of processed tag, show the tag
13             if($(obj).html().toLowerCase().indexOf(text.toLowerCase()) >= 0) {
14                 visible(obj);
15                 noTagsEqual = false;
16                 $(tagsDiv).show();
17             } else {
18                 $(tagsDiv).hide();
19                 notVisible(obj);
20             }
21         }
22     });
```

Listing 20: Funksjonen *updateShownTags*

Funksjonen *updateShownTags* kjøres for hver gang brukeren legger til en ny bokstav i tittelen for en delanmeldelse. Det gjøres en enkel sammenlikning på brukerens tittel og eksisterende tagger, hvor det kun sjekkes om tittelen er en del av eksisterende tagger, f.eks som hvis ”bat” er en del av ”batteri” vil batteri taggen vises. Alle tagger ligger som egne elementer inne i en *conteiner* kalt *tagsDiv*. Sjekken for om tittel er en del av noen tagger skjer ved at alle elementene i kontaineren gås igjennom en etter en. For hver sjekk blir det valgt om taggen vises eller ikke vises.

Hvis tittel er tom vil alle tagger vises. Dette er for å la bruker få oversikt over alle eksisterende tagger, og for å hjelpe nye brukere til å forstå konseptet med tagger. Brukeren trenger kun å trykke på ønsket tagg, hvor taggen vil legges over plassen til tittel. Trykker brukeren på tittel nå, vil taggen fjernes og tidligere tittel vil bli vist.

7 Testing og kvalitetssikring

7.1 Kodekvalitet

7.1.1 Sonarqube

Statisk kodeanalyse er et viktig verktøy for utviklere. Det vil ikke automatisk gjøre koden bedre, men kan ofte bli satt opp slik at koden følger en bestemt stil, og oppdage noen av de mer enkle sårbarhetene i koden. Bortsett fra noe tid brukt i oppsettet, er det ganske raskt å bruke underveis i utviklingen. Det finnes utrolig mange forskjellige verktøy for statisk kodeanalyse[91]. Ettersom det ville tatt flere dager å gå gjennom alle alternativer, og at vi hadde mye erfaring ved bruk av *SonarQube*[92] i fra andre emner, valgte vi å bruke dette verktøyet for kodeanalyse.

For å gjennomføre kodeanalyse i Sonar, må man sette opp en profil med tilhørende regler som koden skal følge. Sonar støtter både Java og JavaScript, som er de språkene vi bruker i prosjektet. Ut av boksen, har Sonar en enkel konfigurasjon som kan utvides til å tilpasses individuelle utviklingsmiljøer. Fra vår erfaring fra tidligere emner har vi sett at det finnes utvidelser som kan tilby mye mer målrettet analyse. For å få bedre hjelp med analyse av Java koden, har vi lastet inn i tillegg *Spotbugs*[93], *FB-Contrib*[94] utvidelsen til *Spotbugs*, og Android Lint profilen for *SonarQube*[95]. Alle disse utvidelsene legger til ekstra profiler som gjør at Sonar kan finne flere feil i koden. Vi slo deretter sammen alle de tilgjengelige profilene, og gikk gjennom reglene som var slått av i de forskjellige profilene for å skru på de som vi følte var en god ide å ha med. I tilfelle det viste seg til å være feil, var det bare å skru av regelen senere.

I tillegg til Sonar har vi underveis i prosjektet brukt det innebygde verktøyet *Android Lint*, som følger med i Android Studio[96]. *Android Lint* har i motsetning til SonarQube mer Android-spesifikke regler. Selv om Sonar klarer å samle en del av feilene som Android Lint oppdager, følger ikke alle med, da utvidelsen ikke har blitt oppdatert på to år[95].

Videre var det nødvendig å sette opp *SonarQube* for våre forskjellige kodebaser. Webløsningen og backend tok i bruk et verktøy tilbudt av SonarQube[97], som kan gjennomføre analyse av prosjekter som ikke har et integrert system for slik analyse, mens vi i Androidløsningen tok i bruk utvidelsen for Gradle[98]. For å få på plass dette systemet på alle de forskjellige enhetene vi utvikler på, har vi brukt en primitiv løsning ved å kopiere installasjonen av SonarQube til de enhetene etter oppsett, ettersom det ikke var forventet at mye ville forandre seg under utviklingen i SonarQube til at vi skulle trenge store oppdateringer.

Selv om vi satt opp SonarQube for alle tre løsningene, har vi ikke hatt aktiv bruk av disse gjennom utviklingen. Et par analyser ble foretatt gjennom utviklingsprosessen, men oppussing av koden ble etter planen lagt til side for å prioritere utvikling. Valget ble tatt på grunnlag av at under utvikling vil mye av koden skrives om stadig, og bruk av mye tid på å holde koden pen og ryddig ville være unødvendig hvis om to uker skal løsningen skrives om. Utenom enkel beskjed om at alle bør holde en viss kontroll over

hva hver gruppemedlem gjør, slik at kode blir ikke for ulesbar, ble ikke analyseverktøy brukt mye under utvikling. Kvaliteten ble holdt til akseptabel nivå ved kort gjennomgang av hverandres kode når gruppemedlemmen var ferdig med sin oppgave i stedet for.

Uansett har vi satt opp Sonarqube, og gjennom bruk av Git er det mulig å vise frem utviklingen av Android løsningen gjennom hele utviklingsprosessen. Under følger en tabell med ukentlig gjennomgang av hvor stor Android løsningen var på den siste versjonen i uken, og antall problemer påvist av Sonar for den versjonen.

Dato	Linjer Java	Bugs	Vulnerabilities	Code Smells
29.01	106	0	5	24
05.02	441	6	6	72
12.02	1040	11	8	174
19.02	1097	11	9	179
26.02	1230	9	10	188
05.03	2176	22	16	345
12.03	2665	26	21	399
19.03	2908	30	24	437
26.03	3193	37	28	487
02.04	3559	49	26	521
09.04	4191	50	36	639
16.04	4282	16	16	368
23.04	4282	16	16	368
30.04	4269	16	16	366
07.05	4286	16	21	367
16.05	4341	16	21	352

Tabell 4: Sonarqube rapporter for Android

I denne tabellen har vi en oversikt over resultatene generert fra SonarQube på forskjellige tidspunkter under utviklingen. Tabellen viser kodefeil fra Sonar i forskjellige klassifikasjoner. *Bugs* er de mest alvorlige, og er konkrete problemer som kan forårsake feil i programmet. *Vulnerability* er sikkerhetsfeil, og *code smell* er ting som ikke påvirker selve kodelogikken, men kan være kode som ikke forholder seg til regler for kodestil og lignende[99].

7.2 Unit tester

Testing på Android kan deles opp i to forskjellige type tester [100]. Den første typen kalles *unit test*, som er vanlige Java tester som brukes for å verifisere at funksjonalitet fungerer som det skal. Forskjellige utviklingsmetoder plasserer skriving av slike tester forskjellig i utviklingsfasen. Målet med slike tester er å forsikre seg om at en funksjon gjør det samme, uansett hvordan den blir utviklet og refaktorert. Den andre typen kalles *instrumentation test*, som går ut på å kjøre applikasjonen på en Android enhet, enten på virtuell eller fysisk telefon. Den prøver å komme seg gjennom applikasjonen, med flere sjekk på veien som skal sjekke at alt er på plass. Begge typene fungerer forskjellig, og har hvert sitt miljø, og begge har sine unike oppgaver som de skal utføre.

Opprinnelig skulle vi utvikle tester for mesteparten av kodebasen. Dessverre har vi ikke fulgt opp dette målet, grunnet flere problemer som har oppstått gjennom prosessen når vi prøvde oss ut med det. For det første har vi hatt minimal erfaring med utvikling av

tester fra tidligere emner. De eneste testene vi har sett, var fra emnet ”Applikasjonsutvikling” hvor vi skulle skrive kode som skulle klare tidligere utviklede tester av foreleseren. Dermed hadde vi litt mangel på kunnskap for å utvikle gode tester. Uansett gjorde vi et forsøk på å utvikle tester for Androidløsningen, som var den eneste kodebasen som det ble gjort forsøk for.

Vår erfaring var at det tok flere timer å sette opp tester for å integrere alt sammen og sende det til SonarQube. Unit tester og instrumenterte tester har hvert sitt miljø og sender rapporter til hver sin plass, og hvis man ikke setter opp SonarQube i Gradle ordentlig, vil man ikke få alle testene i sluttrapporten. Det var lite dokumentasjon tilgjengelig rundt sammensetting av disse testtypene, men vi klarte til slutt å finne en løsning ved å se på en eksisterende metode som klarte å sette de to sammen[101]. Etter å ha generert noen tester ved bruk av løsningen fant vi ut at det fortstatt var noen problemer til stede, og det ble heller et grunnlag for løsningen vi endte opp med. Løsningen som gjør det mulig å slå sammen begge test typene for SonarQube er lagt til i vedlegg D.

Etter å ha klart å få testene til å bli vist fram slik de skulle i sonar, utviklet vi tre enkle tester. Test en skulle logge en bruker inn, test to skulle lage en bruker som allerede eksisterer i databasen, hvor målet var at testen ikke skulle kunne lage brukeren, og test tre skulle lage en bruker, for så å slette opprettet bruker. For å utvikle disse, benyttet vi en løsning i Android Studio kalt Espresso[102], som lar utviklere ta opp sine handlinger i en virtuell enhet, for å så gjenspille handlingene senere i utviklingen automatisk. Espresso hadde en del *bugs* når vi først tok opp testene og vi måtte gå inn å forandre på testene for å få de til å kjøre.

Forsøket på å utvikle tester tok flere timer å sette opp, og enda noen timer for å få på plass tre veldig enkle tester

De tre testene vi implementerte tok mye tid å sette opp i forhold til hvor nyttige de var. Hele prosessen ble gjort av samme gruppemedlem, noe som betydde at opplæring i hvordan testene måtte forandres etter opptak ville bli nødvendig om vi skulle fortsette med testing. Denne prosessen ble forsøkt i midten av februar, når gruppen var opptatt med både utvikling av funksjonalitet for Android- og webapplikasjonen. Vi valgte derfor ikke å utvikle tester med mindre det ville bli noe ekstra tid for det. Dette viste seg å være det rette valget, ettersom de tre testene raskt stoppet å fungere når applikasjonen ble videreutviklet. Hadde vi valgt å utvikle tester gjennom hele utviklingsperioden, ville vi trolig ende opp med å bruke stor del av tiden til å forandre på tester som stopper å fungere på grunn av forandringer i koden. I tillegg krevde sammensettingen av de to testtypene veldig spesifikke versjoner av verktøyene som de bruker, og etter oppdateringer er det nødvendig å forandre på oppsettet også.

7.3 Dokumentasjon av API

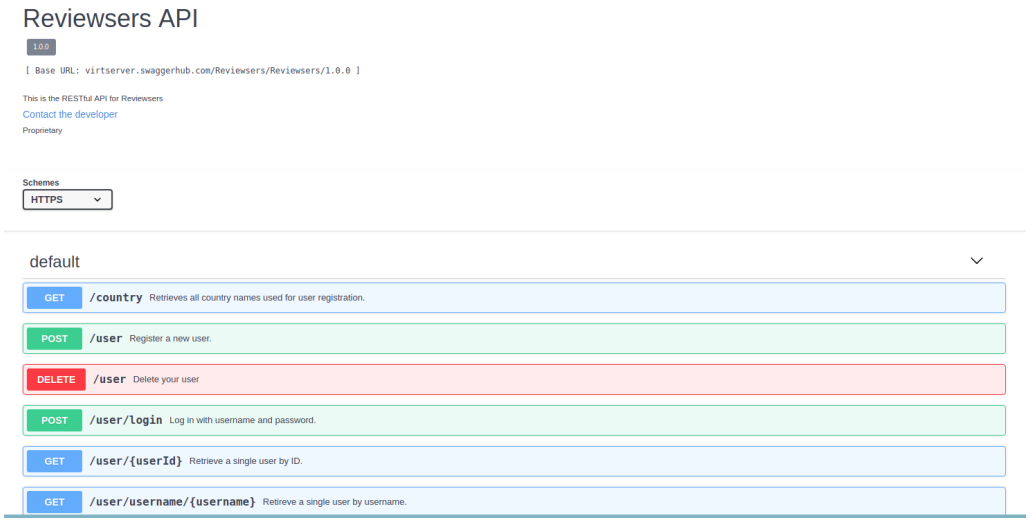
Når man skal utvikle et API, er det særdeles viktig å dokumentere all funksjonalitet og muligheter som API’et tilbyr klientene. Gjør man ikke dette vil det være nærmest umulig for andre utviklere å ta API’et i bruk. Dette er fordi det blir vanskelig å vite hvilke API-ruter (funksjoner i APIet) som er tilgjengelig, parametre for disse, og formatet på responsene. Vi var derfor på utkikk etter et verktøy som spesialiserte seg på dokumentasjon av API og fant fram til Swagger.io[103].

Det som gjør Swagger bedre til dokumentasjon av API i forhold til mer generelle

løsninger som markdown, Google Docs og lignende er at den er spesielt utviklet for dokumentasjon av RESTful API'er og kan automatisk generere dokumentasjonen. Swagger bruker en standard spesifikasjon kalt *OpenAPI* for å beskrive API'ets funksjonalitet, som skrives ned i et JSON- eller YAML-formatert dokument[104].

Ved å bruke OpenAPI-spesifikasjonen, får man en standard måte å dokumentere API'et på hvor alle rutene, dets tilhørende parametre og eksempler på respons beskrives med egne objekter i dokumentet som *paths*, *parameters*, og *responses*[104]. Når man har fylt ut disse objektene med data vil Swagger automatisk generere dokumentasjonen og det kan publiseres i form av HTML-dokumenter slik at andre kan lese den.

I vedlegget under vises et utdrag av hvordan dokumentasjonen av API'et ser ut. Hele dokumentasjonen kan også leses ved å følge lenken i bildets undertekst.



Figur 19: API-dokumentasjon med Swagger.io
https://app.swaggerhub.com/apis/Reviewsers/Reviewsers/1.0.0

7.4 Brukertesting

I løpet av planleggingsfasen diskuterte vi brukertesting av løsningen. Vi kom fram til at vi skulle ha to brukertester. Den første brukertesten skulle gjøres for den første implementasjonen av PRS hvor målet var å kunne forbedre løsningen ut ifra tilbakemeldinger. Den andre testen skulle være av androidapplikasjonen, etter de funksjonelle kravene var imøtekommet. En brukertest av applikasjonen ville være nyttig for å kunne få tilbakemeldinger på funksjonalitet vi hadde implementert underveis. Det ville gi oss mer informasjon om hva som kan endres på i en ferdig versjon av applikasjonen. Det ble derimot begrenset tid mot slutten av prosjektet. Vi så at vi hadde lagt av for lite tid til rapporten, og det var fortsatt mer funksjonalitet på androidapplikasjonen vi ønsket å få på plass. Vi valgte derfor å endre fokuset til å jobbe mer på Android, noe som gjorde at vi kunne starte på rapporten tidligere enn planlagt. Denne seksjonen vil derfor ta seg av testingen vi gjorde av PRS og hvordan vi forandret implementasjonen ut ifra tilbakemeldinger.

Brukere ble spurt om å anmelde sin egen telefon ved å dele den opp i flere delanmeldelser. Vi valgte telefon ikke bare på grunn av at det var et produkt alle hadde men også med tanke på at det vil være et produkt de har god kjennskap til. Vi la selv til *anmel-*

delsetagger for å hjelpe brukeren med å forstå konseptet, men også med mulighet for å legge til egne tagger. Etter testdeltakeren skrev anmeldelsen viste vi fram en produktside med en tilfeldig telefon hvor alle anmeldelser ble lagt inn. Her viste vi fram konseptet med hvordan anmeldelsene kan vises fram kun ut ifra en valgt "tagg", og filtrering av anmeldelsene.

Etter testing gikk testdeltakeren igjennom et spørreskjema hvor testeren fikk spørsmål om PRS og hvordan de syntes det fungerte. Andre spørsmål gikk mer ut på Reviewsers som konsept. Da mer spesifikt hva testerne syntes om ideen med å ha et eget brukersamfunn av anmeldere hvor alle anmeldelser er samlet på ett sted. Vi spurte også hvor testeren trodde de ville bruke løsningen mest: PC, mobil, eller andre plattformer.

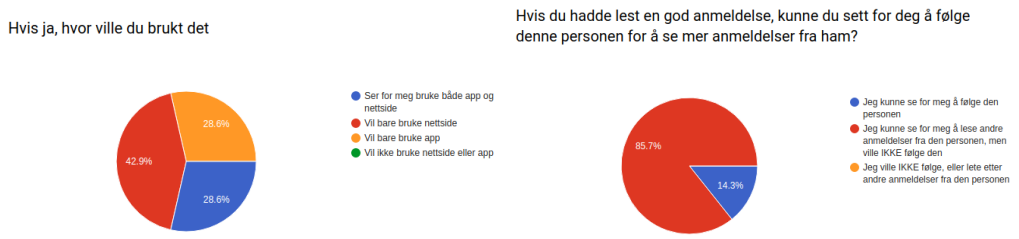
Alle spørsmål som ble brukt i brukertestingen ligger i vedlegg F.

7.4.1 Tilbakemelding

Tilbakemeldingene på både PRS og Reviewsers som ide var veldig positive. Spesifikt var det gode tilbakemeldinger på hvordan anmeldelser deles opp i delanmeldelser (se seksjon 5.4.2), og hvordan anmeldelsene vises fram. Når det kom til tilbakemeldinger for Reviewsers var tilbakemeldingene gode. Vi så en del uenighet om hvilken plattform brukeren ville brukt Reviewsers på. En stor andel (43%, som vist i figur 20) svarte at de kun ville brukt nettsiden, og ikke mobilapplikasjonen. Dette var delvis forventet fordi mange antakeligvis bestemmer seg for å kjøpe et produkt på forhånd hjemmefra før de går i butikken. Denne tilbakemeldingen hadde hatt betydning på hvor mye tid vi ville investert i videreutvikling på webløsningen kontra androidapplikasjonen.

Noe som ble kritisert av to testdeltakere var at hver "delanmeldelse" så for mye ut som egne "mini" anmeldelse. De hadde foretrukket en struktur hvor delanmeldelsene er mindre separert, slik at hele anmeldelsen ser mer ut som en naturlig og sammensatt anmeldelse.

En annen deltaker nevnte at de hadde ønsket en mer oversiktlig produktside, hvor man overordnet kan se de svake og sterke punktene ved produktet (f.eks dårlig batteri, bra skjerm osv...) basert på hvordan brukere har anmeldt produktet.



(a) Hvor ville du brukt Reviewsers?

(b) Ville du fulgt en anmelder?

Figur 20: Utvalgte spørsmål fra brukertesting

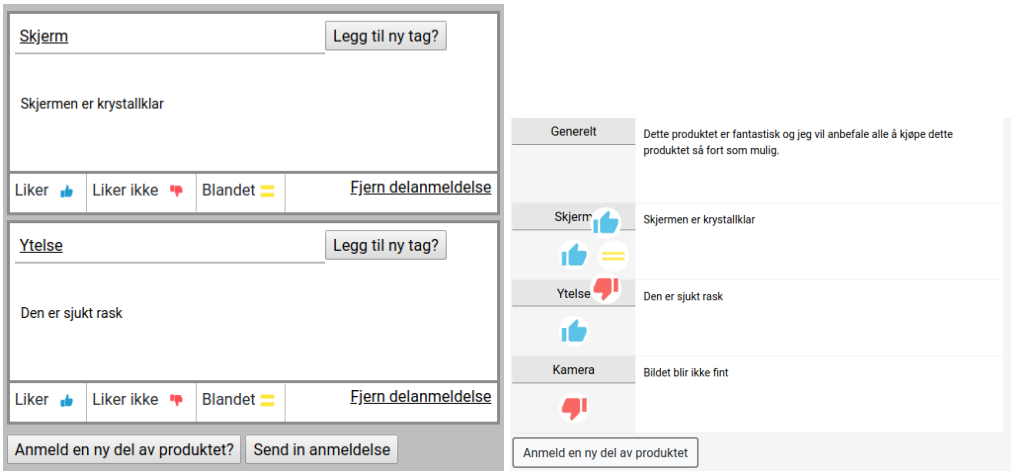
7.4.2 Forandringer som følge av brukertesting

Basert på tilbakemeldinger gjorde vi noen forandringer på løsningen. For det første forsøkte vi å forandre på strukturen og utseende for anmeldelser, slik at de ville se mer naturlige og helhetlige ut. Vi gjorde dette ved å fjerne rammer rundt hver delanmeldelse for å fjerne noe av separasjonen. Brukergrensesnittet ble også forbedret ved å bruke et

popup vindu for å velge vurdering, istedenfor tre knapper ved siden av hverandre.

Forslaget fra deltakeren om en den mer oversiktlige produktsiden ble ikke fullt implementert. Vi implementerte derimot noe liknende, hvor en rektangulær bar med prosentandelen av positive og negative vurderinger vises sammen med hver sin tagg i en nedtrekksmeny (se figur 14).

Forandringene på anmeldelsesstrukturen er vist i figur 21.



(a) PRS **før** forandringer (b) PRS **etter** forandringer

Figur 21: Forandringer gjort som følge av tilbakemeldinger

8 Utrulling

I denne delen vil vi kort diskutere hvordan løsningen eventuelt vil utrulles til et produksjonsmiljø, og krav assosiert med en slik utrulling. Vi vil se på krav til serveren som backendløsningen utrulles til, og lansering av mobilapplikasjonen på plattformets programvaremarked.

8.1 Backend server

8.1.1 Nødvendig programvare

Når vi utviklet backend, hadde vi som utgangspunkt at den skulle utrulles til en server med Linux operativsystem. Selv om all programvare brukt i backendløsningen er open-source og også tilgjengelig for Windows og kanskje vil fungere, vil vi ikke anbefale dette ettersom vi kun har testet på Linux.

Serveren krever oppsett av en MySQL database. På vår testserver brukte vi MariaDB og vil også anbefale denne for utrulling. Selve oppsettet av databasen krever lite arbeid annet enn selve installasjonen, da vi har et script skrevet i Bash (nevnt i seksjon 6.1.2). Denne vil automatisk opprette selve databasestrukturen med nødvendige tabeller.

Til slutt vil serveren trenge en installasjon av Node.js, ettersom selve API'et er utviklet med dette rammeverket. Her kreves heller ikke noe særlig tid på oppsett, takket være npm, som er en *package manager* brukt i Node.js. Npm vil automatisk installere alle avhengigheter og moduler[105] som brukes av API'et, ved å lese fra filen *package.js* som ligger i Git repositoriet.

For å oppsummere, vil serveren kreve følgende programvare installert før backendløsningen kan utrulles til server:

- Server med Linux operativsystem, med administrator tilgang for å installere nødvendig programvare (selve API'et bør derimot ikke kjøres under administrator brukeren av sikkerhetsårsaker)
- En MySQL database (f.eks MariaDB)
- Node.js

8.1.2 Pm2

I et produksjonsmiljø bør man ha en løsning for hva som skjer dersom serveren eller applikasjonen uventet krasjer eller restarter. For dette formålet vil vi anbefale en programvare skrevet i Node.js kalt *pm2*[106]. Dette er en såkalt *process manager* for Node.js, som vil kunne overvåke API-prosessen og restarte den hvis nødvendig. Pm2 vil også gjøre det mulig å se nyttig statistikk over ressursbruk som minne- og prosessorbelastning. En annen ting som gjør *pm2* veldig nyttig i et produksjonsscenario, er at den har en funksjonalitet kalt *cluster mode*[107]. Dette vil tillate å kjøre flere instanser av backend samtidig, slik at API'et kan skaleres opp for å takle flere brukere når dette blir nødvendig i framtiden.

8.1.3 Sikkerhet

For å sikre brukerinformasjonen som overføres til og fra backend, er det som nevnt i kravspesifikasjonen (seksjon 3.2.3) viktig at kommunikasjon mellom server og klient bruker SSL, slik at kommunikasjonen krypteres. Dette vil kreve at man får tak i et SSL-sertifikat for serveren som er utsendt av en *sertifikatsautoritet*. Dette er en organisasjon som godkjenner SSL-sertifikater. På testserveren brukte vi et SSL-sertifikat fra organisasjonen *Let's Encrypt* som sender ut gratis sertifikater[108].

8.1.4 Anbefalning

Vi vil anbefale at oppdragsgiveren ansetter noen for å ha ansvar for utrulling og drifting av backend serveren. Oppsettet vil kreve noe kunnskap om Linux og serverdrift generelt, samt vedlikehold når programvaren trenger oppdatering eller serveren må skaleres opp når brukermassen vokser.

8.2 Mobilapplikasjon

Når en applikasjon lanseres for Android-plattformen, er det flere ting som må tas forbehold for. Google har flere krav som en applikasjon må følge før de tillatter lansering på deres programvaremarked *Google Play*. Bryter man med reglementet, vil ikke Google tillate at applikasjonen publiseres og i verste fall kan man permanent utestenges fra plattformen[109]. Google har en sjekkliste som lister opp punkter som må gjennomgås for å sjekke at applikasjonen forholder seg til retningslinjene deres. Dette er hva de kaller *launch checklist*[110]. Før applikasjonen utrulles, må det brukes tid på å gå gjennom disse punktene.

Vi vil ikke nevne alle punktene fra denne sjekklisten her, men ta for oss det viktigiste. For det første, må applikasjonen følge Google sine retningslinjer for utviklere. Her har vi sett på hvert punkt og gjort en kort vurdering om applikasjonen følger det:

- Forbudt/Ulovlig innhold: Omhandler stort sett applikasjoner som har gambling, villedende økonomiske modeller, og ulovlig innhold. Det er også et punkt her om "brukergenerert innhold", som er relevant for Reviewsers. Her krever Google at det skal være et system for moderering av innhold, noe som vi ikke har implementert under prosjektet. Før utrulling må et slikt system bearbeides.
- Opphavsrett: Applikasjonen bryter ikke med opphavsretten. Vi har hatt skriftlig kontakt med Icecat, og fått avklart at vi kan bruke informasjonen de tilbyr oss.
- Personvern og sikkerhet: Dette må ses nærmere på før utrulling. En nøye vurdering må gjøres på om applikasjonen følger de nye personvernlovene fra EU (GDPR) (se seksjon 3.2.3). Det må også utarbeides et "vilkår for bruk" som brukeren må godta før applikasjonen kan tas i bruk.
- Inntjening og reklamasjon: Ikke relevant for Reviewsers. Vi har ingen "in-app" betaling eller reklame.
- Promotering: Her krever Google at det ikke brukes villedende måter å markedsføre applikasjonen på. Det er heller ikke lov å gi fordeler til brukere mot at de gir positive anmeldelser.
- Spam og minimumsfunksjonalitet: Google tillatter ikke applikasjoner som er ustabile, eller krasjer. For å sikre at applikasjonen ikke bryter med dette punktet må det gjøres en lengre "beta" test av applikasjonen før lansering for å fikse kodefeil.

Noe annet som må på plass før lansering er å kvalitetssikre brukergrensesnittet. Dette aspektet brukte vi ikke så mye tid på i utviklingsprosessen, og det grafiske brukergrensesnittet er derfor kun tilpasset mobiltelefoner. Det finnes flere forskjellige typer enheter som kjører Android annet enn telefoner, eksempelvis nettbrett. Før lansering må brukergrensesnittet tilpasses både mobiltelefon og nettbrett. Dette er imidlertid tilrettelagt ved bruk av *fragments* (se seksjon 5.5.1), så det vil ikke være massivt arbeid.

8.2.1 Anbefalning

Hovedsakelig er det tre ting som må fokuseres på før mobilapplikasjonen vil være klar for lansering. Det første vil være å implementere et system for moderering av anmeldelser, med moderatorbrukere eller automatiske systemer, slik at applikasjonen etterkommer Google sitt reglement for brukergenerert innhold. Det andre vil være å gjøre en lengre "beta" test hvor feil kan oppdages og fikses, og det siste er å optimalisere brukergrensesnittet for flere typer Android enheter.

8.2.2 Webløsningen

Webløsningen vil kanskje være den letteste delen når det kommer til utrulling. Det bør nevnes at webløsningen er langt fra ferdig, og var mer som en test for konseptualisering av anmeldelsessystemet. Derfor vil ikke mye nevnes på dette punktet. Når det er sagt, vil ikke webløsningen kreve noen eksterne avhengigheter annet enn Node.js, som allerede vil være satt opp når backend er på plass. Ingen ekstra webserver kreves, ettersom Node.js fungerer som en egen statisk webserver. Webløsningen kan også utrulles til samme server som backend, og det er dette vi vil anbefale oppdragsgiveren å gjøre. Da kan pm2 brukes (se seksjon 8.1.2) for en instans av backend og en av webløsningen, side om side.

9 Konklusjon

9.1 Drøftinger

9.1.1 Læringsmål

Gjennom utviklingen av prosjektet har vi hatt som mål å bruke så mye som mulig av den kunnskapen vi har opparbeidet gjennom studiet. Den første erfaringen med LaTeXhar gruppen kort opplevd allerede i første semester, men vi har aldri trengt å lære å bruke det skikkelig. For skriving av denne rapporten måtte vi da lære oss bruk av LaTeX, noe som viste seg å være ganske nyttig, og som vil hjelpe oss å skrive akademiske tekster senere i utdanningen.

Et annet godt inntrykk vi har fått er hvordan god planlegging og strukturerte arbeidsrutiner er kritisk for å oppnå et mål i et prosjekt. Gjennom de emnene vi har hatt som forklarte utviklingsprosesser, har vi bare lært teori om hvordan man kan sette opp en utviklingsmetodikk for et prosjekt. Vi har derimot aldri selv fulgt en slik metodikk og bacheloroppgaven var det første større prosjektet vi har hatt. Gruppen har i dette prosjektet sett viktigheten av god planlegging og struktur igjennom en komplisert utviklingsprosess.

Noe som ble en nyttig erfaring var brukertestingen vi fikk gjort av PRS. Vi har gjort brukertesting i et tidligere emne, men prosjektet som vi hadde i dette tilfellet var under veldig spesifikke rammer, hvor vi selv hadde lite å komme med i løsningen. Det å ha vært gjennom en brukertest med en egenutviklet applikasjon og system har vært en nyttig og utfordrende oppgave, hvor vi har fått muligheten til å ta tilbakemeldinger fra testere og videreutviklet systemet ut ifra dette.

Det har også vært lærerikt å jobbe med en reell oppdragsgiver i prosjektet. Dette fordi vi har fått et innblikk i dynamikken når det gjelder kommunikasjon mellom utvikler og kunde ute i arbeidslivet.

Kanskje noe vi ikke fikk mye bruk for, og kunne ønske å ha lært å bruke mer, er *unit tester*. De få testene vi har tatt i bruk ga oss noe innspill, men på grunn av den mer funksjonelle retningen som utviklingen har tatt, la vi mindre fokus på brukertesting enn vi kanskje burde.

9.1.2 Resultatmål

Det opprinnelige målet med prosjektet var en prototype for Android- og webapplikasjonen. Blant gruppen har det også vært ønske om at webløsningen kunne gjøres ferdig i løpet av utviklingsprosessen, noe som ble oppfattet som mulig i løpet av de første ukene.

Til slutt har vi klart å levere en bred løsning for oppgaven, hvor ved androidapplikasjonen vil det ikke kreve mye for publisering av applikasjonen. I tillegg til en helt grunnleggende prototype av en webløsning vi valgte å droppe utviklingen av midt i utviklingsprosessen. Tiden vi anslo at de forskjellige delene av utviklingen ville ta, viste seg å være for lite i de fleste tilfellene. Noen deler av androidløsningen kunne ha vært pusset opp og forbedret mer.

Uansett er vi fornøyd med det vi har klart å lage. Det grunnleggende i applikasjonen er på plass, i tillegg til en stor del ekstra funksjonalitet. Etter flere iterasjoner har de viktigste delene av applikasjonen fått det utseende vi ønsket, og som vi kan føle ikke bare er en plassholder. Bortsett fra oppussing og utvikling av en del ekstra funksjonalitet, som ikke ville ta mange flere uker, føler vi at vi ville ha oppnådd målet om ferdigstillelse av androidapplikasjonen.

9.2 Alternative valg

I løpet av prosjektet har det vært flere større valg vi måtte ta, og drøfte fordeler og ulemper med de forskjellige alternativene. Kanskje et av de største valgene vi måtte ta var om vi skulle utvikle mobilapplikasjonen som en kryssplattform webapplikasjon, eller en *native* applikasjon. Grundigere diskusjon om dette er i seksjon 4.5. Hovedgrunnen til at vi til slutt valgte native begrunnes i at vi hadde tidligere erfaring med Java, og at vi hadde et emne ved siden av hvor vi lærte native Android utvikling.

Dette betyr ikke nødvendigvis at native ville vært det beste valget. Vi ser i ettertid at applikasjonen like godt kunne fungert som en webapplikasjon. Det var forsåvidt ingen funksjonalitet som trenger spesiell tilgang til telefonens maskinvare eller noe annet som ville krevd å bruke Android API'et. Vi kunne da istedet ha hatt en applikasjon for både Android og iOS, og kunne hatt felles kodebase for webløsningen. Derimot måtte vi ha satt oss inn i webutvikling, noe gruppemedlemmene ikke hadde mye erfaring med fra før av.

En annen ting vi ser i ettertid er at vi muligens kunne gått lavere på valgt API-nivå for androidapplikasjonen. Vi valgte å gå for minimum API-nivå 19, og applikasjonen støtter nå ca 95% av alle androidtelefoner (se seksjon 3.2.1). Om vi kunne gått lavere for å støtte ytterligere enda fler hadde vi ikke tid til å teste, men trolig kunne vi valgt noen hakk lavere.

Når det kommer til valg av utviklingsmetodikk er vi heller ikke sikre på om vi gjorde det korrekte valget, hvis i hele tatt et "korrekt" valg finnes. Valget sto mellom Scrum og Kanban, og vi valgte Scrum. Med kun tre gruppemedlemmer kunne nok også den litt mindre omfattende metodikken Kanban fungert. Vi synes kanskje at bruken av Scrum kunne til tider bli litt "tungt" med alle dets utvidelser, som "Planning Poker" og "Scrum-master", som vi hadde liten bruk for. Med tre i gruppen som til stadighet har oversikt over andres arbeid og som møter ofte kunne nok det litt mer "light" rammeverket Kanban fungert vel så bra.

9.3 Framtidig Arbeid

For framtidig arbeid er det mye som oppdragsgiver må få på plass. Vi har i prosjektet utarbeidet det vi ser på som en bred løsning, med flere komplekse komponenter. Fokuset har vært å implementere grunnleggende funksjonalitet for løsningen med en fleksibel struktur. Sammen med god dokumentasjon var målet å gjøre det enklere for oppdragsgiver å videreutvikle løsningen. Under prosjektet har vi oss imellom og med oppdragsgiver diskutert en del problemer den ferdige utgaven av Reviewsers vil måtte løse. Her blir de viktigste delene å få på plass administratorfunksjonalitet, og et mer gjennomtenkt brukergrensesnitt på frontend. Et bedre brukergrensesnitt gjelder spesielt for PRS. Hvorav alle de funksjonelle kravene for anmeldelsessystemet er på plass, mener vi det fortsatt er

en del igjen å gjøre for designet av løsningen.

Administrasjonfunksjonalitet blir en viktig del for Reviewsers. Brukere kan legge inn anmeldelser, anmeldelsestagger (Se seksjon 5.4.2) og nye produkter. Det er da viktig at brukere har muligheten til å rapportere misbruk av disse funksjonalitetene. En administrasjonløsning må eksistere for å behandle denne informasjonen og annen data i Reviewsers.

Webapplikasjonen må ferdigutvikles til en helhetlig applikasjon, som skal ivareta mesteparten av funksjonaliteten til androidapplikasjonen. På web burde det fokuseres på hvordan å bedre vise informasjon enn hva man kan gjøre på en mindre skjerm som ved en mobil.

En annen utfordring diskutert tidligere (Se seksjon 6.3.6) er hvordan bedre å behandle skillet mellom produkter fra Icecat og produkter som blir lagt til av brukere. Vår nåværende løsning fungerer, men vil nok ikke være ideell i en ferdig applikasjon. Hovedproblemet blir å opprettholde en viss kvalitet på produktinformasjonen som legges til av brukere i forhold til informasjonen fra Icecat. Kvaliteten på informasjonen fra Icecat vil også variere i kvalitet, og det vil være tilfeller hvor eksisterende produkter enten ikke har bilde eller ekstra informasjon.

Utviklingen av iOS applikasjonen burde skje etter at prototypen på Android er ferdigutviklet. Vi mener dette er mer hensiktsmessig ettersom det er bedre å ha en ferdig applikasjon som kan brukes som referanse, når man skal implementere løsningen på en ny plattform.

Falske anmeldelser kan aldri helt forhindres, og det burde utvikles et system som på en best mulig måte hindrer brukere i å kunne gi ut falske anmeldelser. Et godt eksempel kan være verktøy som gis til brukere for å rapportere anmeldelser de mistenker er falske. Dette krever at administrativ funksjonalitet er på plass.

9.4 Evaluering av arbeid

Fra begynnelsen har gruppen hatt et relativt høyt arbeidstempo, noe som har bidratt til en bred løsning for oppgaven. Gjennom utviklingen har vi hatt regelmessige møter, hvor vi delte våre meninger og resultater. Arbeidsflyten vi har klart å oppnå ser vi på som smidig og effektiv, med mye jobbing sammen, hvor noen større individuelle oppgaver har blitt gjort hver for seg.

Selv om vi er ganske fornøyde med arbeidet som har blitt utført, ble ikke alt slik vi hadde planlagt, blant annet når det gjelder fordeling av arbeidsoppgaver. I den opprinnelige prosjektplanen planla vi å fordele oppgavene slik at alle gruppemedlemmene kunne alt om prosjektet. Dette var for å gjøre bacheloroppgaven mer lærerik, samt forsikre oss om at i tilfelle noe skulle skje, ville andre kunne fylle inn uten større problemer. Slik har det ikke blitt, hvor hver av gruppemedlemmene spesialiserte seg i hver sin del av prosjektet. I etterkant har vi vurdert dette målet som urealistisk. Gjennom å la gruppemedlemmene ha ansvar for spesifikke deler av prosjektet fikk vi muligheten til å fokusere på flere områder samtidig, hvor vi hele tiden hadde dialog fram og tilbake for hvordan de forskjellige elementene i oppgaven skulle jobbe sammen. Det eneste negative med denne måten å jobbe på, var at det i starten ble litt ujevn arbeidsfordeling. Dette på grunn av at mye av backend måtte settes opp før flere områder i frontend kunne utvikles på.

Den største utfordringen i utviklingen oppstod da vi forsto at vi ikke ville få tid til å

gjøre ferdig utviklingen av webløsningen. Dette var på grunn av økende krav for produktdatabase og et ønske om en mer bred applikasjon på mobil. Det at vi valgte en smidig utviklingsmodell hjalp oss å enklere endre på omfanget for prosjektet og vi fikk utarbeidet en bedre og mer realistisk plan sammen med oppdragsgiver. Vi er fornøyde med håndteringen av problemer underveis i prosjektet, og mener vi valgte en god blanding av struktur og frihet.

Vi planla å bruke *Toggl* som loggføringsverktøy under utviklingen[111]. Ingen av gruppemedlemmene hadde erfaring med loggføring av arbeid, og det ble mange feil og avvik i den første perioden av utviklingen. Ettersom vi møttes daglig, hvor vi under enhver tid hadde oversikt over hva de andre jobbet med, valgte vi å ikke lenger bruke verktøyet. Isteden for at gruppen loggfører tid sammen ble det valgt at man heller kunne bruke det selv, hvis man ønsket en bedre oversikt over tid brukt på eget arbeid. Vi ser nå mot slutten av utviklingen hvor dårlig dette valget var. Det å ha en klar oversikt over hvor mye tid hver av de forskjellige oppgavene tok for de forskjellige gruppemedlemmene, ville vært god informasjon å ha i slutten av hver uke da vi så over arbeidet vi hadde gjort. Vi tror utviklingen hadde blitt bedre hadde vi brukt verktøyet riktig.

9.5 Avslutning

Selv om det i løpet av utviklingsprosessen har oppstått noen hinder som gjorde forandring av den opprinnelige planen nødvendig, er vi fortsatt veldig fornøyd med sluttresultatet. Vi har tatt mye lærdom av prosjektet som vi vil ta med oss videre og ønsker å takke Reviewsers AS for godt samarbeid og engasjement.

Bibliografi

[1] member, G. 2018. Empty review amazon. URL: https://www.amazon.com/review/create-review/#.

[2] Mitchell, D. I. 2018. Kanban board example. URL: https://en.wikipedia.org/wiki/Kanban_(development)#/media/File:Kanban_board_example.jpg.

[3] Google. 2018. Dashboards. URL: https://developer.android.com/about/dashboards/index.html.

[4] Wikipedia. 2018. Programmeringsgrensesnitt. URL: https://no.wikipedia.org/wiki/Programmeringsgrensesnitt.

[5] Pluralsight. 2015. What's the difference between the front-end and back-end? URL: https://www.pluralsight.com/blog/film-games/whats-difference-front-end-back-end.

[6] Wikipedia. 2018. Callback (computer programming). URL: https://en.wikipedia.org/wiki/Callback_(computer_programming).

[7] Wikipedia. 2018. Create, read, update and delete. URL: https://en.wikipedia.org/wiki/Create,_read,_update_and_delete.

[8] Rossen, E. 2009. Ean international. URL: https://snl.no/EAN_International.

[9] Wikipedia. 2018. Gantt-skjema. URL: https://no.wikipedia.org/wiki/Gantt-skjema.

[10] Icecat. 2018. About icecat. URL: https://icecat.us/menu/about/index.html.

[11] Wikipedia. 2018. Åpen kildekode. URL: https://no.wikipedia.org/wiki/%C3%85pen_kildekode.

[12] 2018. Software development kit. URL: https://en.wikipedia.org/wiki/Software_development_kit.

[13] Wikipedia. 2018. Secure shell. URL: https://en.wikipedia.org/wiki/Secure_Shell.

[14] Digicert. 2018. What is ssl, tls and https? URL: https://www.websecurity.symantec.com/security-topics/what-is-ssl-tls-https.

[15] Porter, M. 2015. Verdikjede. URL: https://no.wikipedia.org/wiki/Verdikjede.

[16] Wikipedia. 2018. Michael porter. URL: https://no.wikipedia.org/wiki/Michael_Porter.

[17] Vivino. 2018. Verdikjede. URL: https://www.vivino.com/.

[18] Proff.no. 2018. Pappagallo partners as. URL: https://www.proff.no/roller/pappagallo-partners-as/h%C3%B8vik/bedriftsr%C3%A5dgivning/IF72RQY043Z/.

[19] Datatilsynet. 2018. Nye personvernregler i 2018. URL: https://www.datatilsynet.no/regelverk-og-skjema/nye-personvernregler/.

[20] Ian, S. 2015. *Software Engineering, Global Edition*. PEARSON EDUCACION. URL: https://www.amazon.com/Software-Engineering-Global-Sommerville-Ian/dp/1292096136?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1292096136.

[21] Wikipedia. 2018. Rational unified process. URL: https://en.wikipedia.org/wiki/Rational_Unified_Process.

[22] Sutherland, J., Coplien, J. O., Heasman, L., den Hollander, M., & Ramos, C. 2018. A scrum book. URL: http://scrumbook.org/value-stream/sprint.html.

[23] 2018. Kanban (development). URL: https://en.wikipedia.org/wiki/Kanban_(development).

[24] Sutherland, J., Coplien, J. O., Heasman, L., den Hollander, M., & Ramos, C. 2018. A scrum book. URL: http://scrumbook.org/product-organization-pattern-language/development-team/oyatsu-jinja.html.

[25] Slack homepage. URL: https://slack.com/.

[26] Google. 2018. Supporting different platform versions. URL: https://developer.android.com/training/basics/supporting-devices/platforms.html.

[27] Google. 2018. Support different languages and cultures. URL: https://developer.android.com/training/basics/supporting-devices/languages.

[28] Owasp. 2016. Sql injection. URL: https://www.owasp.org/index.php/SQL_Injection.

[29] Long, N. 2008. Javascript: client-side vs. server-side validation. URL: https://stackoverflow.com/a/162579.

[30] Datatilsynet. 2018. Hva betyr de nye personvernreglene for din virksomhet? URL: https://www.datatilsynet.no/regelverk-og-skjema/veiledere/hva-betyr/?id=6329.

[31] Google. 2018. Download android sdk. URL: https://developer.android.com/studio/.

[32] Google. 2018. Add c and c++ code to your project. URL: https://developer.android.com/studio/projects/add-native-code.

[33] Google. 2018. Add c and c++ code to your project. URL: https://developer.android.com/studio/projects/add-native-code.

[34] Sanchez, R. 2018. Comparing node.js vs php performance. URL: http://www.hostingadvice.com/blog/comparing-node-js-vs-php-performance/.

[35] Foundation, N. 2018. About node.js®. URL: https://nodejs.org/en/about/.

[36] Tan, K. L. Y. M. Z. 2014. Performance comparison and evaluation of web development technologies in php, python, and node.js. *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference*. URL: https://ieeexplore.ieee.org/abstract/document/7023652/.

[37] Foundation, N. 2018. Express - node.js web application framework. URL: https://expressjs.com/.

[38] Mozilla. 2018. Express/node introduction. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction.

[39] MongoDB. 2018. Nosql databases explained. URL: https://www.mongodb.com/nosql-explained.

[40] MongoDB. 2018. Json and bson. URL: https://www.mongodb.com/json-and-bson.

[41] Wikipedia. 2018. Database normalization. URL: https://en.wikipedia.org/wiki/Database_normalization.

[42] MariaDB. 2018. Mariadb versus mysql - features. URL: https://mariadb.com/kb/en/library/mariadb-vs-mysql-features/.

[43] Casteldine, E. 2011. What's so good about jquery? URL: https://www.sitepoint.com/whats-so-good-about-jquery/.

[44] EAN-Search.org. 2018. Ean-search.org - barcode database with over 130 million products. URL: https://www.ean-search.org/.

[45] upcitemdb.com. 2018. Upc lookup database with api access over 149 million unique upc numbers. URL: http://www.upcitemdb.com/.

[46] Icecat. 2018. Icecat: open feed with product information, data-sheets for ecommerce. URL: https://icecat.biz/.

[47] Native, R. React native. URL: https://facebook.github.io/react-native/.

[48] Microsoft. Xamarin. URL: https://www.xamarin.com/.

[49] Google. 2018. Progressive web app. URL: https://developers.google.com/web/progressive-web-apps/.

[50] Gartner. 2017. Gartner says worldwide sales of smartphones grew 7 percent in the fourth quarter of 2016. URL: https://www.gartner.com/newsroom/id/3609817.

[51] Hunsaker, C. 2015. Rest vs soap: When is rest better? URL: https://stormpath.com/blog/rest-vs-soap.

[52] Fielding, R. T. 2000. Representational state transfer (rest). URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[53] Wikipedia. 2018. Model–view–controller. URL: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

[54] Google. 2018. Guide to app architecture. URL: https://developer.android.com/topic/libraries/architecture/guide.

[55] Google. 2018. Material design for android. URL: https://developer.android.com/guide/topics/ui/look-and-feel/.

[56] Google. 2018. Introduction to activities. URL: https://developer.android.com/guide/components/activities/intro-activities.

[57] Google. 2018. Fragments. URL: https://developer.android.com/guide/components/fragments.

[58] Google. 2018. The color system. URL: https://material.io/design/color/the-color-system.html#color-usage-palettes.

[59] Google. 2018. Color tool. URL: https://material.io/tools/color/#!/?view.left=0&view.right=1.

[60] Git homepage. URL: https://git-scm.com/.

[61] Torvalds, L. 2007. Tech talk: Linus torvalds on git. URL: https://www.youtube.com/watch?v=4XpnKHJAok8.

[62] Atlassian. 2018. Git feature branch workflow. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow.

[63] Atlassian. 2018. Bitbucket education for student developers. URL: https://bitbucket.org/product/education.

[64] VersionOne. 2018. Versionone 12th annual state of agile report. URL: https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report.

[65] Google. 2018. Google docs. URL: https://www.google.com/docs/about/.

[66] ShareLaTeX. 2018. Sharelatex, online latex editor. URL: https://www.sharelatex.com/.

[67] Google. 2018. Android studio features. URL: https://developer.android.com/studio/features/.

[68] Knell, T. 2018. Building android projects with maven - part 1: Setup. URL: https://www.synyx.de/blog/building-android-projects-with-maven-part-1-setup/.

[69] Microsoft. 2018. Node.js tutorial in vs code. URL: https://code.visualstudio.com/docs/nodejs/nodejs-tutorial.

[70] Stackoverflow. 2013. What is the difference between sax and dom? URL: https://stackoverflow.com/a/19154095.

[71] Jonathan Ong, D. C. W. 2018. body-parser. URL: https://github.com/expressjs/body-parser.

[72] Google. 2018. org.json. URL: https://developer.android.com/reference/org/json/package-summary.

[73] Foundation, N. 2018. Express routing. URL: https://expressjs.com/en/guide/routing.html.

[74] Foundation, N. 2018. Express 4.x - api. URL: https://expressjs.com/en/api.html.

[75] Auth0. 2018. Introduction to json web tokens. URL: https://jwt.io/introduction/.

[76] Wikipedia. 2018. Base64. URL: https://en.wikipedia.org/wiki/Base64.

[77] Conti, A. 2018. Jwt token auth for express.js. URL: https://github.com/agconti/express-jwt-token.

[78] O'Hara, C. 2018. express-validator. URL: https://github.com/express-validator/express-validator.

[79] O'Hara, C. 2018. validator.js. URL: https://github.com/chriso/validator.js.

[80] Foundation, N. 2018. Using middleware. URL: https://expressjs.com/en/guide/using-middleware.html.

[81] Foundation, N. 2018. Error handling. URL: https://expressjs.com/en/guide/error-handling.html.

[82] Corporation, O. 2018. Full-text search functions. URL: https://dev.mysql.com/doc/refman/8.0/en/fulltext-search.html.

[83] erickson. 2018. What is full text search vs like. URL: https://stackoverflow.com/a/224726.

[84] Corporation, O. 2018. Information functions. URL: https://dev.mysql.com/doc/refman/8.0/en/information-functions.html.

[85] Coyle, A. 2014. Infinite scroll design pattern. URL: https://medium.com/designed-thought/infinite-scroll-design-pattern-4b473113ce09.

[86] Google. 2018. Asynctask. URL: https://developer.android.com/reference/android/os/AsyncTask.

[87] 2018. Glide v4 : Fast and efficient image loading for android. URL: https://bumptech.github.io/glide/.

[88] Google. 2018. Barcode api overview. URL: https://developers.google.com/vision/android/barcodes-overview.

[89] Google. 2018. Android vision api samples. URL: https://github.com/googlesamples/android-vision/tree/master/visionSamples/barcode-reader.

[90] Google. 2018. Control your app permissions on android 6.0 and up. URL: https://support.google.com/googleplay/answer/6270602?hl=en.

[91] Wikipedia. 2018. List of tools for static code analysis. URL: https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis.

[92] SonarSource. 2018. Continuous code quality. URL: https://www.sonarqube.org/.

[93] SpotBugs. 2018. Spotbugs is findbugs' successor. a tool for static analysis to look for bugs in java code. URL: https://github.com/spotbugs/spotbugs.

[94] FbContrib. 2018. fb-contrib™: A findbugs™ auxiliary detector plugin. URL: https://github.com/spotbugs/spotbugs.

[95] Lint, A. 2018. Extension plugin for android lint in sonarqube. URL: https://github.com/ofields/sonar-android.

[96] Google. 2018. Improve your code with lint. URL: https://developer.android.com/studio/write/lint.

[97] SonarSource. 2018. Analyzing with sonarqube scanner. URL: https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner.

[98] SonarSource. 2018. Analyzing with sonarqube scanner for gradle. URL: https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner+for+Gradle.

[99] SonarSource. 2018. Concepts. URL: https://docs.sonarqube.org/display/SONAR/Concepts.

[100] Google. 2018. Test your app. URL: https://support.google.com/googleplay/answer/6270602?hl=en.

[101] Bogdanov, V. 2016. How to set up a unified test coverage report in android with jacoco and sonarqube. URL: https://intersog.com/blog/tech-tips/how-to-set-up-a-unified-test-coverage-report-in-android-with-jacoco-and-sonarqube/.

[102] Google. 2018. Espresso. URL: https://developer.android.com/training/testing/espresso/.

[103] Swagger.io. 2018. Swagger.io. URL: https://swagger.io/.

[104] Swagger.io. 2018. Openapi specification. URL: https://swagger.io/specification/.

[105] npmjs. 2018. What is npm? URL: https://docs.npmjs.com/getting-started/what-is-npm.

[106] Keymetrics. 2018. Pm2 - advanced node.js process manager. URL: http://pm2.keymetrics.io/.

[107] Keymetrics. 2018. Pm2 - cluster mode. URL: http://pm2.keymetrics.io/docs/usage/cluster-mode/#cluster-mode.

[108] Foundation, L. 2018. Getting started - let's encrypt - free ssl/tls certificates. URL: https://letsencrypt.org/getting-started/.

[109] Google. 2018. Enforcement. URL: https://play.google.com/about/enforcement/enforcement-process/.

[110] Google. 2018. Launch checklist. URL: https://developer.android.com/distribute/best-practices/launch/launch-checklist.

[111] Toggl. 2018. Toggl - free time tracking software. URL: https://toggl.com/.

A Prosjektplan

Prosjektplan Reviewsers

Håkon Legernæs
haakoleg@stud.ntnu.no

Maciej Piatkowski
maciejp@stud.ntnu.no

Haakon Reiss Jacobsen
haakorei@stud.ntnu.no

Januar 2018

Innhold

1 MÅL OG RAMMER **1**

1.1 Bakgrunn . . . 1
1.2 Prosjektmål . . . 1
1.3 Anmeldelsessystemet . . . 3
1.4 Rammer . . . 4
1.4.1 Tidsrammer . . . 4
1.4.2 Juridiske rammer . . . 4
1.4.3 Ressursrammer . . . 4

2 OMFANG **5**

2.1 Fagområde . . . 5
2.2 Avgrensning . . . 5
2.3 Oppgavebeskrivelse . . . 5

3 PROSJEKTORGANISERING **6**

3.1 Ansvarsforhold og roller . . . 6
3.2 Rutiner og regler i gruppa . . . 6
3.2.1 Generelle regler . . . 6
3.2.2 Tidsoversikt . . . 6
3.2.3 Regelliste . . . 7

4 PLANLEGGING, OPPFØLGING OR RAPPORTERING **8**

4.1 Hovedinndeling av prosjektet . . . 8
4.1.1 Utviklingsrammeverk . . . 8
4.1.2 Andre vurderte rammeverk . . . 8
4.2 Plan for statusmøter og beslutningspunkter . . . 9
4.2.1 Veileder og oppdragsgiver . . . 9
4.2.2 Gruppemøter . . . 9

5 KVALITETSSIKRING **10**

5.1 Dokumentasjon, standardbruk og kildekode . . . 10
5.1.1 Dokumentasjon . . . 10
5.1.2 Kommentering av kode . . . 10

5.1.3 Kodestil 11
5.2 Konfigurasjonsstyring 11
5.2.1 Versjonskontroll 11
5.2.2 Byggesystem 11
5.2.3 Containere 11
5.3 Risikoanalyse 11

6 PLAN FOR GJENNOMFØRING 13
6.1 Prosjekt tidslinje 13
6.2 Milepæler og beslutningspunkter 13

7 KORT OM LØSNINGEN 17
7.1 Programvarearkitektur 17
7.1.1 Klient 17
7.1.2 Server 17

Kapittel 1

MÅL OG RAMMER

1.1 Bakgrunn

Reviewsers er ideen om et brukersamfunn bestående av produkt anmeldere, hvor alle anmeldelser er samlet på et sted. Her kan bruker lett søke etter produkter eller scanne deres barkode for å få fram anmeldelser av produktet. Denne ideen kommer ifra dagens problemer ved å finne produktanmeldelser på de produktene du er interessert i. Dette kan være på grunn av at brukeren ikke klarer å finne anmeldelser, at de er for få eller for dårlig strukturerte/informative så man ikke kan hente ut noe nyttig brukerinformasjon om produktet. Ofte må bruker ty til produktinformasjon til et produkt. Dette kan være et problem ettersom produsenten ofte vil presentere produktet fra sin beste side, og vil ikke tilby like nyttig som andre brukeres erfaring med produktet.

1.2 Prosjektmål

Vårt mål er å lage en komplett prototype av en Android og web applikasjon, hvor bruker kan søke etter, og anmelde produkter. Hvor man ved Android applikasjonen skal kunne scanne en barkode for å få opp produktet og tilhørende anmeldelser. Vi skal bruke en egen server med en mysql relasjonsdatabase hvor informasjon vil bli laget. Dette vil kreve at vi har tilgang til mange eksisterende produkter. For å bygge opp en egen produktdatabase vil vi bruke en ekstern API hvor man kan søke etter produkter, enten via navn eller barkode.

Android og webappliksajon

Bruker skal kunne opprette en bruker og logge inn på applikasjonen. Brukeren skal kunne bruke en eksisterende Facebook eller Google bruker (kan støtte flere typer hvis trengs).

Når bruker har logget inn skal bruker kunne gjøre følgende:

- Søke etter et produkt for å få opp forslag på produkter
- Anmelde produkter
- Se på andres anmeldelser for et produkt
- Scanne inn barkoden til et produkt for å få opp produkt og anmeldelser (kun Android)
- Følge andre brukere (som en type venneliste)
- Se anmeldelser til de brukerne du følger
- Hvis et produkt ikke finnes, legge inn et produkt

Tredjeparts API

Vi har tenkt til å bruke upcitemdb[**6**] som tredjeparts API for å bygge opp vår egen produktdatabase. Når en bruker skanner eller søker etter et produkt vil søket alltid gjøres via vår database. Dersom produktet ikke finnes i databasen vår vil søket gjøres via den eksterne APIen. Hvis produktet finnes der vil det automatisk bli lagt inn i vår database (dvs. cachet). Om det derimot ikke kommer opp et produkt vil brukeren få muligheten til å legge inn produktet manuelt. Dette kan imidlertid kun bli gjort gjennom skanning av strekkode.

Backend

Målet er å ha en MySQL database hvor all nødvendig informasjon blir lagret slik at en prototype av applikasjonen kan fungere. Vi kommer til å bruke Node.js server for å ta imot spørringer fra klienter og behandle informasjon fra database. Alt dette skal hostes på en egen cloud server som vi selv har full kontroll over. All kommunikasjon mellom backend og klient må bruke SSL-protokollen for sikker kommunikasjon.

Brukerrettigheter

Applikasjonen er planlagt å brukes på et internasjonalt nivå. Det er da et krav at vi følger alle lover og regler når det kommer til hvordan vi skal behandle brukerinformasjon.

All kommunikasjon mellom backend og klient må bruke SSL protokollen for sikker kommunikasjon.

Bakkompabilitet

Vi starter med å utvikle Android applikasjonen for Android versjon KitKat 4.4 og oppover som et minimum. Når vi har en ferdig prototype kan vi vurdere å støtte flere versjoner, men det vil variere på den ferdige applikasjonen.

Anmeldelsessystem

Implementere et godt strukturert anmeldelsessystem som gir brukeren verktøyene til å strukturere gode anmeldeser.

Brukertesting

Målet vil være å få testest både anmeldelsessystemet tidlig i prosessen for så å teste selve applikasjonen når det meste av hovedunskjonaliteten er på plass, for å kunne finpusse på selve applikasjonen.

Språk

Vi kommer til å ha støtte for både norsk og engelsk. Engelsk ville nok holdt, men vi synes det vil være en god ide å sette opp et system som allerede støtter flere språk. Med tanke på at det skal være en internasjonal applikasjon så burde det være lett å ekspandere til flere språk.

Arbeidsfordeling

Når prosjektet er levert er målet vårt at alle skal ha vært igjennom de forskjellige delene av prosjektet, og at ingen av oss skal spesialisere seg på ett bestemt område.

1.3 Anmeldelsessystemet

Ettersom anmeldelsessystemet kommer til å bli den viktigste delen av applikasjonen, har vi på forhånd kommet opp med en ide om hvordan vi skal strukturere opp anmeldelsene.

Selve rating systemet vil være "anbefaller produktet"og "anbefaller ikke produktet". Dette kan enten gjøres ved å skrive en medfølgende tekstbasert anmeldelse, men man kan også velge å kun gi en rating. Når et produkt vises til en bruker vil den totale ratingen være basert på prosentandelen positive mot negative ratinger. Det vil fortsatt være mulig å skrive en anmeldelse uten rating, men vil da ikke bli tatt med i den totale ratingen for produktet.

En normal tekstbasert anmeldelse vil kuttes opp i deler basert på de forskjellige elementene ved produktet. Dette vil gjøres ved at verdt produkt vil kunne deles opp i forskjellige tagsss-om kan brukes for å beskrive de forskjellige delene av produktet. Et eksempel på dette kan være en mobiltelefon som er delt opp i tags: batteri, skjerm, byggekvalitet, kamera og liknende.

Tags vil generes selv av brukeren under anmeldelsesprosessen. Dette vil gjøres ved at brukeren kan skrive sin tag som en tittel, på den delen av produktet brukeren ønsker å snakke om. Her vil det foreslås tidligere tags som andre brukere har brukt før på dette produktet.

Med dette systemet vil tags være brukergenererte, og hvert produkt vil ha egne tags. For nye produkter, eller andre produkter som ikke har noen genererte tags kan man ha en rekke foreslåtte tags. Dette kan være basert på tags tidligere brukt for en type produkt.

En full anmeldelse vil bestå av forskjellige deler hvor hver del har en tag som overskrift for å vise hva delen refererer til. Hver del vil kunne gi en positiv eller negativ rating. Alle delene til sammen vil fortsatt ha en positiv eller negativ anbefaling som spesifisert før.

Dette vil gi nye muligheter for å strukturere anmeldelser. Se for deg at en bruker ser på en pc, men lurer kun på hva folk synes om batterilevetiden, og som ikke er interessert i å lese gjennom alle de fulle anmeldelsene. Brukeren kan nå trykke på tag'en batterifor å kun få opp anmeldelser på batteriet som også vil ha sin egen totale rating.

Brukere kan fortsatt velge å skrive anmeldelsen på et normalt format med en standard ssummarytag, men vi vil prøve å legge til gode verktøy som vil gi brukeren lyst til å bruke tags.

Vi planlegger å lage en tidlig prototype av dette systemet i web applikasjonen for å få gjort en brukerundersøkelse, for så å finalisere designet av denne løsningen.

1.4 Rammer

1.4.1 Tidsrammer

Ettersom dette er en bacheloroppgave som må leveres inn på en spesifikk dato, er det noen konkrete tidsrammer vi må følge. Tidsperioden for hele oppgaven er fra midten av januar til midten av mai. Ifølge denne tidsrammen forventer vi å ha rundt 2 uker planlegging, 2 måneder med utvikling, og 6 uker med rapportskriving. På grunn av den relativt korte tidsrammen, er det viktig å sette klare begrensninger på krav og hvordan sluttproduktet skal se ut slik at vi rekker å gjøre det vi har planlagt.

1.4.2 Juridiske rammer

Applikasjonen må følge alle personvernbestemmelser som er beskrevet i lovene. Bruk av personlige opplysninger er noe som applikasjonen vil gjøre mye av, derfor er det viktig å vurdere hvilken type informasjon som bør lagres og hva det kan brukes til. Å gå gjennom disse bestemmelsene i detalj vil ta altfor lang tid, men Datatilsynet har publisert retningslinjer for programvareutviklere som kan brukes for å gjøre at applikasjonen etterkommer kravene [1].

1.4.3 Ressursrammer

Ressursmessig er det også viktig å huske at gruppen består av bare 3 utviklere. Som også har ikke mye erfaring med utvikling av Android-applikasjoner.

Kapittel 2

OMFANG

2.1 Fagområde

Oppgaven dreier seg hovedsaklig om å utvikle en Android og webapplikasjon. Det vil også utvikles en backend-del på serversiden som tilbyr nødvendige tjenester til de to klientapplikasjonene. Vi må også bruke ett eksternt tredjeparts-API for å få tak i produktinformasjon utifra strekkode, uten at vi selv trenger å bygge opp vår egen database med produkter.

2.2 Avgrensning

Hovedavgrensningen i prosjektet er tid, som begrenser oss til 3-4 måneder for utviklingsperioden. For å forsikre om at fokuset ligger på å utvikle en applikasjon som er så komplett som mulig, vil en native applikasjon for Android bli utviklet først.

2.3 Oppgavebeskrivelse

Oppgaven gitt av oppdragsgiveren dreier seg om å utvikle en applikasjon for Android og web, som skal gjøre det mulig for brukere til å laste opp sine anmeldelser om alle slags produkter. For så å samle alle anmeldelser på et sted i et eget brukersamfunn av anmeldere. Den skal være enkel og rask i bruk, følge alle lover mht personvern, samt følge standarder for utseende som finnes for platformen.

Kapittel 3

PROSJEKTORGANISERING

3.1 Ansvarsforhold og roller

Vi har bestemt at Haakon vil være prosjektlederen. Grunnen til at vi har valgt han er at han har hatt mest kontakt med oppdragsgiverne av alle medlemmene i gruppa. Ytterligere roller vil bli gitt om det blir nødvendig, men vi har bestemt at ingen i gruppen skal ha for spesialisert kunnskap. Alle gruppemedlemmer skal kunne bidra i alle deler av prosjektet med unntak av prosjektlederen som er personen som er ansvarlig for å organisere møter og lignende administrative oppgaver.

3.2 Rutiner og regler i gruppa

3.2.1 Generelle regler

Rutinen i gruppen er å møte hver arbeidsdag om morgenen for å diskutere prosjektet, og justere arbeidsmengden om nødvendig. Fra mandag til onsdag, skal alle gruppemedlemmer forvente å være tilgjengelig for andre til kl. 16:00 for å jobbe gruppevis. Dersom problemer oppstår, må disse rapporteres inn til gruppa så raskt som mulig slik at det kan avklares og arbeidet ikke stopper opp.

3.2.2 Tidsoversikt

For å loggføre tiden som er brukt på å utvikle prosjektet, har vi har bestemt oss for å bruke Toggl[5], et program for å logge tid brukt på oppgaver. Alle gruppemedlemmer er forventet til å rapportere tiden presist og jevnlig. Det er også viktig for senere arbeid på rapporten for å ha en logg over tid brukt under prosjektet.

3.2.3 Regelliste

- Dersom et gruppemedlem ikke kan komme på ett av møtene, må beskjed gis på forhånd.
- Dersom annet ikke er avtalt, må alle møte fysisk på universitetet daglig til Daily Scrum møtene.
- Hvis gruppemedlemmen merker at han blir ikke ferdig med arbeidet sitt til slutten av sprinten, må han gi beskjed snarest.
- Alt arbeid på prosjektet skal registreres før uken er over i Toggl. Dette er for å ha oversikt over tid brukt på prosjektet når rapporten skal skrives.
- Hvis noen blir syk, så skal de jobbe hjemmefra om mulig, og være tilgjengelige for andre over Slack.
- Den primære kommunikasjonsplattformen mellom gruppemedlemmene, og oppdragsgiveren, er Slack. For at meldinger skal ansees som sett av andre, skal disse gå via Slack.

Kapittel 4

PLANLEGGING, OPPFØLGING OR RAPPORTERING

4.1 Hovedinndeling av prosjektet

4.1.1 Utviklingsrammeverk

Vi har bestemt oss for å organisere vårt arbeid med det smidige utviklingsrammeverket Scrum. Denne bestemmelsen var nådd basert på hvordan vi har fordelt vårt arbeid, og hvordan Scrum passet inn i vår arbeidsplan. Muligheten til å møte daglig, koordinere arbeidet basert på ferdigheter til hver gruppemedlem, og justere arbeidsmengden mens vi jobber er noe som var veldig viktig i vårt valg. Daglige møter, samtaler om prosjektet og status på vårt arbeid, i tillegg til en strukturert arbeidsfordeling sier veldig mye om hva Scrum og Daglige Scrum-møter er, noe som gjorde valget naturlig. Vi har bestemt oss for å sette lengden på sprintene til 14 dager.

Dette er parametrene vi har valgt for Scrum.

- Scrum Master: Haakon
- Sprint Lengde: To uker
- Daglige møter: 15 minutter hver dag
- Estimering: Story Points

4.1.2 Andre vurderte rammeverk

En annen arbeidsmetodikk vi har vurdert var Kanban, men vi har bestemt oss for å ikke bruke den på grunn av dets mindre strukturerte arbeidsfordeling. Vi hadde tidlig bestemt at møter

og arbeid som en gruppe ville vært nyttig, både for å forbedre arbeidshastigheten, og for å oppdage problemer med løsningen eller gruppemedlemmene. Gitt at Kanban promoterer ett mer uavhengig arbeidsmiljø, kunne det vise seg å være problematisk om gruppen fikk problemer. For å stabilisere arbeidsmiljøet vårt og for å gå med det mer sikre valget, valgte vi å droppe Kanban som vår arbeidsmetodikk.

4.2 Plan for statusmøter og beslutningspunkter

4.2.1 Veileder og oppdragsgiver

Vi har avtalt med veilederen og oppdragsgiveren å møtes annenhver mandag for å diskutere status/framgang og avklare ting vi er usikre på. Møtet med veilederen finner sted kl 11:00 i veilederens kontor, vi møter så oppdragsgiveren kl 16:00 samme dag på avtalt sted på campus.

4.2.2 Gruppemøter

Gruppen møter hver dag kl 9:00 for *Daily Scrum* (møte definert i Scrum). Her oppdateres alle gruppemedlemmer på status for arbeid som er gjennomført, og hva som er igjen å gjøre i nåværende Sprint-periode, samt hvilket arbeid hvert gruppemedlem har planer om å gjøre denne dagen. Hvis noen støter på hvilket som helst problem under gjennomføring av sine oppgaver eller bugs i koden vil dette også tas opp her, slik at det kan avklares så fort som mulig.

Når sprintperioden er over (dvs. annenhver mandag) vil et lengre møte kalt *Planning Meeting* holdes. Dette er estimert til å vare i 2 timer og hensikten med dette møtet er å planlegge neste sprintperiode ved å opprette nye arbeidsoppgaver i backlog og tildele disse over neste sprint. Vi har opprettet en tidstabell som gir oversikt over hele arbeidsuka. (Se tabell 4.1).

Mandag	Tirsdag	Onsdag	Torsdag	Fredag
9:00-11:00 (Annenhver) Planning Meeting **11:00-11:30** Veileder **11:30-16:00** Planning and Arbeid **16:00** (Annenhver) Oppdragsgiver	**9:00-9:15** Daily Scrum **9:15-16:00** Arbeid	**9:00-9:15** Daily Scrum **9:15-16:00** Arbeid	**8:15-12:00** Mobile Prog. **12:00-12:15** Daily Scrum **12:15-16:00** Arbeid	**9:00-9:15** Daily Scrum **9:15-12:00** Arbeid **12:00-14:00** Mobile Prog. **14:00-16:00** Prof. Prog.

Tabell 4.1: Tidstabell for møter

Kapittel 5

KVALITETSSIKRING

5.1 Dokumentasjon, standardbruk og kildekode

5.1.1 Dokumentasjon

Vi har bestemt oss for å bruke løsningen Confluence utviklet av Atlassian for å organisere det meste av dokumentasjonen for prosjektet. Dokumentasjonen vil være i form av Wiki-sider. Dokumentasjonen vi skriver vil inkludere teknisk dokumentasjon om kode/arkitektur og bruk av backend API. Mye av den tekniske dokumentasjonen vil være auto-generert av løsninger som javadoc og jsdoc. På slutten av utviklingsperioden må også mer brukervennlig sluttbruker-dokumentasjon utvikles, som forklarer hvordan løsningen brukes og dens funksjonalitet.

5.1.2 Kommentering av kode

Å ha en standard for dokumentasjon og kommentering i kildekoden er essensielt slik at hvert gruppemedlem kan forstå hverandres kode. For å ha en standard for dette vil vi bruke javadoc-syntaks når vi skriver Java-kode i Android Studio, og for JavaScript bruker vi jsdoc-syntaks. Hvert gruppemedlem må gjøre seg kjent med disse formene for kommentering før utvikling kan starte. Som en generell regel har vi bestemt at alle filer med kode skal ha noen setninger øverst i filen som beskriver overordnet hva koden gjør, og hver public funksjon/metode skal være kommentert med parametre og returverdier beskrevet. Hvis det ikke er intuitivt hva koden gjør bør det også være kommentarer inne i funksjonen.

5.1.3 Kodestil

For konsistens og for å lettere kunne lese hverandres kode er det viktig å ha en standard for kodesyntaks og stil. For å gjøre det enkelt vil vi bare bruke Google sin *"Java Style Guide"*[3] for Java-kode, For JavaScript vil vi på samme måte bruke *"JavaScript Style Guide"*[4], også fra Google. Disse må ikke følges til punkt og prikke da de er veldig detaljerte, men det mest grunnleggende må følges.

5.2 Konfigurasjonsstyring

5.2.1 Versjonskontroll

En løsning for versjonskontroll er viktig slik at alle gruppemedlemmene enkelt kan samarbeide på prosjektet og ha oversikt over forandringer. Det er også viktig slik at vi kan organisere versjoner av programmet og tilbakestille forandringer. Her har vi valgt å bruke Git på et privat repository på BitBucket. Når ny funksjonalitet er under utvikling skal en ny branch"opprettes i Git, og senere "merget"med master etter det er klargjort at forandringene i koden fungerer som det skal.

5.2.2 Byggesystem

For å automatisere byggeprosessen for Android bruker vi Gradle. Dette er standard når en utvikler i Android Studio. For server/backend delen som utvikles i Node.js, brukes NPM (Node Package Manager) for å organisere nødvendige biblioteker og å skille test- og deployment-miljø.

5.2.3 Containere

For å enkelt kunne teste ut og utplassere backend-komponenten og web-appen på en vilkårlig server, vil vi plassere dem i en *container* ved å bruke Docker[2]. Dette gjør at vi kan utplassere backend- og webkomponenten på hvilken som helst server som kjører Linux. Det gjør også at alle gruppemedlemmer har nøyaktig det samme testmiljøet selv om noen kjører forskjellig operativsystem osv, som er fordelaktig.

5.3 Risikoanalyse

En oversikt over identifiserte risikoer og tiltak er inkludert (Se tabell 5.1).

Risiko	Sannsynlighet	Konsekvens	Tiltak
Prosjektet tar lenger tid enn forventet.	middels	kritisk	Grundig planlegging og å avgrense prosjektet (for eksempel begrense til bare Android).
Bugs og kodefeil stopper utviklingsprosessen	lav	høy	Finn hjelp fort hvis du støter på ett problem. Gruppemedlemmer bør lese seg opp på debugging i det utviklingsmiljøet de bruker.
Spaghetti-kode og/eller uoversiktlig kodebase	middels	høy	Følge kodestandarder som er satt, og følge patterns som vi er blitt enige om å bruke. Bruk av jevnlig refaktorering av kode.
Gruppemedlem forlater gruppa	lav	kritisk	Ingen reelle tiltak. Spesialisering av kunnskap må holdes på ett minimum.
For mye spesialisering/fragmentering av kunnskap nødvendig for utvikling	middels	middels	Arbeidsoppgaver i sprinter bør tildeles slik att hvert gruppemedlem får jobbet på oppgaver fra alle kunnskapsfelt.
Prosjektet krever kunnskap som tar for lang tid å tilegne seg	lav	kritisk	I dette tilfelle må prosjektkrav endres.
Teknologier/løsninger valgt møter ikke prosjektkrav	lav	kritisk	Gjøre nøye research på prosjektkrav og løsninger for å være sikre på at de møter kravene.
Tredjeparts-API (for søking av strekkode) møter ikke kravene	høy	middels	Her må vi utvikle vår egen løsning, hvor brukere kan legge inn produkter manuelt.

Tabell 5.1: Risikoanalyse

Kapittel 6

PLAN FOR GJENNOMFØRING

6.1 Prosjekt tidslinje

Vi har laget et gantt-diagram (Se figur 6.2) som viser prosjektets tidslinje ved bruk av det åpne programmet GanttProject. Sprinter og høy-nivå oversikt over forventede oppgaver i hvert sprint er inkludert i diagrammet.

6.2 Milepæler og beslutningspunkter

24. Jan - Grunnleggende database

Her skal vi ha foreløpig modell for databasen klargjort og implementert i form av SQL setninger.

4. Feb - Grunnleggende Android- og backendapplikasjon

Innen denne datoen regner vi med å ha fått på plass det aller mest grunnleggende av funksjonalitet i Android-applikasjonen og backend. Dette betyr:

- Brukerinnlogging (gjennom backend APIet)
- Mulighet for å scanne en strekkode og få ut dataen (men ikke noe mer)
- Enkel søk etter produkter og filtrering etter hovedkategoriene

19. Feb - Grunnleggende anmeldelse-funksjonalitet

For denne milepælen forventer vi å ha gjort ferdig grunnleggende funksjonalitet for innlegging av anmeldelser, samt visning av søkeresultater i sitt eget view, og en produktside med tilhørende anmeldelser.

- Integrering av tredjeparts strekkode-API (upcitemdb). Dette betyr at vi kobler den opp med applikasjonen så vi vil få opp resultater fra strekkode-API'en, og da at produkter blir lagt inn i vår database.
- View for et produkt med tilhørende anmeldelser pluss nødvendig funksjonalitet for å tillate dette.
- View for søkeresultater. Vil vise søkeresultater (produkter) i eget view. Hvor ved å trykke på et produkt i lisen vil føre deg til produkt view
- Designe og implementere et strukturert anmeldelsessystem på web applikasjonen

Etter dette bør vi ha grunnlag for brukertesting av anmeldelse-systemet.

5. Mars - All hovedfunksjonalitet ferdig implementert og anmeldelsessystem ferdig utviklet

- All hovedfunksjonalitet vil være ferdig utviklet (i forhold til kravspesifikasjon).
- Anmeldelse-systemet kjørt gjennom brukertesting og ferdigutviklet med grunnlag i tilbakemeldinger fra brukertesting

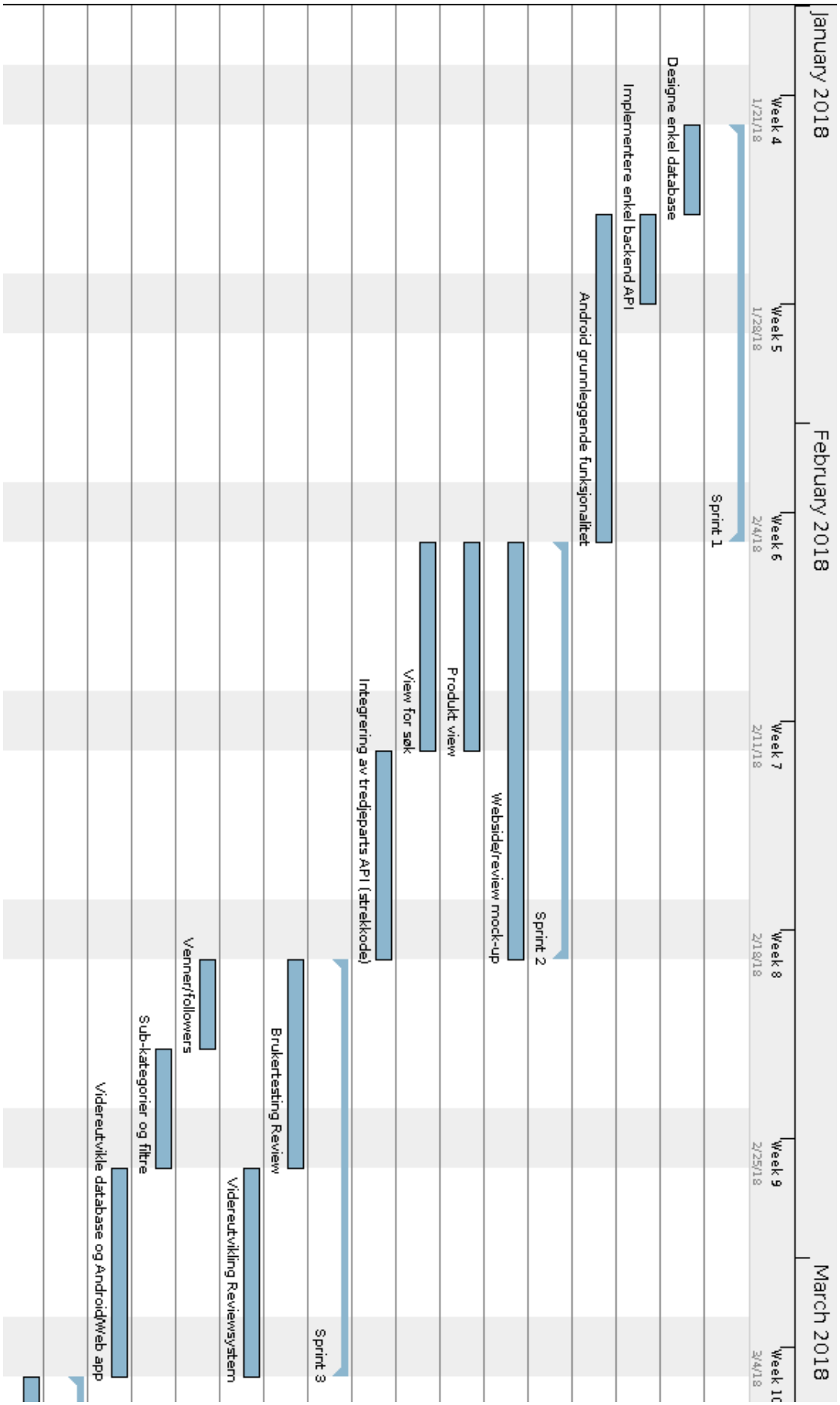
Dette vil bety at prototypen er klar for brukertesting

19. Mars - Prototype har blitt testet

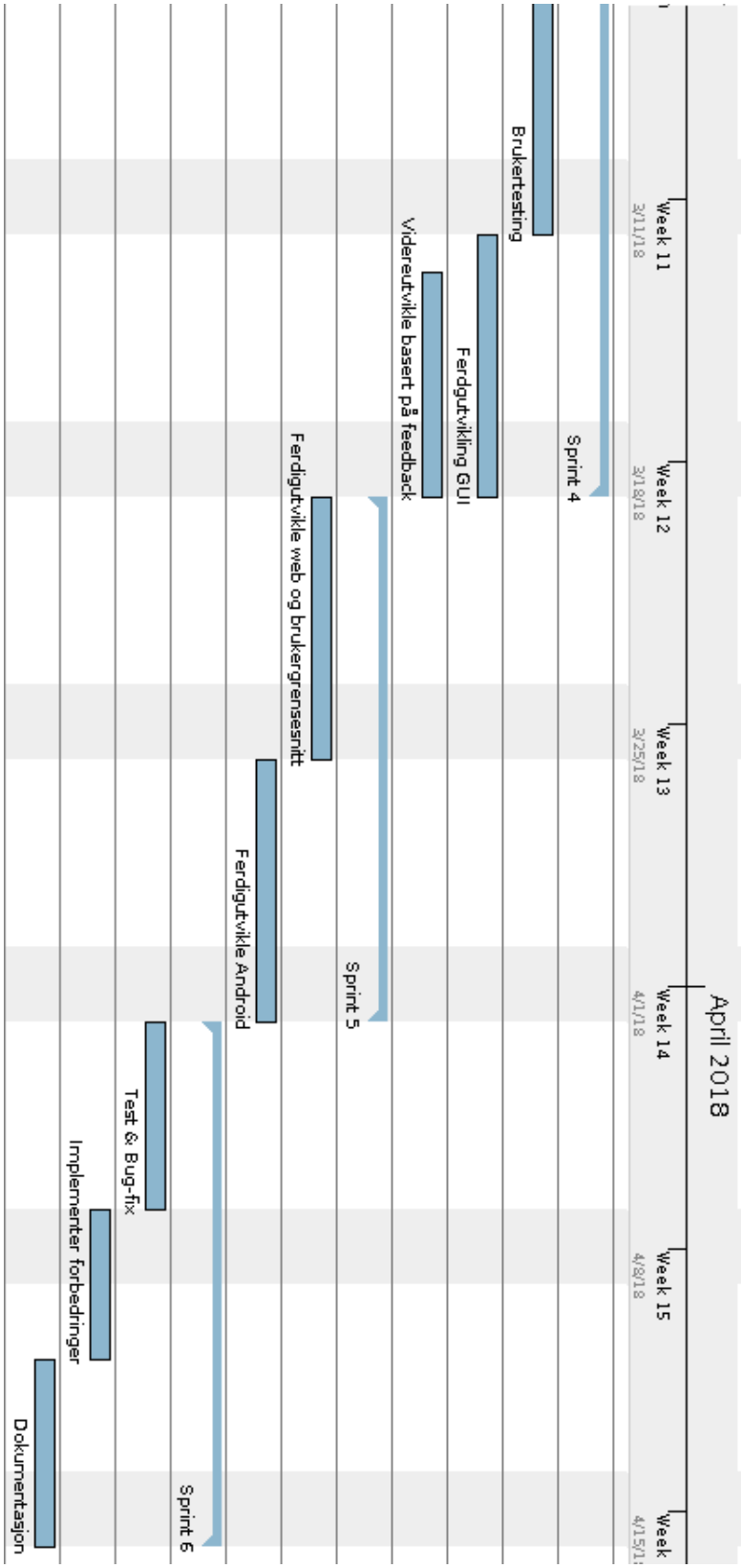
Vi skal her ha gjennomført brukertesting av prototypen, det vil si at vi har fått tilbakemeldinger fra befolkningen på hvordan applikasjonen. Denne tilbakemeldingen skal vi så bruke til å utvikle et bedre brukergrensesnitt og ekstra funksjonalitet vi anser som nødvendig for optimal brukeropplevelse.

2. April - Android- og Web applikasjoner ferdig

Før denne datoen er omme forventer vi å ha på plass både all planlagt funksjonalitet og brukergrensesnitt. Her vil applikasjonen anses for å være ferdig", dvs. vi vil ha kommet til planlagt sluttstadie hvor utvikling stanser og testing begynner for å finne bugs og muligheter til forbedringer.



Figur 6.1: Oversikt over prosjekt tidslinjen i Gantt-Diagram del 1



Figur 6.2: Oversikt over prosjekt tidslinjen i Gantt-Diagram del 2

Kapittel 7

KORT OM LØSNINGEN

7.1 Programvarearkitektur

Prosjektet vil bestå av tre hoveddeler: en web og mobil applikasjon, og en backend på serversiden. De to klientapplikasjonene vil kommunisere med den samme serveren og databasen ved hjelp av en API på serveren/backend som tilbyr nødvendig funksjonalitet til klientene.

7.1.1 Klient

Vi har bestemt oss til å bruke MVC (Model View Controller) arkitekturen for Android-appen, hvor applikasjonen er delt opp i tre lag, en for presentasjon, en for sammenkobling mellom presentasjon og modell, og en annen for forretningslogikk. Dette var ett av kravene fra kravspesifikasjonen, og det gjør det lett å utvide applikasjonen senere.

7.1.2 Server

Server/backend vil bli utviklet ved hjelp av Node.js. Kommunikasjon mellom klienten (app, web) vil bli gjort ved bruk av HTTP requests (GET, POST, DELETE etc...). Serveren vil fungere som ett RESTful API klienten kan bruke for å hente data fra databasen, innlogging/autentisering osv... Det vil bli bruk SSL/HTTPS for kryptering, og data vil bli formattert ved bruk av JSON i hver forespørsel til og fra serveren.

For databasen, vil vi bruke en tradisjonell MySQL-database. Dette var bestemt fordi datamodellen vi har for vår applikasjon anses til å være mest intuitivt til å implementere i en relasjonell database. I tillegg er spørrespråket til SQL-baserte databaser brukbar til å genere statistikker fra dataen, som var ett av kravene i kravspesifikasjonen.

Bibliografi

[1] Datatilsynet. *Programvareutvikling med innebygd personvern*. 2017. URL: https://www.datatilsynet.no/regelverk-og-skjema/veiledere/programvareutvikling-med-innebygd-personvern/.

[2] Docker. *Docker - Build, Ship, and Run Any App, Anywhere*. URL: https://www.docker.com/.

[3] Google. *Google Java Style Guide*. URL: https://google.github.io/styleguide/javaguide.html.

[4] Google. *Google JavaScript Style Guide*. URL: https://google.github.io/styleguide/jsguide.html.

[5] Toggl. *Toggl - Time Tracker & Employee Timesheet Software*. URL: https://toggl.com/.

B Prosjektavtale

NTNU

Norges teknisk-naturvitenskapelige universitet

Vår dato Vår referanse

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

Pappagallo Partners AS

_____ (oppdragsgiver), og

Maciej Piotrowski

Hjalmar Lyngnes

Haakon Reiss-Isaksen _____ (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra _____ til_____ .

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra NTNU på Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): ØIVIND KOLLOCH

Oppdragsgivers kontaktperson (navn): PER OVE R. NÆSSET

Student(er) (signatur): Haakon Reiv Jacobsen dato 15/1 18

Halvor Løsnæs dato 15/1 18

Maciej Piotrowski dato 15/1 18

dato

Oppdragsgiver (signatur): dato 5/1-18

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): dato

C Kode

```
#!/bin/bash
# Script to perform initial setup of the database

# CONFIG
DBUSER=admin
DBPASS=SKJULT
DBNAME=reviewsers

if [ -z $(command -v mysql) ] ; then
    echo "MySQL must be installed"
    exit 1
fi
if [ -z $(command -v npm) ] ; then
    echo "Node.js and npm must be installed"
    exit 1
fi

# Create the database
echo "Enter MySQL root password"
mysql -u root -p -e "
    CREATE DATABASE ${DBNAME};
    CREATE USER '${DBUSER}'@'localhost' IDENTIFIED BY '${DBPASS}';
    GRANT ALL PRIVILEGES ON ${DBNAME}.* TO '${DBUSER}'@'localhost';
    FLUSH PRIVILEGES;
    SET GLOBAL innodb_file_format=Barracuda;
    SET GLOBAL innodb_file_per_table=1;
    SET GLOBAL innodb_compression_algorithm=zlib;"

# Import database structure
mysql -u${DBUSER} -p${DBPASS} $DBNAME < database.sql

# Import the data from IceCat
echo "Importing IceCat..."
cd icecat_importer
npm install
npm start "SuppliersList.xml" "CategoriesList.xml" "files.index.xml"

# Import sample data
cd ..
mysql -u${DBUSER} -p${DBPASS} $DBNAME < sampledata.sql
```

Listing 21: Utdrag fra scriptet som importerer databasen

D Gradle fil

```
//Sonar + JaCoCo report start
plugins {
    id "org.sonarqube" version "2.6.2"
}

sonarqube {
    properties {
        def libraries = project.android.sdkDirectory.getPath() +
                "/platforms/android-26/android.jar," +
                "build/intermediates/classes-jar/debug/classes.jar"
        ...
        property "sonar.sources", "src/main/java,src/main/res"
        property "sonar.binaries", "build/intermediates/classes/debug"
        property "sonar.libraries", libraries
        property "sonar.java.binaries", "build/intermediates/classes/debug"
        property "sonar.java.libraries", libraries
        property "sonar.tests", "src/test/java"
        property "sonar.java.test.binaries",
                   "build/intermediates/classes/debug"
        property "sonar.java.test.libraries", libraries
        property "sonar.scm.provider", "git"
        property "sonar.jacoco.reportPaths",
                fileTree(dir: project.projectDir, includes: ['**/*.exec']) +
                fileTree(dir: project.projectDir, includes: ['**/*.ec'])
        property "sonar.java.coveragePlugin", "jacoco"
        property "sonar.junit.reportsPath", "build/test-results/testDebugUnitTest"
        // path to junit reports
        property "sonar.android.lint.report", "build/outputs/lint-results-Debug.xml"
        // path to lint reports
    }
}

apply plugin: 'jacoco'
jacoco {
    toolVersion = "0.8.1"
}

task jacocoTestReportCombine(type: JacocoReport,
        dependsOn: ['testDebugUnitTest', 'createDebugCoverageReport']) {
    reports {
```

```
        xml.enabled = true
        html.enabled = true
    }
    def fileFilter = ['**/R.class', '**/R$*.class',
                '**/BuildConfig.*', '**/Manifest*.*',
                '**/*Test*.*', 'android/**/*.*']
    def debugTree = fileTree(dir: "${buildDir}/intermediates/classes/debug",
                excludes: fileFilter)
    def mainSrc = "${project.projectDir}/src/main/java"
    sourceDirectories = files([mainSrc])
    classDirectories = files([debugTree])
    executionData = fileTree(dir: project.projectDir, includes: ['**/*.exec']) +
            fileTree(dir: project.projectDir, includes: ['**/*.ec'])
}
//Specific Sonar + JaCoCo setup end

apply plugin: 'com.android.application'
android {
    compileSdkVersion 27
    defaultConfig {
        ...
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            testCoverageEnabled false   //Necessary for Sonar + JaCoCo
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                    'proguard-rules.pro'
        }
        debug {
            testCoverageEnabled true    //Necessary for Sonar + JaCoCo
        }
    }
    testOptions {
        unitTests{
            includeAndroidResources = true
            returnDefaultValues = true
        }
    }
}
tasks.withType(Test) {
    jacoco.includeNoLocationClasses = true
}
...
```

E Notater

Fredag 02.03

Vi diskuterte planleggingen av de tre neste sprintene.

Ting vi må få på plass: SSL / HTTPS Redesign av review på Android og Web Homeskjerm på mobil Product view Produktsøk basert på dybde

Ting vi må ha ferdig i det ferdige prosjektet Koble sammen eksisterende funksjonalitet Følge EU reglene for behandling av brukerinformasjon

Søndag 04.03

Diskusjon rundt bruk av RUP for å ytterligere tilrettelegge for mulig funksjonalitet i fremtiden. Vi bestemte oss for å bruke Swagger.io som dokumentasjonsverktøy for APIet.

Ting å ta opp i planleggingsmøtet: Toggle Dokumentasjon av kode Ferdig bilde av funksjonalitet som skal være på plass innen levering

Mandag 05.03

Vi diskuterte måter vi kan forbedre produktsøk slik at det kommer opp ”iphone” når man søker på ”iphone” og ikke tilbehøret. Gjør home activity til default activity på Android, og kun åpne login når man ikke er logga inn. Vi bestemte oss for å begynne å lage use-case diagram siden det meste av funksjonaliteten er på plass. Det bør være slik at mest brukte tagger for et produkt er de som blir foreslått til brukeren. Koble brukerid opp mot tag, for å finne ut hvem som har kødda?

Onsdag 07.03

I dag diskuterte vi hvordan product view på android skal struktureres. Vi bestemte å vise en ”carousel” view for bilder, hvor man kan swipe mellom produktbilder, og det skal vises en expandable tabell med spesifikasjoner. Vi snakket om funksjonaliteten hvor brukeren kan legge inn nytt produkt. Vi ser for oss at vi kan lage en funksjon som lar brukeren legge inn et nytt produkt, hvor man fyller ut kategori, legger ved bilde og lignende.

Tirsdag 13.03

Vi kom fram til at en viktig ting vi må få på plass er pagination når man viser anmeldelser, fordi det kan potensielt være flere hundre anmeldelser på ett produkt. Må også ha en måte å sortere anmeldelser etter tag slik at brukere kan lese om spesifikke deler av produktet.

Fredag 16.03

Vi diskuterte hvilken funksjonalitet som var viktigst for å gjøres ferdig. Dette er å forbedre søkealgoritmen, kanskje senere gjøre noe reasearch på å finne en bedre framtidig løsning. For å bedre søkeresultat bør de sorteres etter popularitet (antall clicks). Vi vurderte å legge til engelsk språk i android men kom fram til at det ikke var verdt tiden. Bestemte å legge inn funksjonalitet hvor man kan dele en review med personer på kontaktlista eller lignende.

Mandag 19.03

Vi tenker å finpusse på eksisterende funksjonalitet og legge til kun hjemskjerm og slette bruker. Annen funksjonalitet vil bli å sortere reviews og sortere produkter etter mest sett på (mest klicks). Etter dette tenker vi å gjøre brukertesting på hele applikasjonen. Dette vil være for rapporten sin del.

Spørsmål til veileder: Om han synes vi trenger å kunne legge til nytt produkt i applikasjonen, eller om vi skal begrense det til rapporten

Om han synes vi burde bruke mer tid på web-applikasjonen. Om vi burde gjøre noe mer enn kun å vise et eksempel på et system for å skrive og vise anmeldelser

Kommer fram til at brukertesting vil være relevant før/under rapportfasen

Kutte funksjonalitet for å legge inn et nytt produkt. Da kun skrive om hvordan vi tenker å gjøre dette i rapporten

Tirsdag 03.04

Kontakte oppdragsgiver om møte

Torsdag 05.04

Vi diskuteterte hvordan sorteringen og organiseringen av tags og dets reviews med like dislike og nøytral skal sorteres.

Mandag 16.04

I dag hadde vi en diskusjon rundt hvilket spørsmål vi skal stille veileder og oppdragsgiver. Vi bestemte oss for at vi burde legge til en funksjon hvor brukeren får all sin info og alle reviews i et tekstformat. Dette er nødvendig pga. GDPR. Noe vi ikke har er en melding som snakker i et enkelt språk om hvilke data vi kommer til å lagre og hva vi skal bruke den til.

Spørsmål til veileder: Du vet nå hva vi har gjort under prosjektet. Hva er det du synes vi burde fokusere på i rapporten, eller om det er noe vi burde ta med som vi ikke har med.

Hvor viktig er designet av applikasjonen. Vi har fått forslag fra en profesjonell over ikoner og farger.

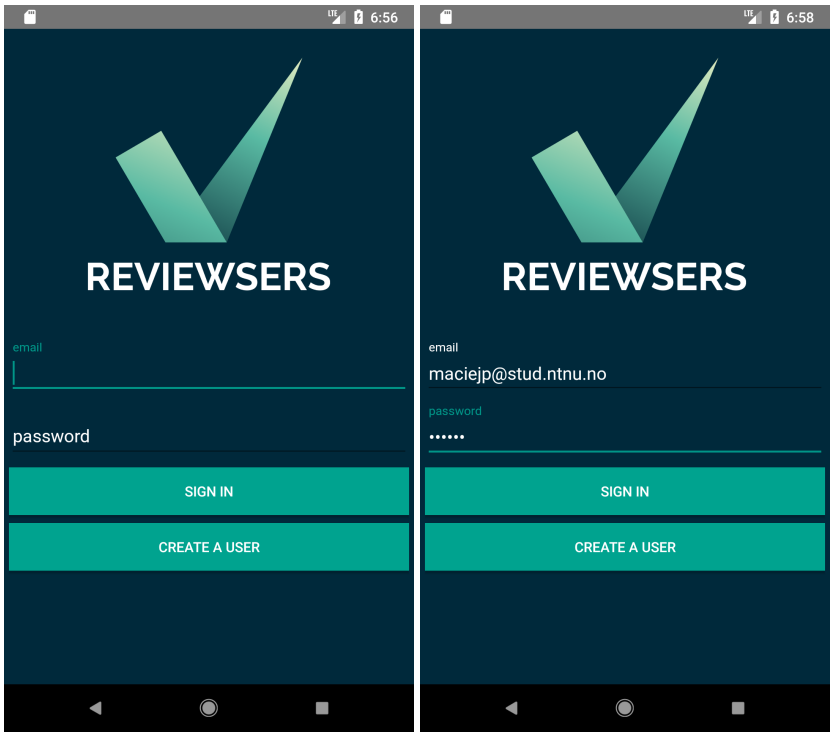
Spørre igjen om kompleksiteten.

Spørsmål til oppdragsgiver: Applikasjonen er ikke ferdig Anbefaler videreutvikling av android Designet. Trenger ikonene Forslag til å legge inn en bedre beskrivelse av anmeldelsessystemet

F Spørsmål til brukertesting

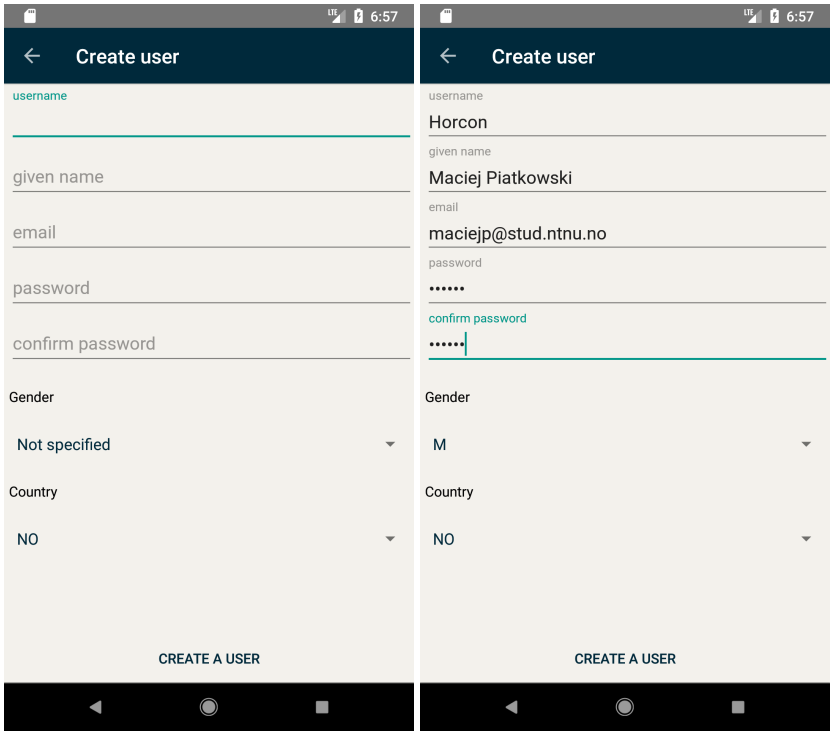
- Hvordan synes du det var å anmelde?
- Var det lett å legge inn nye tags?
- Hva synes du om å bruke ”tags” til å dele opp anmeldelsen?
- Hvordan var det å bruke tags for å se anmeldelser om deler av produktet?
- Var det nyttig å ha en egen rating (positiv, negativ, nøtral) for hver tag?
- Var det oversiktlig å se anmeldelser?
- Føler du at å dele opp anmeldelsene i mindre deler gjør at du lettere kan finne informasjon?
- Hva finner du mer naturlig, klikke på samme tag-knappen for å fjerne filtrering, eller trykke på ”Se alle anmeldelser”?
- Hvis du hadde lest en god anmeldelse, kunne du sett for deg å følge denne personen for å se mer anmeldelser fra ham?
- Hva synes du om navnet på applikasjonen? (Reviewsers)
- Ville du selv valgt et annet navn på prosjektet? Hvis ja, hvilket?
- Hva synes du om konseptet ved å dele opp anmeldelser i flere delanmeldelser?
- Ville du brukt en app/nettside dedikert for anmeldelser? Hvis ja, hvor ville du brukt det?
- Andre ting du vil gi tilbakemelding på?

G Bilder av androidapplikasjonen



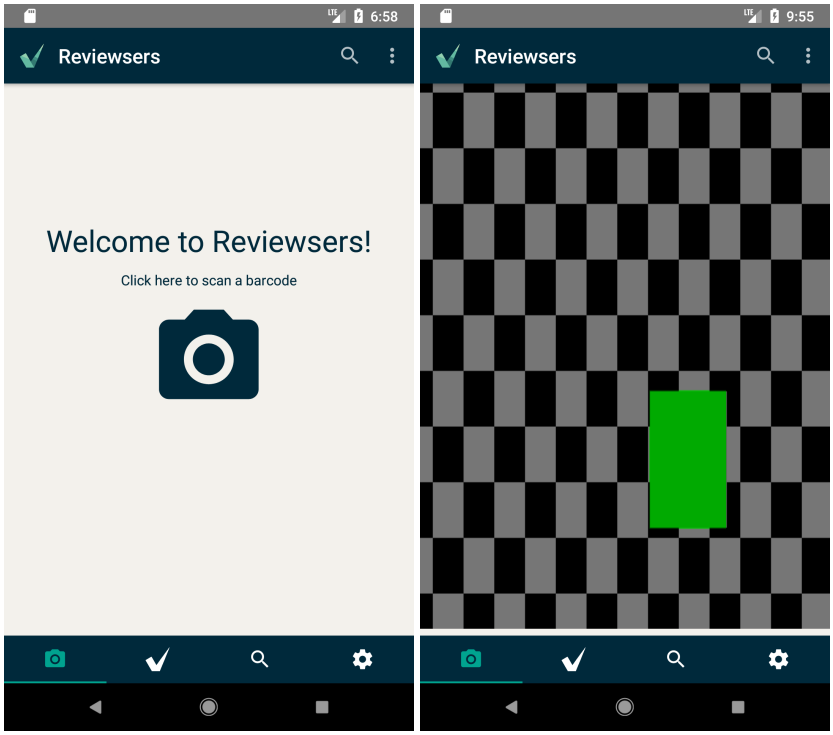
(a) Innlogging tom (b) Innlogging utfylt

Figur 22: Bilder fra androidapplikasjonen



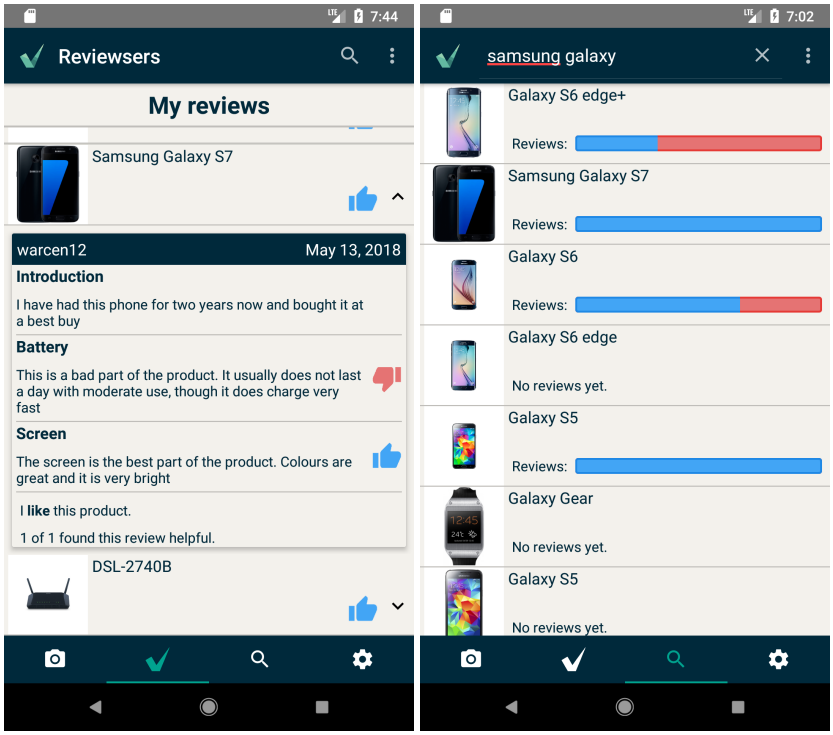
(a) Registrering tom (b) Registrering utfylt

Figur 23: Bilder fra androidapplikasjonen



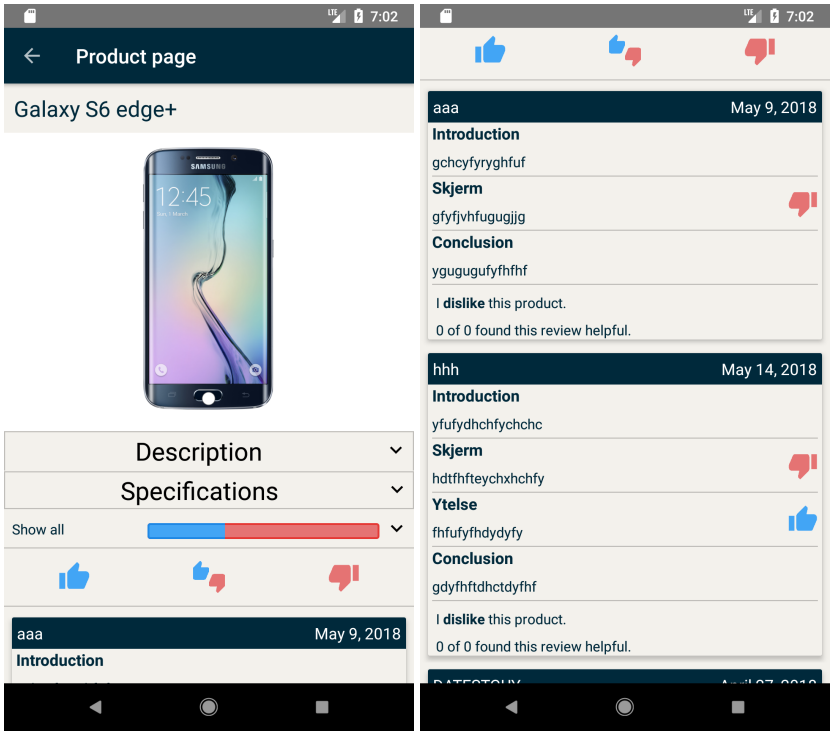
(a) Hjemsiden (b) Skanning av produkter

Figur 24: Bilder fra androidapplikasjonen



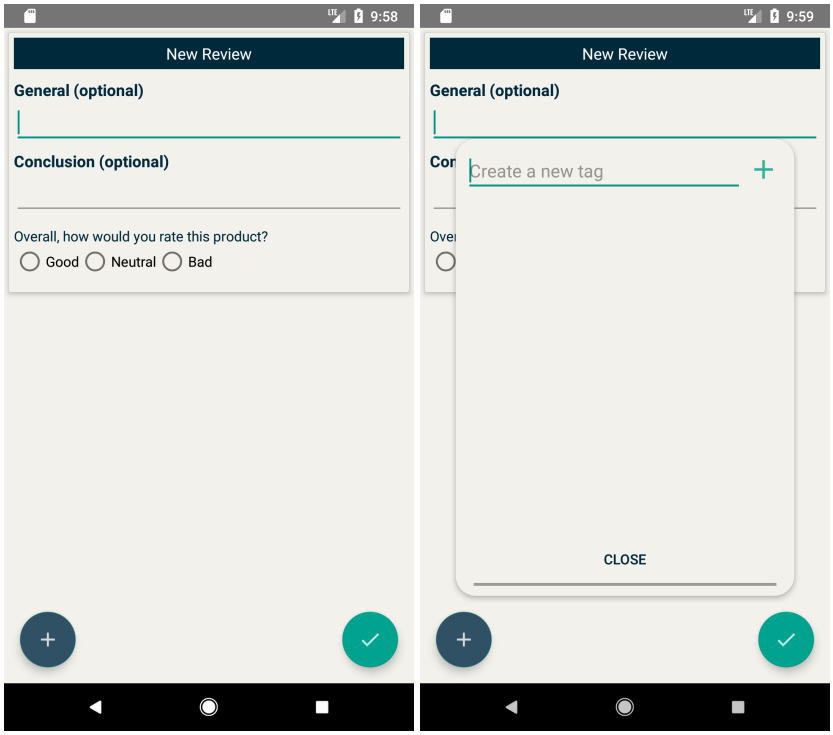
(a) Brukerens egne anmeldelser (b) Søk etter produkter

Figur 25: Bilder fra androidapplikasjonen



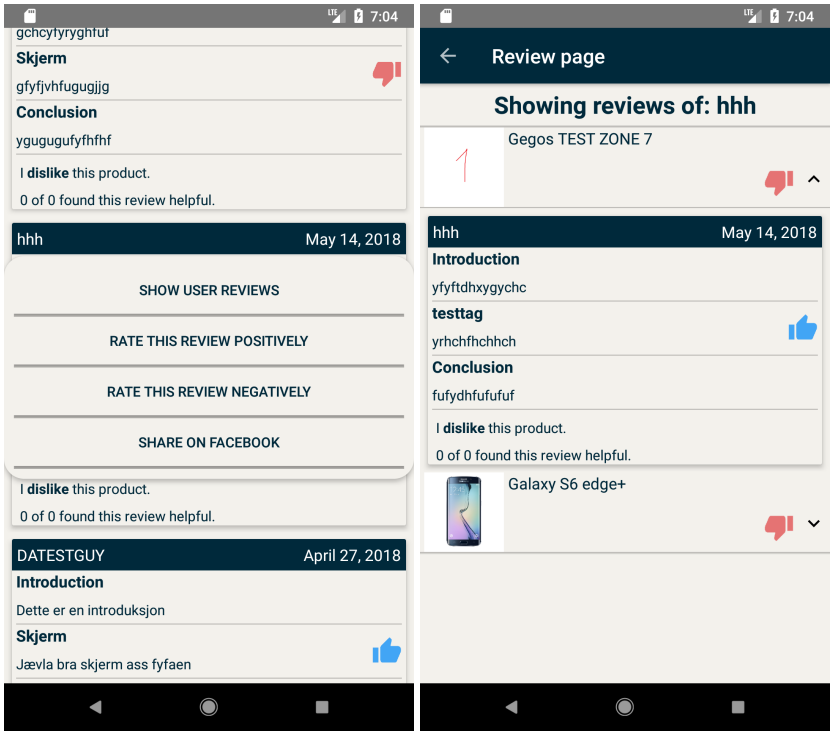
(a) Produktside (b) Anmeldelser

Figur 26: Bilder fra androidapplikasjonen



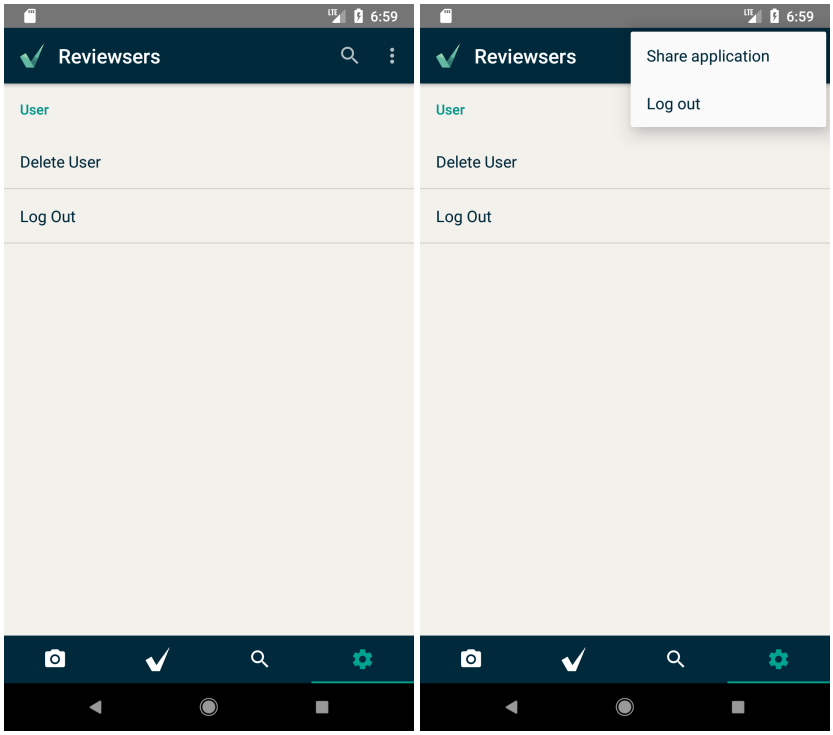
(a) Skriving av anmeldelser (b) Innlegging av nye tags

Figur 27: Bilder fra androidapplikasjonen



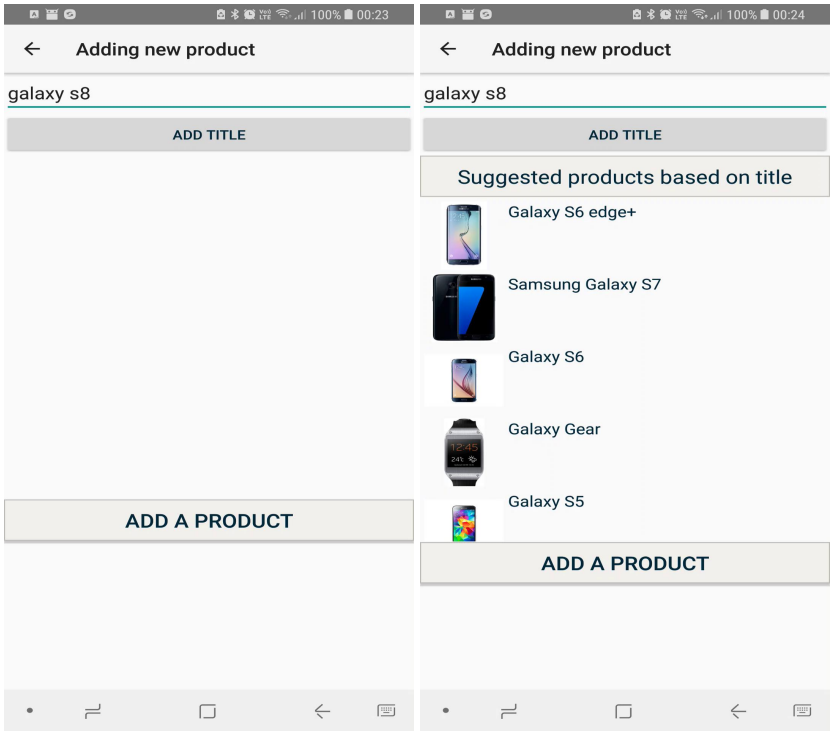
(a) Klikk på anmeldelse (b) Innlogging utfylt

Figur 28: Bilder fra androidapplikasjonen



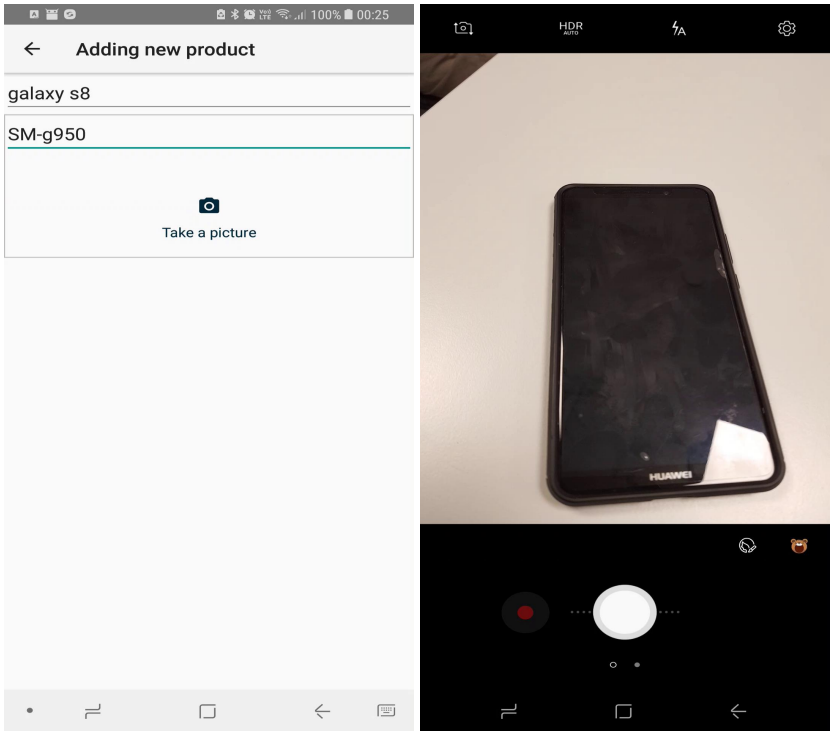
(a) Innstillinger (b) Verktøykasse

Figur 29: Bilder fra androidapplikasjonen



(a) Nytt produkt 1 (b) Verktøykasse

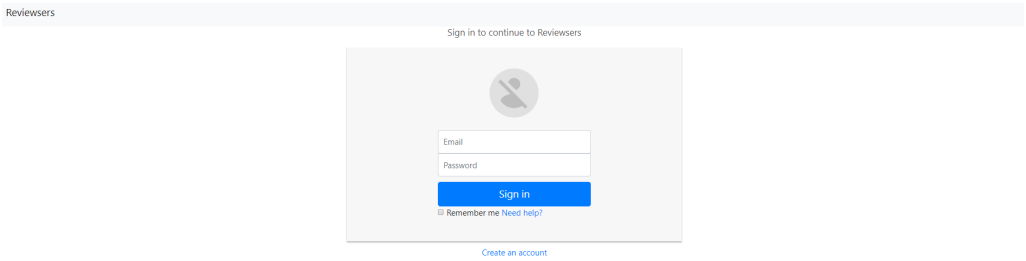
Figur 30: Nytt produkt 2



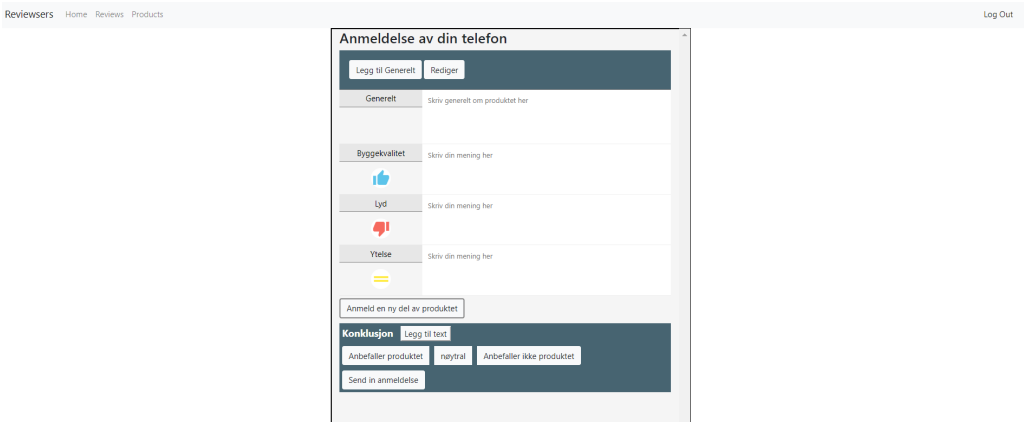
(a) Nytt produkt 3 (b) Verktøykasse

Figur 31: Nytt produkt 4

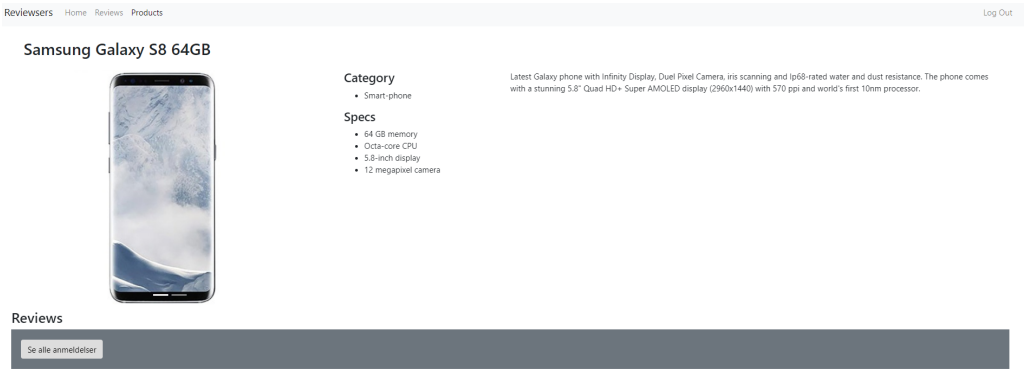
H Bilder av webapplikasjon



Figur 32: Login web

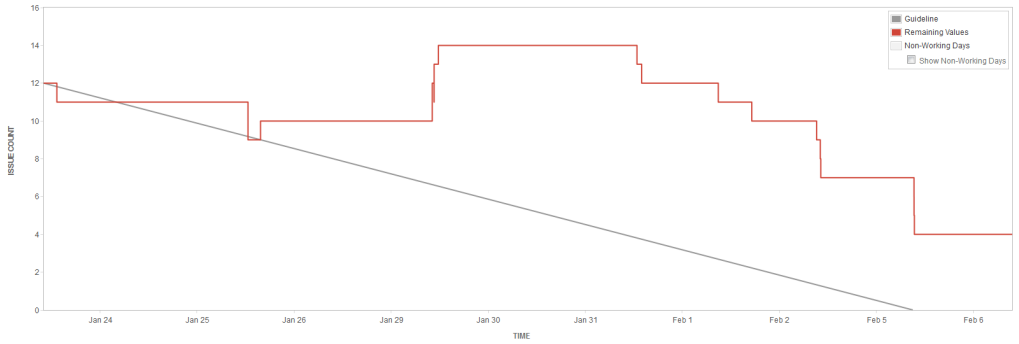


Figur 33: PRS web

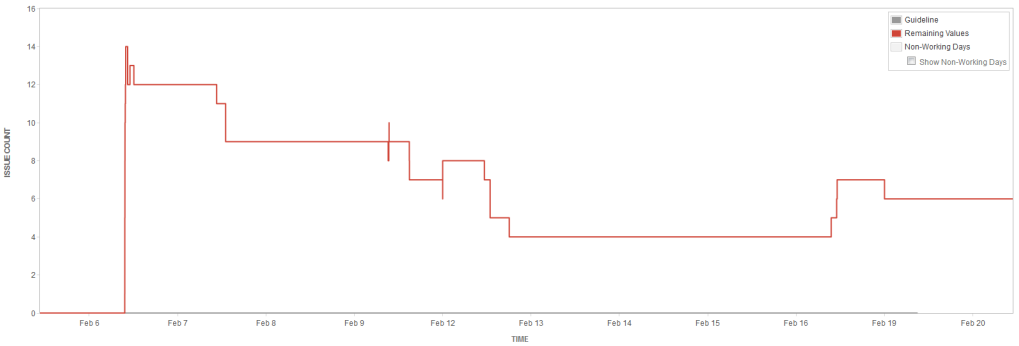


Figur 34: Product View Web

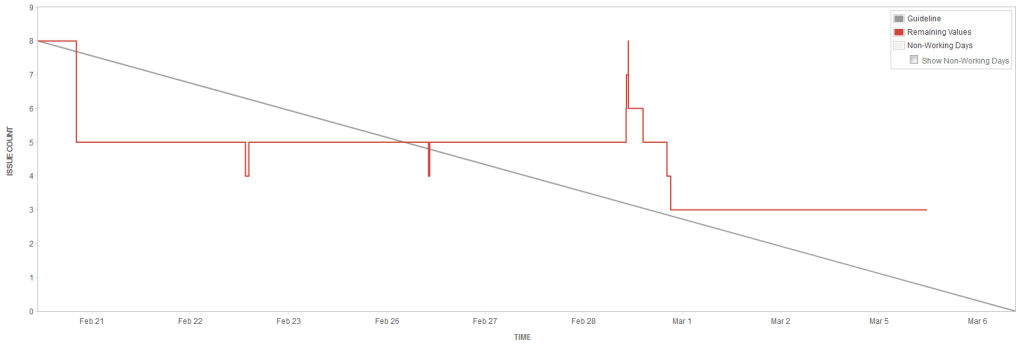
I Gjennomgang av sprinter under utvikling



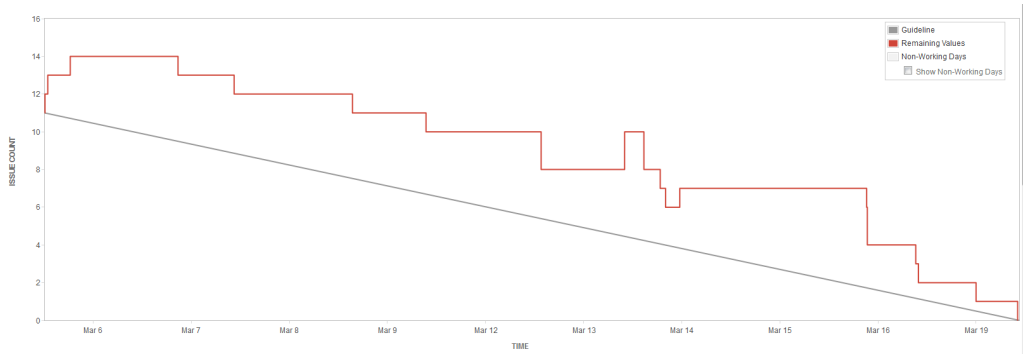
Figur 35: Sprint 1



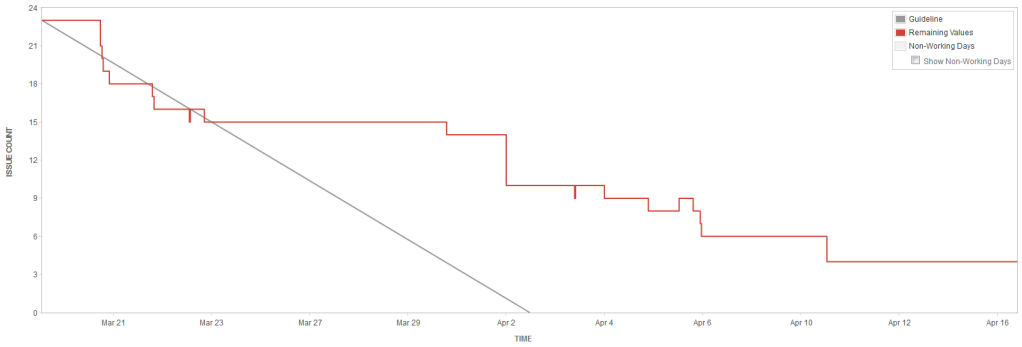
Figur 36: Sprint 2



Figur 37: Sprint 3



Figur 38: Sprint 4



Figur 39: Sprint 5