NTNU

Norwegian University of
Science and Technology

Viten i senter

Forfattere
Steffen Granberg
Steinar Opphus
Ole André Slettum

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

12.05.2017

Veileder Tom Røise

Sammendrag av Bacheloroppgaven

Tittel:	**Viten i senter**
Dato:	12.05.2017
Deltakere:	Steffen Granberg Steinar Opphus Ole André Slettum
Veiledere:	Tom Røise
Oppdragsgiver:	Vitensenteret Innlandet
Kontaktperson:	Bodil Hansen
Nøkkelord:	Programmering, Vitensenteret, Web, API, SCRUM
Antall sider:	77
Antall vedlegg:	12
Tilgjengelighet:	Åpen

Sammendrag:

I en moderne verden er det viktig å være oppdatert på teknologi og mobilapplikasjoner er veldig populære. Vitensenteret kom til oss med et ønske om å utvikle mobilapplikasjoner som kunne gi brukeropplevelsen på Vitensenteret en ekstra dimensjon. Løsningen er to mobilapplikasjoner hvor man kan skanne eksperimentstasjoner på Vitensenteret samt et webgrensesnitt hvor de ansatte kan administrere løsningen og legge inn informasjon, bilder og hint knyttet til eksperimentstasjoner. Løsningen inneholder også et backend API skrevet i .NET Core. Med dette prosjektet har vi levert en løsning som kan testes i et reelt produksjonsmiljø og etterhvert lanseres. Vi har hatt fokus på profesjonell arbeidsmetodikk, ved for eksempel bruk av scrum og Jira.

Summary of Graduate Project

Title:	**Viten i senter**
Date:	12.05.2017
Authors:	Steffen Granberg Steinar Opphus Ole André Slettum
Supervisor:	Tom Røise
Employer:	Vitensenteret Innlandet
Contact Person:	Bodil Hansen
Keywords:	Programmering, Vitensenteret, Web, API, SCRUM
Pages:	77
Attachments:	12
Availability:	Open

Abstract: In a modern world it's important to be up to date with technology and mobile applications are very popular. Vitensenteret came to us wanting us to develop mobile applications that could give the user experience at Vitensenteret an extra dimension. The solution includes two mobile applications for iOS and Android with which you can scan experiment stations at Vitensenteret and a web interface where employees at Vitensenteret can administer the solution and add pictures, hints and information about experiment stations. The solution also includes a backend API written in .NET Core. With this project we have delivered a solution that can be tested in a real production environment and eventually released to the public. One of our main focus areas has been implementing professional development methodologies, for instance by using scrum and Jira.

Forord

Denne bacheloroppgaven er skrevet ved Institutt for Informatikk og Datateknikk ved NTNU i Gjøvik av Steinar Opphus, Steffen Granberg og Ole André Slettum.

Vi ønsker å takke vår veileder Tom Røise for meget godt samarbeid og god veiledning. Vi ønsker også å takke Vitensenteret Innlandet, spesielt vår kontaktperson Bodil Hansen for god oppfølging og tilgjengelighet underveis i prosjektet. Videre ønsker vi å takke alle ansatte ved NTNU i Gjøvik som har hjulpet oss med små og store problemer underveis i prosjektet.

Til slutt vil vi takke hverandre for et godt og konstruktivt samarbeid gjennom mange uker. Dette har vært en lærerik og givende prosess.

Innhold

Figurer

Tabeller

Forkortelser

API Application Programming Interface.

CSS Cascading Style Sheets.

HTML HyperText Markup Language.

IDE Integrated Development Environment.

JSON JavaScript Object Notation.

ORM Object-Relation Mapper.

QR QR-kode.

RUP Rational Unified Process.

UML Unified Modeling Language.

Ordliste

backend er i programvareutviklingssammenheng ofte betegnelsen på serveren, eller tjenestelaget i f.eks. lagdelingsmodellen[4] . 6, 9, 12, 19, 20, 23, 28, 29, 30, 35, 38, 51, 58, 63, 65, 70, 154

backlog (*product backlog*) er en liste over alle arbeidsoppgaver som må utføres for å utvikle programvare eller et annet produkt. Det kan være *features*, *bug-fixes* eller andre oppgaver[5] . 8, 9, 11

beacon er en enveissender som blir brukt til å markere viktige plasser eller objekter. Den sender et *bluetooth*-signal med jevne mellomrom, som enheter i nærheten mottar[6] . 3, 9, 13, 15, 21, 43, 76, 77, 153

callback er kode (funksjoner) som er sendt med som parameter til annen kode, og som er forventet å bli kalt på et senere tidspunkt[7] . 39, 41

dependency injection er en metode for å oppnå løs kobling mellom objekter. Istedenfor å initialisere objekter i klassen, eller refererer statisk til dem, skytes objektene inn som en *dependency* under runtime[8] . 31, 33

endpoint er inngangspunktet til en tjeneste, prosess eller kø i tjenesteorientert arkitektur[9]. I vårt tilfelle brukes det som en beskrivelse av en URL som en API-tjeneste kan nås på . 12, 18, 35, 40, 51, 55, 56, 67, 70

frontend er i programvareutviklingssammenheng ofte betegnelsen på klienten, eller presentasjonslaget i f.eks. lagdelingsmodellen[4] . 6, 9, 12, 19, 20, 51, 54, 55, 56, 62

Object-Relation Mapper er et objekt-orientert system som konverterer databasetabeller til klasser, tabellrader til objekter og celler til objekt-attributter[10] . xi, 20, 33

pipeline er flyten av en forespørsel gjennom alle lagene med logikk i backend[1] . 29, 36

QR-kode (Quick Response Code) er en mosaikkode, eller todimensjonal strekkode, som kan leses av alle mobiler med kamera og rett applikasjon. En QR-kode kan lagre et stort antall alfanumeriske tegn, og brukes ofte til å lagre nettadresser eller kontaktinfo[11] . xi, 9, 13, 15, 21, 40, 47, 65

Rational Unified Process er et iterativ systemutviklingsrammeverk[12] . xi, 8

repositories se repository . 23, 32

repository er en datastruktur i et versjonskontrollsystem (slik som Git) som lagrer metadata for et sett med filer og/eller kataloger[13]. I backend .NET Core brukes ordet repository for å beskrive klasser som utfører databaseoppdateringer, dette kan være å legge til, fjerne eller endre objekter i databasen. . xii, 30, 32

RESTful (Representational state transfer) er en webtjeneste som sørger for interoperabilitet mellom datamaskiner og internett[14] . 6, 23

routing er dirigering av trafikk på en nettside. I Polymer betyr det å sende brukeren til et bestemt sted basert på URL . 12, 24

scrum er et iterativt og inkrementelt smidig rammeverk for å utvikle komplekse informasjonssystemer[5] . 6, 8, 9, 10, 58

sprint er en konkret fase i utviklingen i et smidig utviklingsprosjekt, ofte med en varighet mellom én og tre uker[15] . 6, 8, 50, 51, 60, 64

Unified Modeling Language er en industristandard for datarelatert modellering[16] . xi

use case er en liste med hendelser i et handlingsforløp, vanligvis brukt for å definere interaksjonen mellom en aktør og et system. *Use cases* blir ofte skissert i et UML-diagram[17] . 8, 14

workshop er et kort og intensivt arbeidsseminar innenfor et avgrenset fagområde[18] . 12, 50, 60

1 Introduksjon

1.1 Bakgrunn

Vitensenteret Innlandet er et populærvitenskapelig opplevelses- og læringssenter innen matematikk, naturvitenskap og teknologi. Senteret er lokalisert på Gjøvik og er et av ti offisielle vitensentre i Norge. De søker å inspirere og lære barn og ungdom mer om realfag.

Hovedvirket til Vitensenteret er å ta imot skoleklasser for undervisningsopplegg rundt de forskjellige realfagene. Senteret er åpent for publikum i helger og ferier, hvor de besøkende kan eksperimentere på en rekke stasjoner som er knyttet til tema som matematikk, astronomi, biologi og landbruk. Mer om de forskjellige typer besøk i seksjon 1.2.

Vitensenteret Innlandet ønsker nå å utvikle en mobilapplikasjon som kan bidra til å heve besøksopplevelsen. De ser at teknologi er viktig for de besøkende, og tilbyr blant annet gratis trådløst internett. Vitensenteret tror at en mobilapplikasjon vil gjøre at de besøkende får en enda bedre opplevelse, og at det kan føre til hyppigere besøk. Ambisjonen for bruk av applikasjonen er at den besøkende drar hjem fra senteret med et ønske om å utforske realfag videre. Applikasjonen vil gi brukerne mulighet til å fortsette utforskningen etter endt besøk.

1.2 Besøkende

Senteret er åpent for publikum i helger og ferier. Resten av året er Vitensenteret et pedagogisk læringssenter der barnehager, skoler og andre foreninger kan bestille pedagogiske opplegg innenfor realfag av kvalifiserte og flinke pedagoger ansatt på Vitensenteret.

Vi skal nå beskrive hvordan disse besøkende, både helge-, ferie- og skolebesøk foregår. Dette håper vi gir et innblikk i hvordan en mobilapplikasjon kan være et supplement til forskjellige typer besøk.

1.2.1 Pedagogiske besøk

I ukedagene er det i hovedsak skoler og barnehager som bruker Vitensenteret. Vitensenteret tilbyr tema innenfor en rekke realfagsområder som matematikk, astronomi, kjemi, biologi og landbruk med mer. Vitensenteret bruker drama som metode, noe som vil si at hvert tema innledes med en dramatisering. Dette kan enten være et besøk fra en kjent vitenskapsmann som Gallileo Gallilei eller at en kriminalteknikker trenger hjelp på laboratoriet.

De fleste skolebesøk har fast opplegg i 1,5 time, men noen av besøkene kan avsluttes med fri lek i senteret. Dette vil da foregå på samme måte som et vanlig helgebesøk, hvor alle eksperimentene kan brukes - ofte innenfor grunntemaet for besøket.

1.2.2 Helger og ferier

I helger og ferier er Vitensenteret åpent fra klokken 11:00 til 16:00. Da er hele senteret tilgjengelig og man kan leke og bruke de forskjellige eksperimentstasjonene.

Familier kommer ofte sammen og besøker senteret i alt fra én time til hele dagen. Lunsj kan nytes i kafeen.

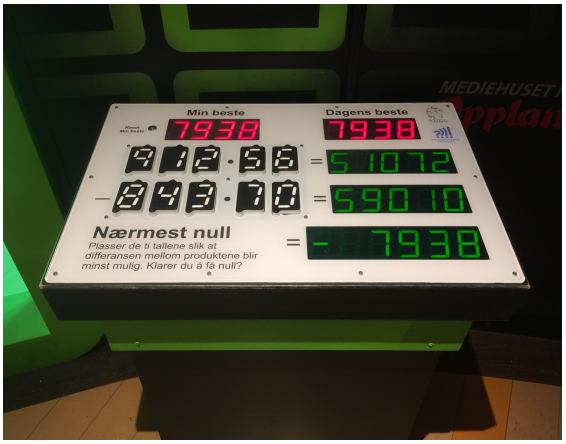
Gjennom dagen er det faste tidspunkter for når det skjer ting i senteret. I helger og ferier er det planetarievisning hver dag klokken 13:00 og i ferier er det ofte eksperimentering med de ansatte klokken 12:00. Planetarievisning er en reise i verdensrommet til Vitensenteret hvor man kan se nøyaktig hvordan stjernehimmelen er eller blir samme kveld.

1.2.3 Filosofi

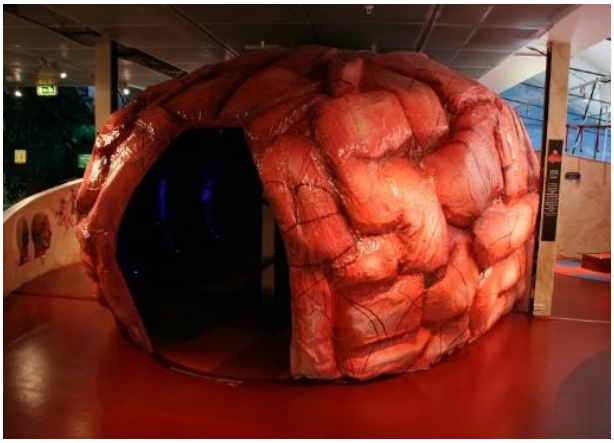
Vitensenterets filosofi er at man lærer best gjennom *hands-on* eksperimentering og i møte mellom mennesker, derfor er det minimalt med tekst og informasjon på eksperimentene. Eksperimentene har som mål å være mest mulig selvforklarende. Dette gjør terskelen lavere for å prøve seg frem og finne ut hvordan noe virker.

Gjennom mobilapplikasjonen ønsker Vitensenteret å videreføre denne filosofien ved at det læres gjennom bruk av digitale eksperimenter. I tillegg ser de at noen ønsker enda mer informasjon og tekst som supplement, gjennom en mobilapplikasjon kan man nå ut til de som er nysgjerrige uten at det går på bekostning av senterets filosofi.

1.3 Hva er et eksperiment?



(a) Mattespillet Nærmest Null i Origo



(b) Hjernen

Figur 1: Eksempler på eksperimentstasjoner

Gjennomgående i rapporten kommer vi tilbake til utrykkene eksperiment og eksperimentstasjon. Under følger en nærmere forklaring av disse begrepene.

På Vitensenteret kan publikum gå rundt i senteret å eksperimentere i de forskjellige områdene. Disse områdene representerer ofte en gren eller et tema innenfor realfag. Eksempler på ulike områder er Origo (matematikk), Landbruk, Hjernen, Astronomi, Bioteknologi og lignende.

Alle disse områdene har mange stasjoner som vi kaller eksperimenter eller eksperimentstasjoner. Disse varierer i størrelse og omfang, men alle har som mål at de besøkende skal lære noe om området de befinner seg i - ofte gjennom *hands-on* utprøvning.

En slik stasjon kan være alt fra et mattespill (Figur 1a) i Origo, Løpebane hvor man kan teste hvor raskt man løper 15 meter eller en stor hjerne (Figur 1b) de besøkende kan gå inn i. Det er ingen fasit på hvordan en stasjon er utformet.

1.4 Prosjektbeskrivelse

Vi skal levere en mobilapplikasjon som besøkende på Vitensenteret Innlandet kan bruke både under og etter besøket. Applikasjonen vil kunne brukes på telefoner med iOS og Android. Det skal også utvikles en webapplikasjon som gjør det mulig for ansatte ved Vitensenteret å oppdatere, slette og endre informasjon på en enkel måte. Mobilapplikasjonen skal ha følgende funksjonalitet:

- Brukeren skal kunne skanne et eksperiment og få informasjon om muligheter knyttet til dette eksperimentet.
    - Informasjon om eksperimentet som navn, bruk og hint.
    - Bruke telefonen som en del av eksperimentet, ved interaksjon eller med digitale versjoner.

    - Få tilgang til enkelte eksperimenter og spill hjemmefra.
- Informasjonsside med informasjon om Vitensenteret.
- Mulighet til å gjennomføre en quiz der spørsmålene er knyttet til ting på Vitensenteret.
- Topplister for quiz.
- Bruke lokaliseringsteknologier, for å gi stedspesifikk informasjon.

Webapplikasjonen skal brukes for å administrere all informasjon og funksjonalitet i mobilapplikasjonen. Den skal ha følgende funksjonalitet:

- Legge til en ny eksperimentstasjon.
- Legge til og endre informasjon om eksperimentstasjonene.
    - Informasjon om stasjonen.
    - Hvordan stasjonen brukes og eventuelle hint.
    - Plassering av stasjonen (eksperimentstasjonene flyttes av og til rundt på senteret).
    - Legge til eller endre bilder knyttet til stasjonen.
- Opprette og endre informasjon om de ulike områdene/rommene i senteret.
- Legge inn eller endre bilder knyttet til de forskjellige områdene i senteret.

1.5 Rammer

Sammen med Vitensenteret har vi utarbeidet noen rammer for gjennomføringen av prosjektet. Det er blant annet viktig at det er bakoverkompabilitet i systemene, da en del barn og ungdom ikke har tilgang til de nyeste teknologiene. Dette gjør at vi ønsker å gå så langt tilbake på versjoner av operativsystem som vi kan.

- Webapplikasjonen skal fungere på nettlesere som støtter HTML5. Dette vil si nyere versjoner av Chrome, Firefox, Safari og minimum Internet Explorer 9.
- Android applikasjonen skal fungere fra og med telefoner med API level 17.
    - Per 7. Januar 2017 har gjennomsnittlig 81.5% av Android-brukere operativsystemversjon som minst støtter API level 18[19]. Android-applikasjonen vil støtte API level 18 og nyere. Dekningsgraden av antall enheter er på verdensbasis.
    - Siden vi har tenkt å bruke teknologier som beacons, kan vi ikke gå lavere enn API level 18, da SDK til Estimote krever dette[20]. Mer om beacons i seksjon 5.6
- iOS applikasjonen skal fungere fra og med telefoner med iOS 9.0.
    - Per 23. Januar 2017 har 96% av iPhone-brukere iOS-versjon 9.0 eller nyere[21], 96% dekningsgrad anses som tilstrekkelig av Vitensenteret, det gjør at vi kan dra nytte av mange nye funksjoner og muligheter fra versjon 9.0.
- Servere skal settes opp slik at det i fremtiden kan kjøres på en lokal server hos Vitensenteret.

1.6 Avgrensing

1.6.1 Avgrensing produkt

Det er flere Vitensentere i norske byer, vår kunde er Vitensenteret Innlandet og vi har avgrenset oppgaven til ikke å ta hensyn til de andre sentrene. Prosjektet har som mål å lage en løsning som er god på det den gjør og som kan heve besøksopplevelsen, samt at prosjektet skal ha potensiale til å bli utviklet videre - enten av eksterne aktører eller som et nytt bachelorprosjekt. Det skal i denne oppgaven ikke jobbes med utrulling og produksjonssetting av løsningen.

Det vil bli utviklet to spill - digitale implementasjoner av eksperimentstasjoner. Av tidshensyn må man velge mellom en solid struktur for videreutvikling eller utvikling av mange digitale eksperimentstasjoner. Vitensenteret føler det er viktigere med en solid plattform for videreutvikling og mulighet for å legge til

nye eksperimenter med tilhørende bilder, informasjon og hint enn flere spill. Slik systemet blir utviklet så vil videreutvikling av nye spill være relativt enkelt - man programmerer de på respektive plattformer og registrerer de som ekstrafunksjoner i webapplikasjonen.

1.6.2 Avgrensing rapport

Rapporten vil fokusere på temaer som programvaredesign og utviklingsmodell. Den vil ha mindre fokus på koden og selve programmeringen i prosjektet.

1.7 Brukergrupper

Målgruppen til dette prosjektet kan grovt deles inn i tre; brukere av applikasjonen, brukere av webapplikasjonen og lesere av denne rapporten.

Brukere av applikasjonen

Kjernemålgruppen er brukerne av mobilapplikasjonene. Dette er besøkende på Vitensenteret, og nærmere bestemt de med smarttelefon og en viss teknologisk interesse. Både barn og voksne er en del av målgruppen, de yngre barna vil kanskje bruke applikasjonen på foreldrenes telefoner mens de eldre barna har sine egne telefoner.

I ukedagene er senteret kun åpent for forhåndsbestilte grupper[22], i hovedsak fra barnehager og skoler. En stor del av brukerne er derfor barn og ungdom, noe som må tas hensyn til ved utforming av grensesnittet, brukervennlighet og vanskelighetsgrad på for eksempel quiz-spørsmål.

I helgene er senteret åpent for besøkende i alle aldersgrupper med forskjellig teknologisk kompetanse og ulik funksjonsevne. Universell utforming[23] er derfor et viktig fokusområde slik at flest mulig kan bruke applikasjonen.

De fleste besøkende er naturlig nok norsktalende, men oppdragsgiver ønsker muligheten til å utvide til flere språk ved behov. Vi internasjonaliserer derfor alle tekststrenger, og legger inn verdier for norsk og engelsk.

1.7.1 Brukere av webapplikasjonen

Webapplikasjonen kommer til å bli brukt av ansatte på Vitensenteret for å endre og legge til ny informasjon, nye eksperimenter eller nye quizspørsmål. Grunnen til at vi lager en webapplikasjon i utgangspunktet, er at brukere uten programmeringskompetanse skal kunne vedlikeholde innholdet i applikasjonen etter produksjonssetting. I tillegg bør grensesnittet være brukervennlig og intuitivt, slik at alle skal kunne gå inn og endre informasjon, i stedet for at man blir avhengig av individuelle superbrukere.

1.7.2 Lesere av rapporten

Målgruppen for denne rapporten er alle som ønsker en innsikt i hvordan prosjektet ble utført fra et utviklingsperspektiv, først og fremst veileder og sensor, men også oppdragsgiver, eventuelle utviklere som tar prosjektet videre i fremtiden eller potensielle arbeidsgivere.

1.8 Hvorfor valgte vi oppgaven

To av gruppemedlemmene har erfaring med Vitensenteret fra tidligere, både gjennom “Lær kidsa koding” [24], som er et samarbeidsprosjekt mellom NTNU i Gjøvik og Vitensenteret, og gjennom jobb direkte på senteret. Vi visste derfor på forhånd at Vitensenteret ville bli en perfekt oppdragsgiver for en bacheloroppgave, med deres fokus på vitenskap, pedagogikk og generelle vitebegjær. Da de uttrykte et ønske om å få lagd en mobilapplikasjon for besøkende, var vi kjapt ute for å vise vår interesse. Allerede på det første møtet med oppdragsgiver, der målet bare var å sondere terrenget og ingen endelige beslutninger hadde blitt tatt, fikk vi bekreftet det positive inntrykket vi hadde. Møtedeltakerne var alle veldig positive, nysgjerrige og gode idéer myldret. De kom med både kreative og konstruktive forslag, og viste generelt stor entusiasme. Etter dette møtet var det et enkelt valg for vår del.

De to mest tungtveiende argumentene for at vi valgte Vitensenteret som oppdragsgiver, og nettopp dette prosjektet, var for det første den friheten vi fikk innenfor de rammene som ble gitt. Omfanget og kompleksiteten var stor, men overkommelig, og vi hadde mange muligheter for å bli utfordret teknologisk. For det andre er det veldig motiverende å lage et produkt som en oppdragsgiver ser nytten av og som sannsynligvis kommer til å bli tatt i bruk. Selv om fokuset i en bacheloroppgave nødvendigvis må bli denne rapporten og selve utviklingsprosessen, har vi hele veien hatt et ønske om å lage et godt produkt som Vitensenteret kan videreutvikle og ta i bruk.

1.9 Forprosjekt

Høsten 2016 hadde vi et prosjekt og to mappeinnleveringer i to forskjellige emner som til sammen ble et slags forprosjekt til denne bacheloroppgaven:

- IMT3102 - Objektorientert systemutvikling[25]
- IMT3672 - Mobile Development Project[26]

Mye av tankene og konseptene utarbeidet i disse prosjektene (heretter referert til som "forprosjektet") ble brukt som basis for denne bacheloroppgaven, og mye av det faktiske arbeidet ble brukt til utarbeidelse av selve prosjektplanen (se appendix A). Pilotversjonen av Android-applikasjonen (se punkt 1.9.2) ble brukt som basis for den ferdigstilte applikasjonen - men den ble kraftig refaktorert, forbedret og utvidet. Resten av systemet er **i sin helhet** utviklet i løpet av bachelorprosjektet.

1.9.1 Objektorientert systemutvikling

Mappeinnlevering 3

I denne mappeinnleveringen skulle vi "avklare, systematisere og beskrive relevante behov, krav og forventninger oppdragsgiver og brukere har til applikasjonen," og samtidig "foreta en systematisk kartlegging av teknologiske muligheter og rammer innen (mobil)tekonologi-feltet og diskutere disse i forhold til ideer og krav".

Deler av dette arbeidet ble brukt som grunnlag for prosjektplanen. Se appendix B.1 for mappeinnlevering 3, og appendix A for prosjektplan.

Mappeinnlevering 4

I denne mappeinnleveringen skulle vi "komme opp med et velfundert forslag til programvaredesign for applikasjonen der både arkitektur, moduldesign, objektorientert design på utvalgt del og ulike grensesnittdesign (brukergrensesnitt, hardwaregrensesnitt og grensesnitt til andre applikasjoner) inngår." Det var en forventning at vi skulle foreta "caserelevante teknologivurderinger og prosjektspesifikke dybdevurderinger".

Deler av undersøkelsene gjort i mappe 4 har blitt tatt inn i bacheloroppgaven. Se appendix B.2 for mappeinnlevering 4.

1.9.2 Mobile Development Project

I dette emnet lagde vi en fungerende pilotversjon av Android-applikasjonen vår.

Funksjonalitet i pilotversjon:

- Grensesnitt modellert etter Googles *Material design*-prinsipper.
- Quiz med resultat og ledertavle.
- Et eksperiment med et mattespill som ekstrafunksjon.
- En informasjonsside med åpningstider, arrangementer og veibeskrivelse.

Se appendix B.3 for skjermbilder fra pilotapplikasjonen.

1.10 Gruppemedlemmer

Alle gruppemedlemmene studerer bachelor i programvareutvikling, og har derfor det samme utgangspunktet for faglig kunnskap.

I tillegg til det ordinære studieløpet har vi tatt ekstra fag innenfor andre fagområder. Steinar og Steffen har hatt valgfaget database- og applikasjonsdrift mens Ole André har tatt Risikostyring: Metodikk og standarder som valgfag. I database- og applikasjonsdrift lærer man drift og oppsett av databaser og applikasjoner i skyen, kunnskapen herfra er spesielt nyttig å anvende i backend-delen av systemet. I risikostyring lærer man å utarbeide risikoanalyser av større systemer på ulike plan - holdninger, atferd, informasjonssikkerhetskultur og teknisk informasjonssikkerhet. Den viktigste kunnskapen fra risikostyring som kan hjelpe oss i bachelor-oppgaven er bevisstheten rundt å tenke informasjonssikkerhet helt fra starten av prosjektet og videre som en integrert del av det kontinuerlige arbeidet.

Gjennom studiet har vi opparbeidet oss kunnskap om hele systemutviklingsprosessen - design, planlegging, implementasjon og testing av programvare. Vi har lært både native utvikling på Windows og Mac, webutvikling i HTML, CSS og JavaScript, mobilutvikling i Java for Android, algoritmer, matte, operativsystemer, informasjonssikkerhet og systemutvikling. Vi mener utdanningen har gitt oss et godt grunnlag til å takle denne oppgaven og videre arbeidsliv.

1.11 Roller

Komponentene i prosjektet var såpass adskilt at det var mulig å dele inn hele oppdraget i flere individuelle komponenter fra starten av prosjektet. Mobilapplikasjonene og webapplikasjonen måtte på et tidspunkt begynne å kommunisere med backend-serveren, men mye av funksjonaliteten kunne vi utvikle parallelt uavhengig av hverandre før man integrerte de. Prosjektet var delt inn i fire komponenter - backend i .NET Core, webapplikasjon i Polymer, Android-applikasjon og iOS-applikasjon.

Steffen Granberg var scrummaster og utvikler. Han hadde hovedansvar for backend, et RESTful API skrevet i .NET Core. Som scrummaster var han ansvarlig for å ha overordnet kontroll på at utviklingen gikk i riktig retning og for å føre agenda i sprint-møter.

Steinar Opphus var utvikler og hadde hovedansvar for frontend og Android-applikasjonen. Han var også referent i møter.

Ole André Slettum var utvikler og hadde hovedansvar for iOS-applikasjonen. Han var også hovedansvarlig for universell utforming av mobilapplikasjonene.

Prosjektdeltakerne hjalp hverandre på tvers av prosjektene selv om de hadde hvert sitt ansvarsområde. Det at man hadde hovedansvar betød at man var ansvarlig for at krav som var satt var oppfylt til avtalte tidspunkter, men alle bidro med kode til de aller fleste kodebasene. Alle prosjektdeltakerne hadde felles ansvar for både prosjektplan og rapport, samt å utføre sine oppgaver i henhold til faste rutiner og kodestandarder.

Tom Røise var veileder. Prosjektgruppen og Tom hadde fast møte hver onsdag klokken 09:00. Tom sin rolle var å veilede oss i arbeidet - krav til dokumentasjon, gjennomføring, lese gjennom oppgaven og sikre konsistens samt hjelpe til med faglige spørsmål der dette var relevant.

Bodil Hansen var produkteier og representant for Vitensenteret, hun uttrykte Vitensenterets visjon og ønsker under prosjektet. Hun hadde møte med prosjektgruppen annenhver uke på slutten av hver sprint. Det var også åpning for at andre ansatte på Vitensenteret ble med på disse møtene, de kunne også komme med innspill på lik linje som produkteier.

1.12 Om rapporten

Vi har i rapporten valgt å bruke mange engelske betegnelser og ikke oversette de, siden det er disse som er oftest brukt i fagmiljøet, og lettest gjenkjennelig.

I PDF-versjonen av denne rapporten kan man navigere fra innholdsfortegnelsen ved å trykke på

kapitlene. Kildehenvisninger, og utvalgte fremmedord og forkortelser er klikkbare. Fremmedordene har en noe mørkere blåfarge enn andre ord, disse linkene tar deg til forklaringer av ordet.

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

Rapportens struktur

Rapporten er inndelt i ti kapitler, som omhandler disse hovedemnene:

- Beskrivelse av prosess og arbeidsmetodikk
- Design og implementasjon av løsningen
- Beskrivelse av hvordan vi sikret kvaliteten underveis i arbeidet
- Oppsummering og anbefaling til videre arbeid

2 Utviklingsprosess

2.1 Utviklingsmetodikk

En systemutviklingsmetode er et rammeverk med aktiviteter som har som formål å systematisere utviklingsprosessen fra idé til fungerende system/programvare[27]. Det finnes mange forskjellige metoder som har ulike fokusområder og passer til forskjellige typer prosesser, men de har alle til felles at de forsøker å gi utviklingen en systematisk tilnærming.

I prosjektplanen (se appendix A) beskriver vi hvordan vi hadde planlagt å jobbe ved prosjektstart. Dette kapitlet vil gå grundigere inn på argumentasjonen og hvordan vi faktisk gjennomførte arbeidet.

2.2 Valg av metodikk

Karakteristika ved prosjektet som var styrende for valg av modell

- Moderat omfang i utgangspunktet, men med så mange idéer og muligheter at vi ikke rekker å gjøre alt.
- Få rammer og åpen avgrensning
- Mye funksjonalitet som blir avklart underveis
- Oppdragsgiver ønsker å være involvert i prosessen

Argumentasjon

Vitensenteret som oppdragsgiver hadde få rigide krav til sluttproduktet og ga helt fra begynnelsen av inntrykket av at de var veldig fleksible både med tanke på funksjonalitet og fremgangsmåte - noe som gjorde prosjektet mindre ideelt for de mer rigide utviklingsmetodikkene som krever mer av planleggingsfasen. Ut i fra karakteristikkene beskrevet over, bestemte vi at scrum ville være den rette smidige utviklingsmodellen for oss, i alle fall som basis.

Å ha sprinter med oppgaver tatt fra en større backlog gjør hele prosessen fleksibel. Vi kan prioritere de viktigste oppgavene først og la mindre småoppgaver som ikke er kritisk vente til vi har tid til overs.

Regelmessige *sprint review*-møter med produkteier gir oppdragsgiver muligheten til å være involvert i utviklingsprosessen, dette gir produkteier mulighet til å komme med idéer rundt ny funksjonalitet som ikke ble nevnt i starten av prosessen.

For å sikre tilstrekkelig dokumentasjon av prosessen bruker vi en del artefakter fra Rational Unified Process (RUP)[12], som for eksempel *use cases* for å kartlegge og definere funksjonalitet, system sekvensdiagram for å utdype handlingsforløpet i disse, risikoanalyser med mer.

2.3 Gjennomføring

I forprosjektet hadde vi sprintlengde på én uke, men siden hovedprosjektet var større både i arbeidsmengde og lengde, valgte vi å øke til to uker. Det ble da mindre *overhead* i form av møter og resulterende referater, men samtidig nok styringspunkter for å sikre at prosjektet var på rett kurs.

Hver sprint gikk fra tirsdag til og med mandag to uker senere.

2.3.1 Møter med oppdragsgiver

Sprint review

Det ble avtalt *sprint review*-møte på Vitensenteret i slutten av hver sprint. Produkteier (Bodil Hansen) møtte i utgangspunktet fast hver gang, mens andre ble invitert ved behov. Alle ansatte var selvfølgelig velkomne på hvert møte.

Hovedfokus for hver *sprint review* var å fortelle om fremgangen og hva vi hadde gjort i siste sprint, status, vise eventuelle demoer og få innspill til hva som skulle fokuseres på i neste sprint.

Et sånt møte ble blant annet avholdt 30. januar (se appendix J for referat). Her ble første del av møtet brukt til å diskutere oppgavene som ble ferdig utført i løpet av foregående sprint. Siden dette var tidlig i prosjektet, var det lite koding som var ferdig, og mest rapport og spesielt prosjektplan. Plan for videre arbeid ble presentert (gantt-skjema og milepæler), og deretter ble det satt av litt tid til å diskutere bruk av *beacons* kontra QR-koder for å identifisere eksperimentstasjoner med mobilapplikasjonene (argumentasjonen står i referatet).

2.3.2 Interne møter

Sprint planning

Sprint planning ble avholdt på tirsdag i begynnelsen av hver sprint, konkrete arbeidsoppgaver ble definert som *tasks/stories* i Jira og tildelt de enkelte utviklerne. Hver oppgave ble merket med fokusområde (frontend, backend, iOS, android eller rapport), og *planning poker* ble brukt for å estimere *storypoints* (se avsnitt 2.3.5 for mer om *storypoints*). Verktøyet Pointing Poker[28] ble brukt til *planning poker*.

Et sånt møte ble blant annet avholdt 14. februar (se appendix J for referat). Her gjorde vi en del vurderinger på videre rutiner, slik som skalaen på *story points*, la til manglende oppgaver i backlog, flyttet disse over i ny sprint og tok noen generelle avgjørelser i forbindelse med avgrensing og omfang.

Sprint retrospective

Én gang i måneden (for hver andre sprint), ble det avholdt *sprint retrospective*-møter. Her analyserte vi de to foregående sprintene for å finne ut hva som hadde fungert godt og som burde videreføres, og hva som hadde fungert dårlig og hva som eventuelt kunne gjøres av forbedringer. Rapporter fra Jira ble brukt som hjelpemidler i analysen, slik som *burndown*, *control* og *velocity -charts*. Se i vedlegg K og spesielt K.2 for mer detaljer.

Noe av effekten disse møtene hadde, var at vi ble betraktelig flinkere til å fordele en mengde arbeidsoppgaver som var realistisk å fullføre i løpet av en sprint, som *burndown*-grafene i figur 2 bekrefter.

Daily sprint

Vi kjørte en kort daily sprint hver dag vi jobbet sammen. Disse møtene ble brukt for å gi utviklerne, og spesielt scrum master, et innblikk i fremdrift og hva alle jobbet med.

En utvidet daily sprint ble avholdt i begynnelsen av hver uke der timelogg (se vedlegg L for ukesloggen og L.2 for sammendrag) ble presentert, og med en mer detaljert forklaring av fremdrift (og eventuelt visning av demo).

2.3.3 Møte med veileder

Statusmøte med veileder Tom Røise ble avholdt i utgangspunktet hver onsdag klokka 09:00. Her var fremdriften til denne rapporten hovedfokus, men vi kjørte også demo av systemene og hadde på et tidspunkt også omvisning på Vitensenteret (se appendix J).

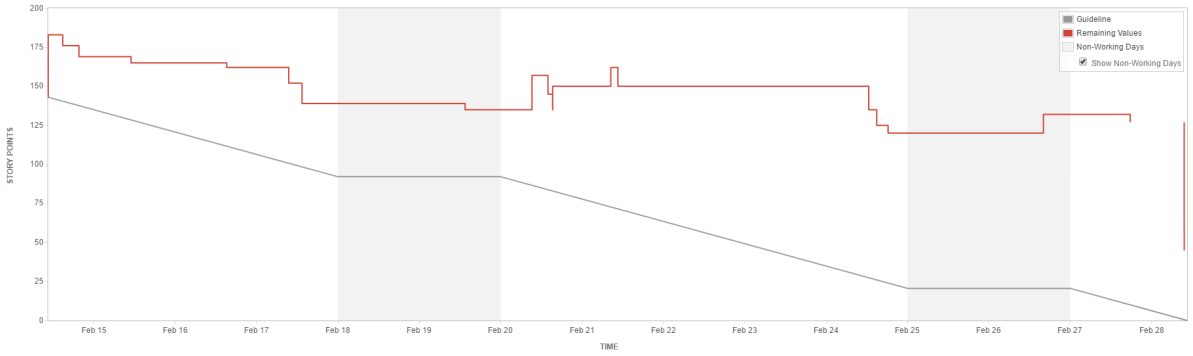
Se appendix J for eksempler på referater fra mange av møtene.

2.3.4 Scrum board

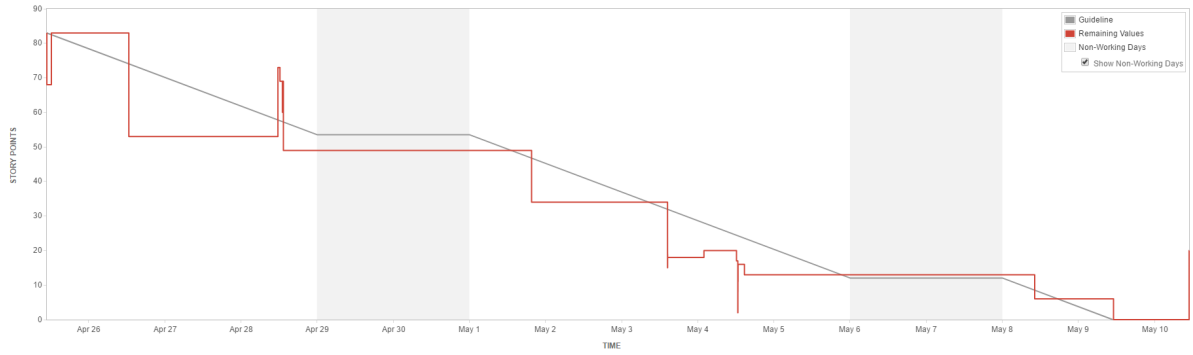
Vi brukte Jira som arbeidsstyringsverktøy og la inn alle arbeidsoppgaver som *stories/tasks* i *backlog*'en der. I hver *sprint planning* bestemte vi hvilke av oppgavene i backlog som skulle være med i neste sprint, og dette ble vårt *scrum board* (se figur 3).

Hvert kort (oppgave) ble fargekodet etter hvem som hadde hovedansvaret og ble flyttet mellom kolonnene etter hvilke status de hadde underveis i sprinten.

- **To-do**: Oppgaver som ikke er påbegynt.
- **In Progress**: Påbegynte oppgaver.



(a) Sprint 2



(b) Sprint 7

Figur 2: Sammenligning av burndown i begynnelsen og slutten av prosjektet (fra Jira)

- **Ready for QA (Quality Assurance)**: Oppgaver som er ferdige, men trenger ekstra kvalitetssikring av andre.
- **Done**: Ferdige oppgaver.

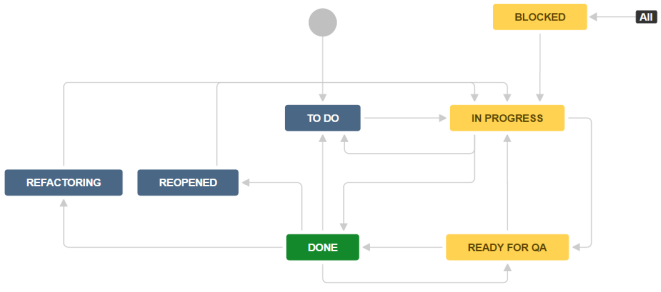
En oppgave kunne også ha tre andre statuser i spesielle tilfeller, *refactoring*, *reopened* og *blocked*.

Refactoring ble brukt når en kodeoppgave som var ferdig ble åpnet på nytt for å refaktoreres (det vil si at koden ble forbedret). *Reopened* ble brukt når oppgaver ikke relatert til koding ble åpnet på nytt for å forbedres. *Blocked* var oppgaver som ikke var ferdige eller har feil, som var så kritiske at de blokkerte andre deler av systemet. Se figur 4 for diagram som viser arbeidsflyten for en oppgave i Jira.

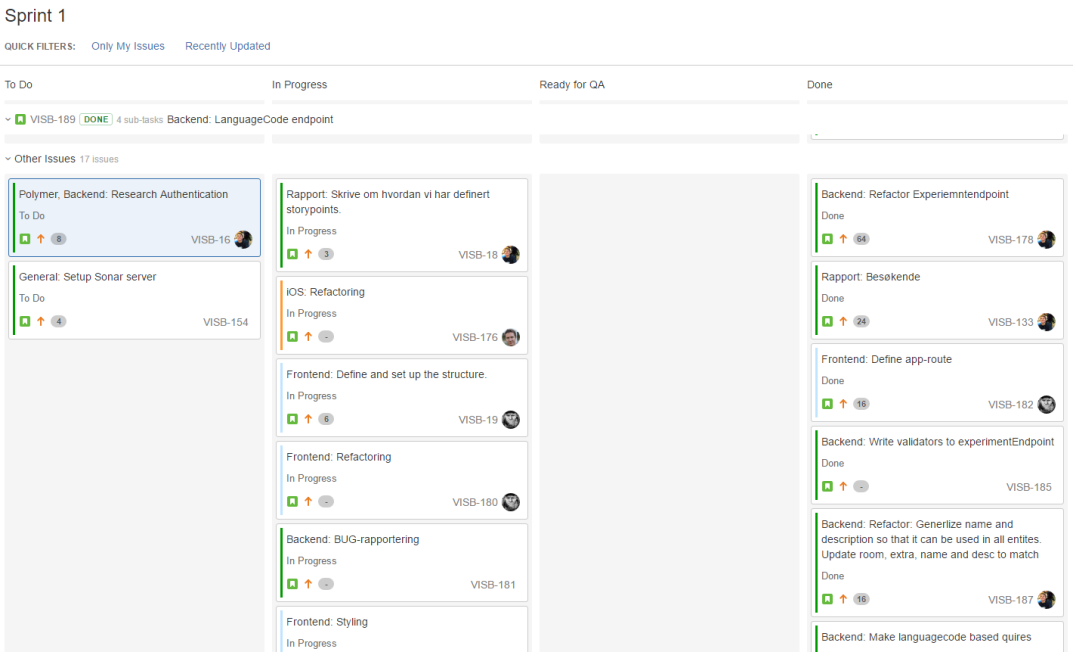
2.3.5 Story points

En viktig del av scrum er estimering av arbeidsoppgaver - eller *tasks* som vi kaller dem. I tidligere prosjekter vi har gjennomført som gruppe med metodikken har vi estimert tid ved hjelp av tidsenheter. Det vil si at vi så på en task, og så sa hvert gruppemedlem hvor lang tid de trodde denne oppgaven ville ta, og ble enige om hva tidsestimatet skulle være.

Det vi merket oss var at vi ofte var veldig uenige om tidsbruk på flertallet av oppgavene, og tidsestimeringen vi endte opp med var ofte vag eller feil.



Figur 4: Workflow i Jira (hentet fra Jira)



Figur 3: Scrum board i Jira (skjermbilde fra Jira)

Derfor ville vi denne runden se på en
annen tilnærming til estimering.

Story points[29] en en rangering som beskriver forskjellige aspekter ved en oppgave. I motsetning til tidsestimering så ser story points på flere faktorer som spiller inn i oppgaven, herunder kompleksiteten, forventet tidsbruk og risikoer. Det som er viktig å merke seg er at story points ikke er en annen måte å beskrive tid på. Det vil si at *storypoints* ikke skal eller kan konverteres til tidsenheter, men heller beskrive hvor kompleks en oppgave er sammenlignet med en annen.

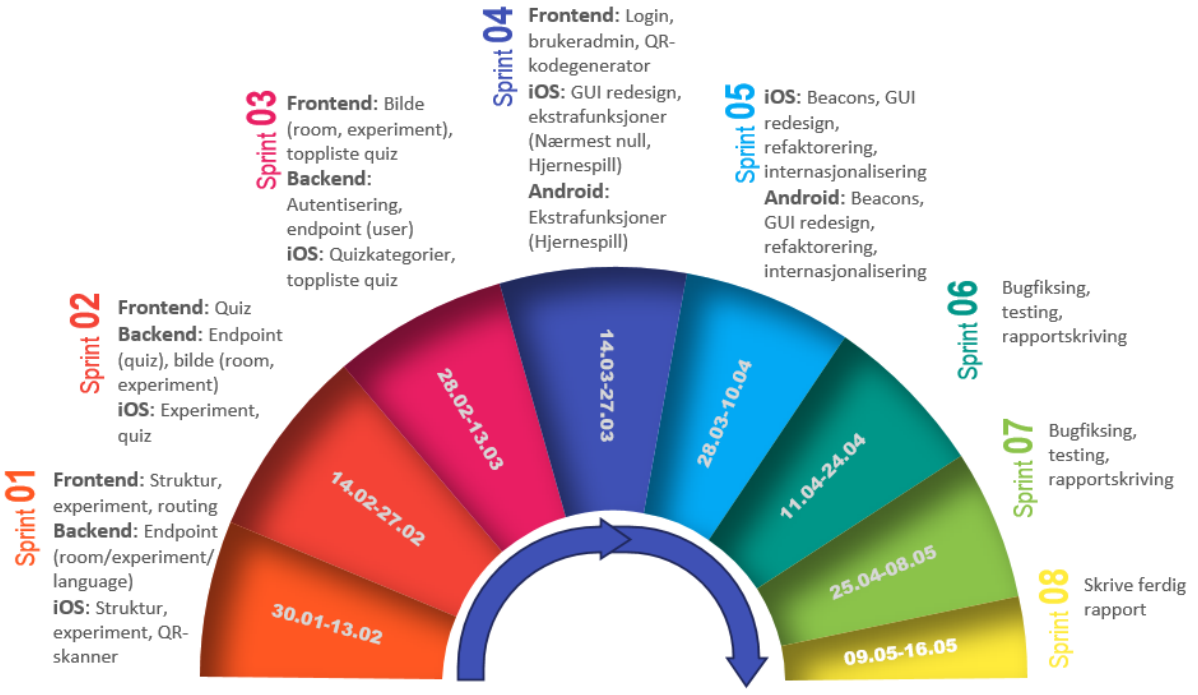
For å bruke story points måtte vi lage en skala, vi valgte å følge en vanlig tilnærming ved å bruke en adapsjon av fibonacci-sekvensen[30] $1, 2, 3, 5, 8, 13, 21, 34, 45$. Vi valgte å omgjøre slutten på rekken slik at vår skala ble: $1, 2, 3, 5, 8, 13, 21, 40, 100$. Det viktige her er at en task med verdi 2, skal være dobbelt så kompleks som en task med verdi 1, og en task med 3 skal være dobbelt så kompleks som en task med 2.

I et utviklingsteam er det vanlig med utviklere som jobber i forskjellig tempo. Dette kan enten være basert på erfaring eller at området utvikleren jobber på er mer komplisert. Derfor er det viktig å ha en tilnærming som ikke sier noe om hvor flink utvikleren er. En dyktig erfaren utvikler vil kunne utvikle i et raskere tempo enn en fersk utvikler gitt samme oppgave. Det vil her være vanskelig for disse to utviklerne å bli enige om en tidsestimering for oppgaven.

Det er her story points kommer inn, de beskriver ikke tiden, men kompleksiteten til en oppgave, noe som det er mye lettere å bli enige om. For de fleste har et forhold til hvor kompleks en oppgave er i forhold til en annen.

Det blir så en oppgave for teamet og definere hvor mange storypoints hver utvikler har kapasitet til å gjennomføre i en sprint, dette kan og bør varierer slik at det gir et realistisk bilde av teamsammensetningen.

Vi er veldig fornøyde med hvordan estimeringen fungerte utover i bachelorprosjektet. Vi klarte å definere *storypoints* for hver task og det ble bedre for hver utvikler å få oppgaver tilpasset sin egen hastighet.



Figur 5: Kodeoppgaver i hver sprint (laget med MS Visio)

2.4 Sprintoversikt

Som beskrevet over (seksjon 2.2) valgte vi å jobbe etter scrum-metoden. Vi har fulgt denne smidige metodikken under hele prosjektet, og spesielt med alle oppgaver relatert til koding. Vi hadde satt oss noen grove mål med tidsfrister i prosjektplanen (appendix A), men backloggen med konkrete oppgaver ble fylt opp underveis slik at vi enkelt kunne tilpasse oss etter innspill fra både produkteier og veileder og hadde rom for å gjøre andre endringer underveis basert på egne erfaringer og idéer.

I arbeidet med å skrive denne rapporten har vi derimot jobbet mer sekvensielt. Her har disposisjon blitt utarbeidet i forkant, og kapitler eller delkapitler fordelt underveis. Vi har prøvd å sette opp en struktur som er logisk i forhold til det vi har ønsket å formidle, og det er derfor ikke samsvar mellom rekkefølgen i rapporten og hvordan applikasjonene er utviklet.

2.4.1 Kort oppsummering av arbeidet i hver sprint

Prosjektplan (12. jan - 31. jan)

Prosjektplan ble ferdigstilt og research utført på våre respektive ansvarsområder. Separate *repositories* ble satt opp til hvert område og inkludert en grunnleggende struktur, samt Swagger (les mer om Swagger i seksjon 7.1.2).

Sprint 01 (31. jan - 14. feb)

Endpoint for *language*, *room* og *experiment* ble laget på backend, samt støtte for å få språkspesifikke responser på kall. På *frontend* ble det laget en tidlig versjon av sidene for å administrere eksperimenter og rom, i tillegg til implementering av *routing*. På iOS ble QR-skanner implementert, samt utkast til eksperiment- og informasjonssiden. Første utkast til introduksjonen i denne rapporten (kapittel 1) ble skrevet, i tillegg til at vi begynte på kapitlet om kravspesifikasjon (kapittel 3).

Steffen holdt en workshop om .Net Core med en innføring i C#.

Sprint 02 (14. feb - 28. feb)

På *backend* ble det opprettet *endpoint* for quiz, *quiz category*, *-question* og *-leaderboard*, og mulighet for opplasting av bilder for rom og eksperiment. På *frontend* ble det opprettet sider for å administrere disse *endpoint*'ene. Logikken bak eksperimentsidene ble ferdig på iOS og quiz påbegynt.

Sprint 03 (28. feb - 14. mar)

Det ble gjort research for autentisering på både *back-* og *frontend*. *Endpoint* for autentisering og brukere ble laget, i tillegg til støtte for sletting av bilder og quiz-kategorier og -spørsmål. På *frontend* ble det etter ønske fra produkteier implementert en teksteditor (TinyMCE[31]) med støtte for formatering på beskrivelsen av et eksperiment, samt toppliste til quiz og mulighet for å laste opp bilder til rom og eksperiment. På iOS ble det lagt til mulighet for å velge kategori på quiz, og se toppliste. Kapittel 4 (teknologier) ble skrevet.

Ole André holdt en workshop om iOS-utvikling i Swift.

Sprint 04 (14. mar - 28. mar)

Det ble avholdt en GUI-workshop for skissering av hvordan grensesnittet på mobilapplikasjonene skulle se ut (skissene er lagt ved i appendix I).

Login og brukeradministrasjon ble implementert i *frontend*, i tillegg til en QR-kodegenerator. Mange små endringer ble gjort etter brukertest av *frontend* med produkteier. På iOS ble brukergrensesnittet redesignet etter retningslinjene vi bestemte i GUI workshop'en. I tillegg ble spillene *Nærmest null* og *Hjernespill* implementert. På Android ble hele strukturen refaktorert etter design patternet *Model-View-Presenter* (se seksjon 5.5.2 for mer detaljer). *Hjernespill*et ble også implementert.

Sprint 05 (28. mar - 13. apr)

Beacons ble implementert på både iOS og Android. I tillegg ble det gjort mye refaktorering, redesign og internasjonalisering på begge plattformene. Feil oppdaget med *Linter* og *Sonar* (les mer om disse verktøyene i kapittel 8) ble fikset. Hovedfunksjonalitet i både *backend*, *frontend*, iOS og Android ferdig!

Sprint 06 (10. apr - 24. apr)

Hovedfokuset ble nå satt til rapport, samtidig som det ble fikset bugs og refaktorert kode. Designdelen (kapittel 5) ferdigstilles og gjennomleses nøye.

Sprint 07 (24. apr - 08. mai)

Rapportskriving fortsatte. I tillegg ble det kjørt en brukertest og en presentasjon til arbeidsgiver. Vi hadde møter hvor vi jevnlig gikk igjennom rapporten og laget lister både internt og fra veiledning om ting vi ønsket å forbedre. Noen feil ble oppdaget under testene og fikset. Se kapittel 9 for mer om testingen.

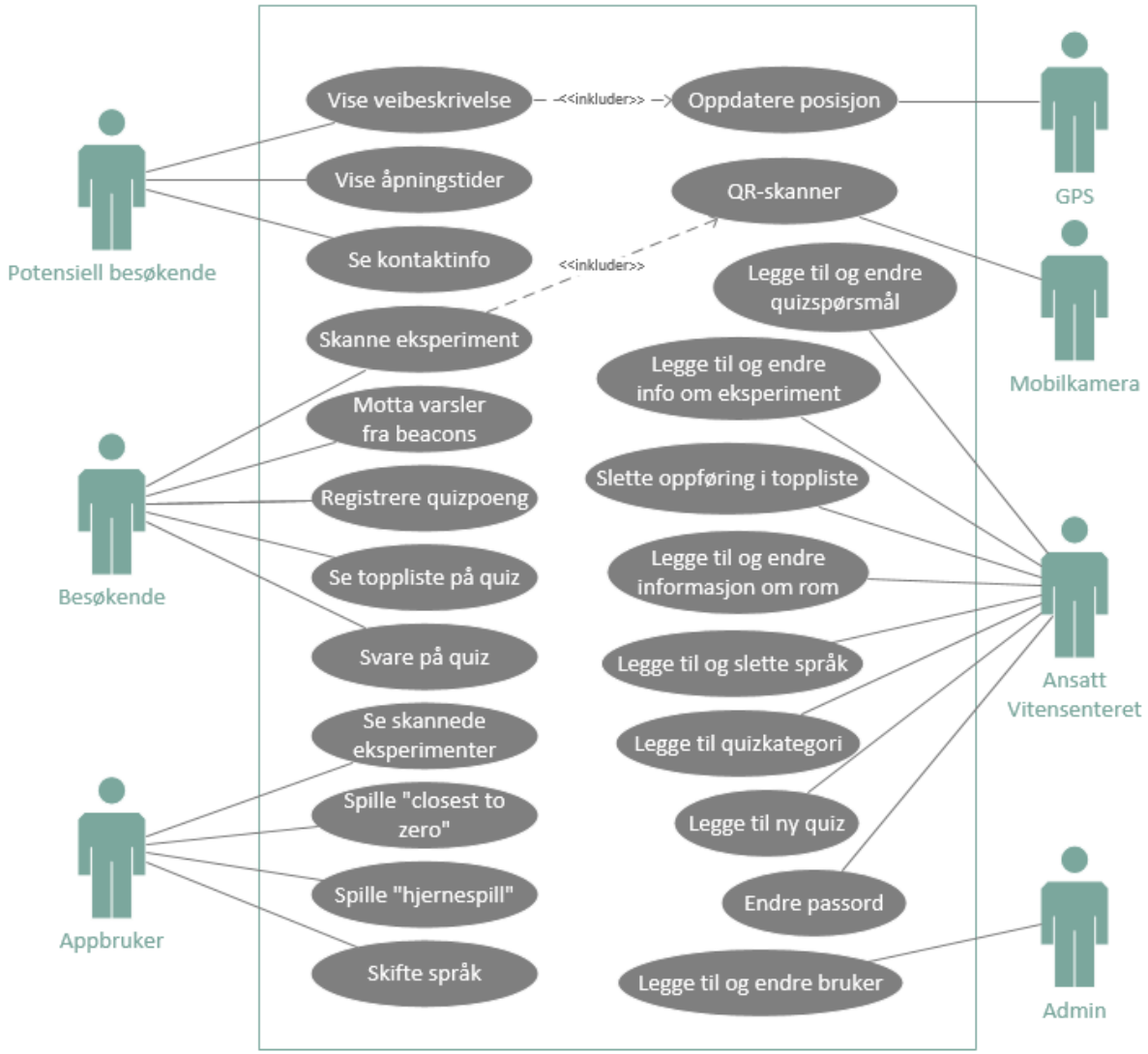
Sprint 08 (08. mai - 16. mai)

Rapporten ble ferdigstilt, og sendt til oppdragsgiver for en gjennomlesning og tilbakemelding. Siste veiledning på rapporten ble gjennomført sammen med veileder. Hele dager gikk med til lesing og retting av rapporten. Rapporten ble levert.

3 Kravspesifikasjon

3.1 Use Cases

Vi ønsket å bruke use case og use case-diagram fordi vi fra tidligere prosjekter har god erfaring med nytten av en slik strukturert oversikt over de forskjellige oppgavene brukerne av applikasjonene kan utføre. Det er mye lettere å se sammenhengen og det totale systemet når det settes opp på denne måten. Kompleksiteten er forskjellig på de ulike casene, og spesielt interaksjonen brukeren skal gjøre når den utfører en quiz kan være utfordrende. Dette blant annet fordi den har flere alternative scenarier, derfor valgte vi å lage en utvidet use case som beskriver denne mer i detalj.



Figur 6: Use case diagram (laget i MS Visio)

3.1.1 Noen high-level use case-beskrivelser

Se vedlegg F for beskrivelser av alle *use case*'r. Her er kun et utdrag.

Use case navn: Vis veibeskrivelse
Aktører: Potensielle besøkende
Mål: Se veibeskrivelse til Vitensenteret
Beskrivelse: Potensiell besøkende trykker på knapp som sender han til telefonens innebygde kart-applikasjon. Senterets koordinater blir sendt med som destinasjon.

Use case navn: Skanne eksperiment
Aktører: Besøkende
Mål: Skanne og få tilgang til eksperimentstasjonens side
Beskrivelse: Besøkende bruker telefonen sin til å skanne QR-koder som han finner på eksperimentet. Applikasjonen sender brukeren videre til en aktivitet som viser mer informasjon om eksperimentet - info, hint og eventuell ekstrafunksjonalitet

Use case navn: Motta varsler fra beacons
Aktører: Besøkende
Mål: Brukere er inne på senteret og mottar informasjon som pushes fra beacons-enheter
Beskrivelse: Brukeren har bluetooth aktivert og mottar informasjon fra beacons - for eksempel "Velkommen til senteret" og "Nå er det planetarievisning"

Use case navn: Svare på quiz
Aktører: Besøkende og bruker av mobilapplikasjon
Mål: Brukeren kan se toppliste på quizen han har valgt eller nettopp har gjennomørt
Beskrivelse: Brukeren velger kategori og quiz og blir sendt til siden hvor han kan starte quizen, her ser han også quizens toppliste. Når brukeren er ferdig med en quiz er det en knapp hvor han kan trykke seg videre til topplisten

Use case navn: Spille "hjernespill"
Aktører: Bruker av mobilapplikasjon
Mål: Aksessere og spille spill
Beskrivelse: Brukeren får tilgang til hjernespill ved å skanne det på senteret. "Hjernespill" har fire knapper med ulike farger, disse knappene lyser opp i en bestemt sekvens som brukeren må gjenta. Hver gang brukeren gjentar sekvensen riktig startes en ny sekvens hvor antall deler i sekvensen øker med en hver gang.

Use case navn: Legge til og endre informasjon om eksperimentstasjon
Aktører: Ansatt på Vitensenteret
Mål: Legge til ny stasjon i mobilapplikasjonen
Beskrivelse: Legge til ny stasjon som kan bli skannet ved å legge til en QR-kode på eksperimentet. Brukeren kan legge til informasjon som navn, komplementær informasjon, video, bilder og spørsmål

3.1.2 Use case (utvidet)

Use case navn: Svare på quiz
Omfang: Mobilapplikasjon
Primær aktør: Besøkende
Interessenter:

- Besøkende: Besøkende vil svare på quiz, lagre sin poengsum og se toppliste
- Vitensenteret: Vil ha oppdateringer av topplister

Mål: Starte og fullføre quiz
Beskrivelse: Appbruker starter quiz og svarer på spørsmål. Hvert spørsmål må besvares innen 10 sekunder, hvis ikke går quizen videre til neste spørsmål og brukeren får ingen poeng på gitt spørsmål. Når quizen er ferdig vises brukeren en side der han ser sin score og kan se toppliste for quizen, registrere score, spille ny quiz eller returnere hjem til applikasjonens hovedside.
Type: Essensiell
Pre-betingelse: Har valgt quiz han vil gjøre og har internett-tilkobling slik at han kan laste ned quizen
Post-betingelse: Brukeren har besvart quizen
Grunnleggende programflyt:

1. Brukeren trykker Start quizknapp
2. Quizen starter og henter ut alle spørsmål som er knyttet til den valgte quizen
3. Spørsmål blir gitt til brukeren med fire alternativer
4. Brukeren besvarer spørsmålene
5. Brukeren repeterer punkt 3-5 helt til alle spørsmål er besvart
6. Brukeren sendes til en side der han ser sin score og kan se toppliste, laste opp score, starte ny quiz eller returnere til applikasjonens hovedside
   Alternative scenario:
   Brukeren trykker se toppliste

   1. Brukeren vises toppliste for respektiv quiz
   2. Hvis brukeren er er på topp 10 utheves hans score, hvis ikke vises topp 10 og brukerens plassering

   Bruker trykker ”Last opp score”

   1. Brukeren laster opp score med et tidligere ubrukt navn
   2. Score lastes opp
      ELLER
   3. Brukeren laster opp score med et tidligere brukt navn som ikke er hans
   4. Brukeren får beskjed om at navnet ikke kan brukes og blir bedt om å prøve et annet navn

   Bruker trykker ”Spill ny quiz”

   1. Brukeren blir sendt tilbake til kategorivalg for quiz

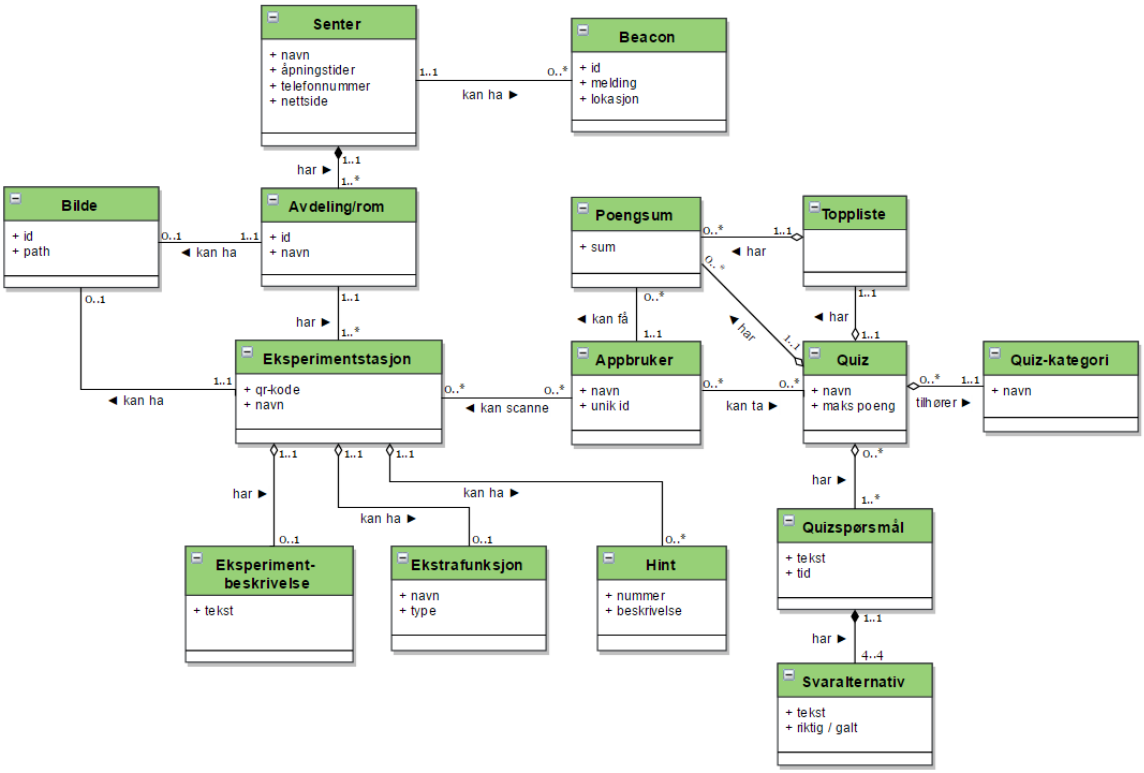
   Brukeren trykker hjem

   1. Brukeren sendes hjem til hovedsiden

3.2 Product backlog

Product backlog ble fortløpende oppdatert i JIRA. Vi laget tasker for alle oppgaver som skulle utføres, enten dette gjaldt koding, rapportskriving eller administrativt. Dette ga oss god oversikt over alle oppgavene som til en hver tid måtte gjøres. Vi endte opp med totalt 589 tasker. Et utdrag av disse finnes i Appendix 1.

3.3 Domenemodell



Figur 7: Domenemodell (laget i draw.io)

3.4 Operasjonelle krav

Sammen med oppdragsgiver, har vi gjennom prosjektperioden i fellesskap kommet frem til operasjonelle krav som Vitensenteret ønsker at et fullverdig, produksjonssatt system skal kunne håndtere.

Siden oppdragsgiver ikke besitter den samme tekniske innsikten, har vi stilt spørsmål og kommet med forskjellige scenarier som har gjort at vi i fellesskap har kommet frem til disse kravene.

- Systemet bør takle 500 samtidige brukere.
    - Gjennomsnittlig besøk på helg varierer mellom 20-50, men på spesielle arrangementer kan senteret oppleve opp mot 300 besøkende. Hvis vi også tenker oss at rundt det dobbelte bruker mobilapplikasjonen hjemme samtidig, bør systemet takle dette.
- Svartid på nettverkskall bør ligge på mellom 1-3 sekunder.
- Det er viktig for Vitensenteret at oppetiden på systemet er høy og vi har kommet frem til at 99.5% oppetid er ønskelig. Dette er ekstra viktig i helger og ferier, da senteret er besøkt av brukere av applikasjonen.

3.5 Sikkerhetskrav

Ettersom vi underveis i prosessen har tatt valg som påvirker sikkerhetskravene har disse kravene forandret seg underveis. Siden Vitensenteret har liten kompetanse rundt IT og sikkerhet har dette vært en prosess der vi har diskutert diverse problemstillinger, reflektert i kravene under.

Vi valgte å gå bort i fra brukerkonto for de besøkende for å unngå at barn skulle behøve å opprette en brukerkonto med e-postadresse og passord. Vi lagrer ingen personopplysninger på mobilapplikasjonen, dette senker kravene noe. Derimot skal de ansatte ha brukere i vårt system, denne informasjonen må lagres på en sikker måte. I samarbeid med Vitensenteret har vi kommet frem til følgende sikkerhetskrav:

1. Det skal ikke være mulig for uvedkommende som ikke er ansatt på Vitensenteret å redigere informasjonen som ligger i databasen. Herunder gjennom webapplikasjonen eller med direkte tilgang til API'et.
2. Brukerkontoene som Vitensenteret bruker for å få tilgang til systemet skal være lagret på en forsvarlig måte.
3. Det skal være tidsbestemt innlogging, noe som vil si at selv med tilgang til en gyldig *access token* skal ikke noen ha tilgang til evig tid.
4. Applikasjonen skal støtte HTTPS gjennom SSL/TLS tilkobling, selvom det ikke er krav om at dette settes opp ferdig i løpet av bacheloroppgaven.
5. Applikasjonen skal ikke lagre informasjon som kan identifisere bruk, bruksmønstre eller lignende.

Forklaringer på hvordan disse sikkerhetskravene er imøtekommet i implementasjonen står utover i rapporten. Det er spesielt punktene beskrevet under som direkte løser noen av disse kravene.

- For å nå krav 1, 2 og 3 beskrevet over, ble det implementert autentisering på alle *endpoint* som la til eller endret informasjon i databasen. Hvordan dette ble implementert og hvordan sikkerheten ble ivaretatt står beskrevet i seksjon 5.3.1.
- For å nå krav 3, har vi tilrettelagt for muligheten til å sette opp HTTPS. Dette har vi beskrevet mer i seksjon 10 spesielt subseksjon 10.1.1.

4 Teknologier

4.1 Polymer

4.1.1 Valg av JavaScript-rammeverk til frontendutvikling

Sammen med HTML og CSS, er JavaScript kjerneteknologien som blir brukt til å lage web-applikasjoner [32]. Der HTML og CSS brukes til den overflatiske presentasjonen, trengs JavaScript for mer komplisert funksjonalitet og interaksjon.

De siste årene har det kommet mange JavaScript-rammeverk som gjør det enklere å bygge interaktive web-applikasjoner, og det teknologiske landskapet er i stadig utvikling. Dette, i kombinasjon med at det er mange forskjellige (og sterke) meninger om hva som er "best" og verdt å satse på, kan gjøre det vanskelig å orientere seg og gjøre et informativt valg.

Det vi la vekt på i valget av rammeverk for å lage frontend-applikasjonen, var først og fremst at det skulle være raskt å ta i bruk og samtidig ha fokus på brukervennlighet og oversiktlighet. Googles Polymer hadde vi god kjennskap til fra før, siden vi hadde lært dette gjennom et tidligere emnet (WWW-Teknologi). I dette emnet fikk vi god kunnskap om Polymer, og vi likte rammeverket godt. Det er også tett knyttet opp til Googles design-prinsipper, og gjør det lett å følge *Material design*-retningslinjene deres for å lage oversiktlige og brukervennlige applikasjoner. Denne erfaringen gjorde at Polymer var det naturlige valget for å lage frontend-applikasjonen.

4.1.2 Om Polymer

Polymer er i hovedsak utviklet av Google [33] med bidragsytere via GitHub. Ved hjelp av rammeverket kan man lage egne elementer, som kan brukes på samme måte som de vanlige HTML-tagsene som <body>, <header> og <footer>. Dette vil si at man på en måte utvider HTML-språket med egne tags, som lett kan gjenbrukes.

Polymers viktigste funksjoner:

- Enkelt å lage egne elementer
- Både én- og toveis-binding av variabler (*properties*)
- *Computed properties* kun med HTML (uten bruk av JavaScript)
- Både kondisjonelle og repeterende *templates*
- Stort bibliotek med ferdige elementer[34]

Det står mer om hvordan vi har brukt Polymer i prosjektet i seksjon 5.2.

4.2 .NET Core

Siden vi har valgt å utvikle native på iOS og Android betyr dette at vi får mange teknologier og språk å sette oss inn i og forholde oss til. Derfor var det i backendløsningen viktig for oss at vi valgte teknologi som var godt dokumentert og relativt lett å komme igang med, siden backend-løsningen helst skulle være litt foran frontend og mobile slik at de underveis i prosessen kunne kode opp mot API'et direkte - ønsket vi å begrense oppstartstiden for å komme igang med utviklingen. Vitensenteret hadde ingen krav til backend-teknologi, annet enn at det skal kunne kjøre på deres servere i fremtiden.

En av gruppemedlemmene hadde laget slike RESTapier før, og hadde god erfaring med .NET rammeverket til Microsoft. I vårt prosjekt ønsket vi også å kunne kjøre løsningen på unix-baserte systemer, fordi to av tre utviklere har Mac OS som primær OS, og vi har god kjennskap til linuxservere gjennom flere fag på studiet. Derfor ble Microsofts nylig lanserte .NET Core en mulig løsning. Dette er et *open*

source kryssplattformrammeverk som har som mål å nå ut til et bredere publikum enn det eldre .NET rammeverket.

.NET Core ble lansert i versjon 1.0 juni 2016, med en påfølgende stor oppdatering november 2016 til versjon 1.1. .NET Core er en fullstendig omskriving av det kjente .NET rammeverket til Microsoft, med fokus på kryssplattformmuligheter. Tidligere løsninger som .NET MVC 5 har begrenset utviklere til Microsoft sine systemer: Windows og Azure, men Core kan kjøres på Windows, Mac OS og Linux.

I forprosjektet diskuterte vi en rekke ulike teknologier vi kunne bruke til å skrive APIet. Node.js[35] var lenge et alternativ, det samme var Java JAX-RS[36]. En av fordelene ved eventuelt å velge Node.js var at man kunne utvikle frontend og backend i samme språk(JavaScript). Ettersom vi ønsket å utvikle webapplikasjonen i Polymer som har en litt annen tilnærming enn andre JavaScript-rammeverk så vi ikke fordelene ved å utvikle i samme språk. Vi valgte Microsoft .NET Core fordi vi satt med mye erfaring og kunnskap rundt dette. Ettersom resten av løsningen var mer ukjent og vi ønsket å bruke mest tid på de mer komplekse løsningene rundt mobil, var argumentet med at vi kunne komme raskt i gang det som til slutt veide tyngst

Hvordan vi har valgt å implementere API'et i .NET Core vil vi komme tilbake til i seksjon 5.3.

4.3 Entity Framwork Core (EF Core)

Entity Framework Core eller EF Core er en *Object-Relation Mapper*[10] som muliggjør utviklere å kommunisere med databasen ved hjelp av .NET objekter[37]. Dette sparer igjen utviklere for mye tid, da utviklere ikke trenger å skrive SQL-setninger eller manuelle spørringer til databasen. ORMet tar seg av alt fra kommunikasjon til databasen, utforme spørringer og mappe dataene til objekter i koden. Entity Framework Core er Microsofts eget ORM spesiallaget til .NET Core, derfor var dette et enkelt valg for oss.

Fordelene ved å bruke et ORM er at det forenkler prosessen med å sette opp en database[38]. ORM er heller ikke trivielt å lære hvis man er vant til å skrive egne spørringer og behandle databasen manuelt. Det er mange prosesser som blir automatisert, noe som kan føre til at utviklerne ikke forstår helt hva som skjer i bakgrunnen.

Fordelene veier som regel opp for ulempene ved å bruke et ORM. Selvom det kan være utfordrende å både lære og sette opp, vil denne tiden kunne spares inn når det skal brukes senere. Systemet gjør også at man enkelt kan endre datamodellen uten å røre koden som behandler modellen, noe som gjør det lettere å endre databaseskjemaene utover i prosjektet.

Hvordan vi har benyttet Entity Framework inn i *backend*-løsningen vår har vi skrevet mer om i seksjon 5.4.

4.4 Swift

Swift er Apple sitt egetutviklede programmeringsspråk som brukes til å bygge applikasjoner og programmer på macOS, iOS, watchOS, tvOS og Linux. Swift er et multi-paradigme språk - protokoll-orientert(interfaces), objekt-orientert, blokk strukturert og imperativt med innslag av funksjonell programmering. iOS-applikasjonen vår er skrevet i Swift.

I planleggingsfasen måtte vi velge mellom å skrive en kryssplattformapplikasjon i et rammeverk som React Native eller native applikasjoner. Vi valgte native applikasjoner, viser her til seksjon 4.5 for diskusjon rundt dette valget. I utvikling av native applikasjoner for iOS har man flere alternativer for programmeringsspråk. Man kan skrive deler av applikasjon i C/C++, men for å jobbe med iOS-grensesnittet må man bruke Swift eller Objective C. Vi valgte å bruke Swift, av flere årsaker: Vi måtte uansett lære oss et nytt språk, ingen av prosjektdeltakerne hadde erfaring med hverken Swift eller Objective C. Swift er både lettere å lese og skrive, og krever mindre kode enn Objective-C for sammenliknbare operasjoner. Swift er også raskere enn Objective-C. [39]

Det står mer om iOS-applikasjonen og hvordan den er oppbygd i seksjon 5.4

For å jobbe med brukergrensesnitt i iOS-applikasjoner må man bruke enten Swift eller Objective-C, men nettverkslogikk og annen kjernelogikk kan man gjerne skrives i språk som C++ og Go. Vi valgte å kun bruke Swift, hovedsaklig for å minimere antall kodebaser, språk og teknologier.

4.5 Technical memo: Kryssplattform eller native?

Faktorer	Skal mobilapplikasjonene utvikles som to separate *native*-applikasjoner eller som én kryssplattformapplikasjon?

Diskusjon	• Hva vi kunne fra før
	  ◦ Høsten 2016 hadde vi to fag i Java, det ene som forprosjekt for bachelorprosjektet. Videre hadde vi ingen kunnskap om hverken React Native eller Swift, vi måte uansett lære oss en ny plattform. Vi hadde altså likt utgangspunkt for de to plattformene.
	• Videre støtte fra Apple og Google
	  ◦ React Native er ikke offisielt støttet av Apple og Google, dette gjør at ny funksjonalitet i iOS og Android kan risikere å ikke støtte React Native funksjoner. [40].
	• Sensorer
	  ◦ En av hovedgrunnene til at vi valgte å utvikle native var fordi vi planla å bruke flere mobilsensorer i applikasjonen. Det er lettere å bruke telefonens sensorer når man utvikler native. [41] I de ferdige mobilapplikasjonene ble det ikke brukt noen mobilsensorer, men mulighetene er mange for at man kan gjøre det ved videreutvikling.
	• Design
	  ◦ Både Android og iOS har sine egne retningslinjer for design. Google har sitt Material design[42] mens Apple har sine *Human Interface Guidelines* for design[43]. Prinsippene er basert på forskning og søker også å skape en standard mellom applikasjoner på de ulike plattformene som gjør at brukerne gjenkjenner applikasjonene og gjør dem lettere å bruke ettersom de er vant med navigasjon o.l.
	• Ytelse
	  ◦ *Native*-applikasjoner er raskere enn kryssplattform[44]. Vi ønsket en rask og god opplevelse for brukeren.

Løsning	• Basert på ovenstående punkter valgte vi i samarbeid med Vitensenteret å utvikle mobilapplikasjonene native i Swift og Java. • Vi ble enige med Vitensenteret om å ikke utvikle noen applikasjon for Windows og andre OS pga deres lave markedsandel - i Norge har Android og iOS en markedsandel på 99.6%[45]. • *Backend*en var den samme for begge applikasjonene, det var kun *frontend*en som trenger to kodebaser. • Hvis Vitensenteret på et senere tidspunkt ønsker å bruke sensorer i mobilen i forbindelse med eksperimenter så er dette mer praktisk når man har valgt å utvikle applikasjonen som en native applikasjon. Mulighetene er mange for sensorbruk integrert med eksperimenter på Vitesenteret.

4.6 Technical memo: Beacons eller QR-koder for scanning av eksprimenter?

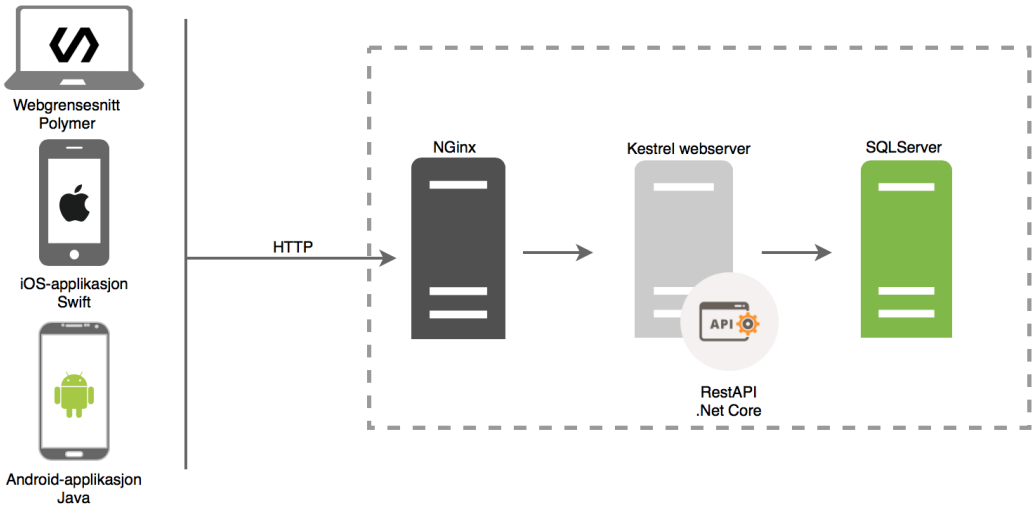
Faktorer	I applikasjonen skal det kunne skannes et eksperiment og så skal applikasjonen vise hvilken eksperimentstasjon brukeren befinner seg på. Vi vurderte to mulige løsninger for dette. QR-koder og beacons.
Diskusjon	Beacons er enda en relativt ny teknologi, og starter først nå å bli brukt kommersielt. Derimot finnes det noen produsenter som har utviklet gode og veldokumenterte rammeverk og løsninger rundt denne teknologien, Estimote[46] er blant disse. Beacons er små enheter som ved hjelp av bluetooth-teknologi sender ut "her er jeg"meldinger[47]. Mobilen kan så lytte etter slike typen meldinger, og gi en respons hvis den kommer innenfor rekkevidde av en beacon. Fordeler med beacons: • Det er en moderne måte å løse nærhetsteknologi. • Brukeren trenger ikke utføre en aktiv handling, men vil motta signaler når den går forbi en eksperimentstasjon. • Beacons har stor rekkevidde, og kan oppdages fra flere titalls meter. • Flere store aktører er tungt inne i denne teknologien, og det vil komme mange nye muligheter i tiden som kommer. Ulemper med beacons: • Det er utfordringer rundt det å ha flere beacons tett ved hverandre. • Dyrere enn andre alternativer i innkjøp. • Ansatte ved Vitensenteret må passe på at de ikke stjeles, og at de fungerer. Dette innebærer å skifte batteri. Det andre alternativet var QR-koder. QR-kode er en type strekkode som kan brukes til å kode alt fra lenker til tekst. Brukeren skanner informasjonen ved hjelp av et mobilkamera. Fordeler med QR-kode: • Det er en kjent teknologi for mange, og trenger minimalt med forklaring. • Det er ingen kostnader tilknyttet å lage de, og de trenger ikke å byttes ut. • De kan ikke fjernes eller stjeles. Ulemper med QR-kode: • Brukeren må utføre en aktiv handling for å motta informasjonen. • Det kreves en applikasjon med en QR-scanner implementert.
Løsning	Vitensenteret Innlandet har områder som Origo - et matteområde - som har mange eksperimenter i veldig nærhet til hverandre. Dette gjør at det vil være upraktisk og vanskelig å bestemme hvilke beacon en mobil er nærmest, når avstanden kanskje bare er noen få centimeter. Innkjøp av så mange beacon vil være en kostnad Vitensenteret ikke vil ta før de ser at applikasjonen er noe som blir brukt. Det vil også være driftskostnader tilknyttet beacons herunder batteri, plassering og erstatning av ødelagte/stjålne. QR-koder er en teknologi som er kjent for de fleste, og det kan enkelt legges til på informasjonsplakatene som Vitensenteret allerede benytter per eksperiment. Selv om vi så at beacons er en mer moderne og kul teknologi, så var det verken brukervennlig eller kostnadseffektiv å bruke som teknologi for skanning av eksperimenter. Til slutt endte vi opp med å bruke beacons på en annen måte i mobilapplikasjoen, nemlig som verktøy for stedsbasert informasjon. Vitensenteret støttet oss i vår vurdering og valg av scanne-teknologi, mer fra et kostnadsperspektiv enn fra et teknisk perspektiv. Se mer om implementasjonen av beacons i seksjon 5.6.

5 Design og implementasjon

Nå skal vi se nærmere på hvordan vi har implementert de forskjellige delene av vårt system. Vi har underveis forsøkt å følge de anbefalte måtene å implementere de ulike teknologiene på, men underveis i prosjektet ble det også tatt mange valg som påvirket sluttresultatet.

Alle kodesnuttene er hentet fra våre prosjekter.

5.1 Overordnet struktur

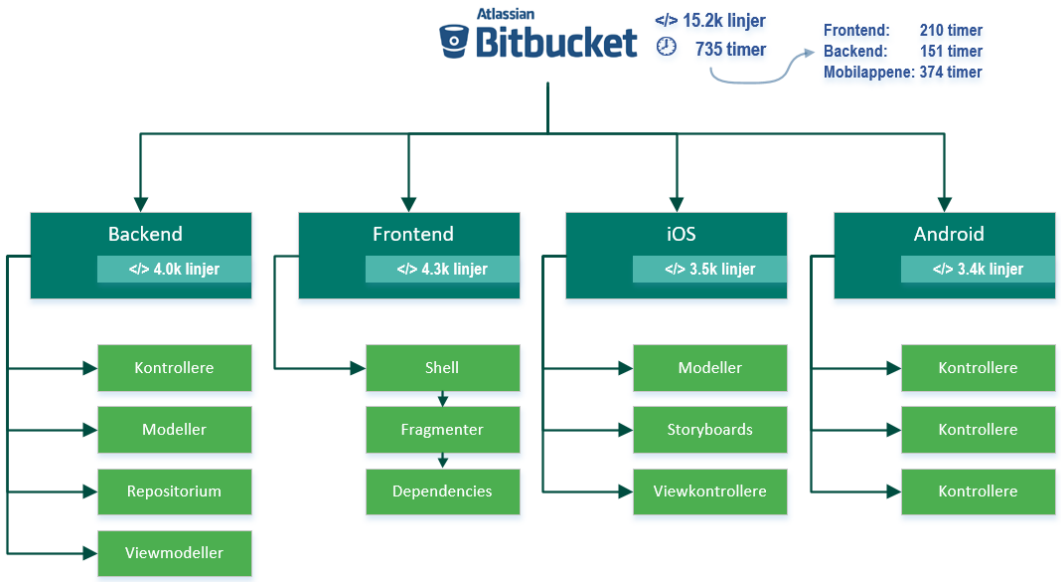


Figur 8: Systemkomponenter og kommunikasjon (laget i draw.io)

Vi valgte å implementere løsningen vår som en moderne RESTful API-tjeneste, hvor vi har en backend server som behandler data. Klienter kommuniserer med backend'en for å oppdatere/hente ut data. Klientene blir i fra et informasjonssynspunkt "lette" da de ikke lagrer informasjon lokalt på enheten. All informasjon hentes fra en server, som kommuniserer med en database. Siden APIet er skrevet i Microsoft .NET Core som leverer en enkel webserver kalt Kestrel, var det naturlig å bruke denne til dele ut APIet. Kestrel ble lansert sammen med Microsofts .NET 5 - forløperen til .NET Core- og har blitt oppdatert til å støtte .NET Core fullt ut. Kestrel er en enklere webserver, som er rask og laget for å bruke sammen med .NET Core plattformen. Microsoft anbefaler selv å bruke denne i produksjon[48]. Derimot skal aldri Kestrel stå direkte ut mot det åpne nettet, da den ikke er en fullverdig webserver[49]. Den mangler noe sikkerhet på grunn av dens unge alder - som etterhvert vil bli forbedret. Microsoft anbefaler derfor å ha den bak en *reverse proxy*, et råd vi følger. Vi bruker NGinx[50], som er en open source *reverse proxy* - mye brukt i produksjon. Den er rask, sikker og pålitelig.

På figur 8 kan vi se hvordan systemet er satt på på overordnet nivå. Webgrensenittet, iOS og Android applikasjonen kommuniserer alle med API'et gjennom HTTP-kall som POST, PUT, DELETE og GET. I seksjon 10 står det mer utdypende om hvordan konfigurasjonen på NGinx, Kestrel og produksjonsmiljøet generelt er satt opp.

Figur 9 viser grovstrukturen for selve kodebasen. Alt ligger i forskjellige *repositories* i Bitbucket og hvert repo er delt inn i overordnede lag, som figuren også viser. I tillegg har vi lagt til antall kodelinjer og timer brukt på kodingen, for å gi et inntrykk av størrelse.



Figur 9: Oversikt over repoer og kodelinjer (laget i draw.io)

5.2 Frontend web-grensesnitt

Administrering av all informasjon synlig i mobilapplikasjonene skal gjøres via web-applikasjonen. Dette verktøyet er lagd med rammeverket Polymer, som tidligere er beskrevet i seksjon 4.1.

Vi har brukt Polymers "App Toolbox" for både grunnleggende layout og navigasjon, såkalt "routing". *Polymer App Toolbox* er en "samling av komponenter, verktøy og templates for å bygge progressive web-applikasjoner med Polymer"[51].

Vi valgte en enkel responsiv applayout, kraftig inspirert av *Google Inbox*. Den grunnleggende layouten består av følgende:

- **Toppfelt** - Tittel og bakgrunnsfarge endres dynamisk ut i fra innholdet i hovedområdet.
- **Navigasjonsmeny** - På venstre side av skjermen. Denne skjules automatisk hvis nettleservinduet blir for smalt eller man prøver å bruke applikasjonen på mobil.
- **Hovedområde** - Siden som er valgt i menyen vises i dette området.

Se figur 10 for oversiktsbilde.

5.2.1 Elementer

Elementer er selve byggeklossene i Polymer. Man kan finne både offisielle Polymer-elementer og elementer lagd av privatpersoner i Polymers egen katalog [34], og selvfølgelig lage sine egne. Å legge inn funksjonalitet i form av elementer er med på å motvirke duplisering av kode, siden gjenbruk er veldig enkelt.

Innholdet i et element defineres i HTML, mens mer avansert funksjonalitet kjøres med JavaScript. Etter elementet er definert, kan det brukes på samme måte som en vanlig HTML-tag.

5.2.2 Routing

Navigering av applikasjonen gjøres ved hjelp av *routing*[52]. Dette er en prosess som endrer innholdet i "hovedområdet" basert på URL'en som står i adressefeltet.

Figur 10: Grunnleggende layout og områder (*frontend*)

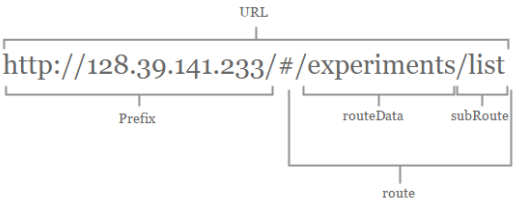
```
<app-location route="{{route}}"
              use-hash-as-path></app-location>
<app-route route="{{route}}"
         pattern="/:page"
         data="{{routeData}}"
         tail="{{subroute}}"
         active="{{pageActive}}"></app-route>
```

Listing 5.1: Oppsett av routing hentet fra vår kodebase

Elementet <app-location> (se kodelisting 5.1) har som eneste oppgave å hente inn URL'en i adressefeltet og gjøre den tilgjengelig til <app-route> som *route*.

I routing-eksempelet (se figur 11), setter <app-location> alt foran nummertegnet (#) som prefix, som ignoreres av <app-route>, og alt etter som "route".

<app-route> analyserer "route" og ser etter det som eventuelt er satt som *pattern*. I dette tilfellet blir alt mellom **første** og **andre** skråstrek definert som *page*, og lagret i variabelen *routeData*. Alt etter dette kalles *tail* og lagres i variabelen *subroute*.



Figur 11: Routing-eksempel

```
<iron-pages selected="{{routeData.page}}"
            attr-for-selected="data-page"
            role="main">
    <experiments-pages data-page="experiments"
        page="{{routeData.page}}"
        route="{{subroute}}"></experiments-pages>
</iron-pages>
```

Listing 5.2: Iron-pages

<iron-pages> er et element som fungerer som en enkel *content switcher*. Den velger hvilke elementer/sider den skal vise i hovedområdet basert på hva variabelen *routeData.page* er satt til. Hvis vi

fortsetter å bruke eksempelet over, er page nå "experiments". <iron-pages> har da sett at elementet <experiment-pages> har "experiments" som *data-page*, og derfor satt dette inn i hovedområdet (se figur 10).

Når man setter inn elementer i Polymer, kan man samtidig definere attributter som elementet kan ta i bruk internt. I kodelisting 5.2 er attributtene *data-page*, *page* og *route* definert i elementet <experiment-pages>. *Data-page* brukes bare for å velge hvilken element som skal være aktivt, men *page* og *route* brukes i elementet.

I samsvar med Polymers prinsipper om modularitet, bruker vi *encapsulated routing*[53] i hele applikasjonen. I stedet for å ha en sentralisert fil med alle mulige *paths*, håndterer elementet <experiment-pages> all *routing* relatert til eksperiment-sider. Lignende sider med routing brukes også i andre deler av applikasjonen.

```
<app-route route="{{route}}"
        pattern="/list"
        active="{{listActive}}"></app-route>

<app-route route="{{route}}"
        pattern="/new"
        active="{{newActive}}"></app-route>

<app-route route="{{route}}"
        pattern="/experiment/:id"
        data="{{experimentData}}"
        active="{{experimentActive}}"></app-route>
```

Listing 5.3: Routing i <experiment-pages>

I eksempelet over, er *route* nå satt til det som var *subroute/tail*, det vil si "list". Elementet <experiments-list> blir da satt til aktivt og vises i hovedområdet.

```
<experiments-list page="{{page}}"
                active="{{listActive}}"></experiments-list>

<experiment-edit experiment-id="{{experimentData.id}}"
                active={{experimentActive}}
                page="{{page}}"></experiment-edit>

<experiment-new page="{{page}}"
                active={{newActive}}></experiment-new>
```

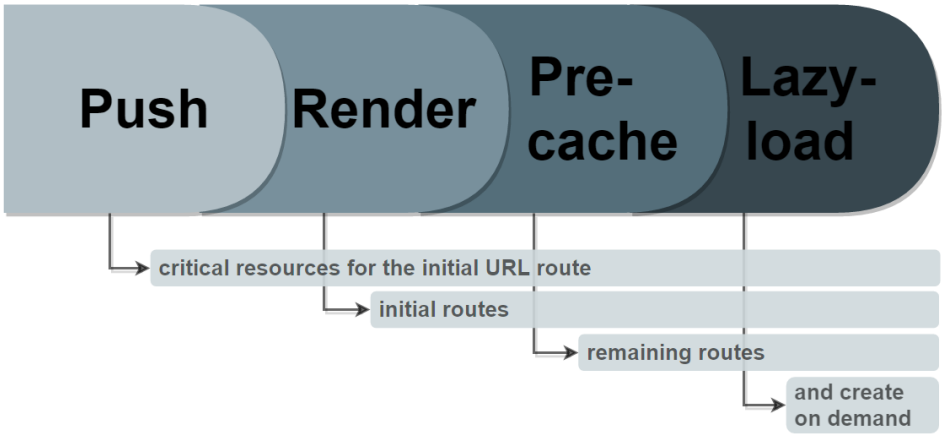
Listing 5.4: Elementer i <experiment-pages>

Hvis URL blir endret slik at *route* blir "/new", settes <experiment-new> inn. Hvis *route* f.eks. blir "/experiment/abe81a9d-06c7-4abc-a737-136ae5d6454c", settes <experiment-edit> inn for eksperiment med id "abe81a9d-06c7-4abc-a737-136ae5d6454c".

5.2.3 PRPL-pattern

Vi har satt opp web-grensesnittet som et SPA (Single-page application[54]) med en struktur som i stor grad følger **PRPL**-patternet[55] til Google.

Patternet ble navngitt av Polymer-teamet og lansert på Google I/O 2016. Det er et *pattern* lagd for å strukturere og serve progressive web-applikasjoner (PWAs) med fokus på ytelse ved oppstart og ved interaksjon.

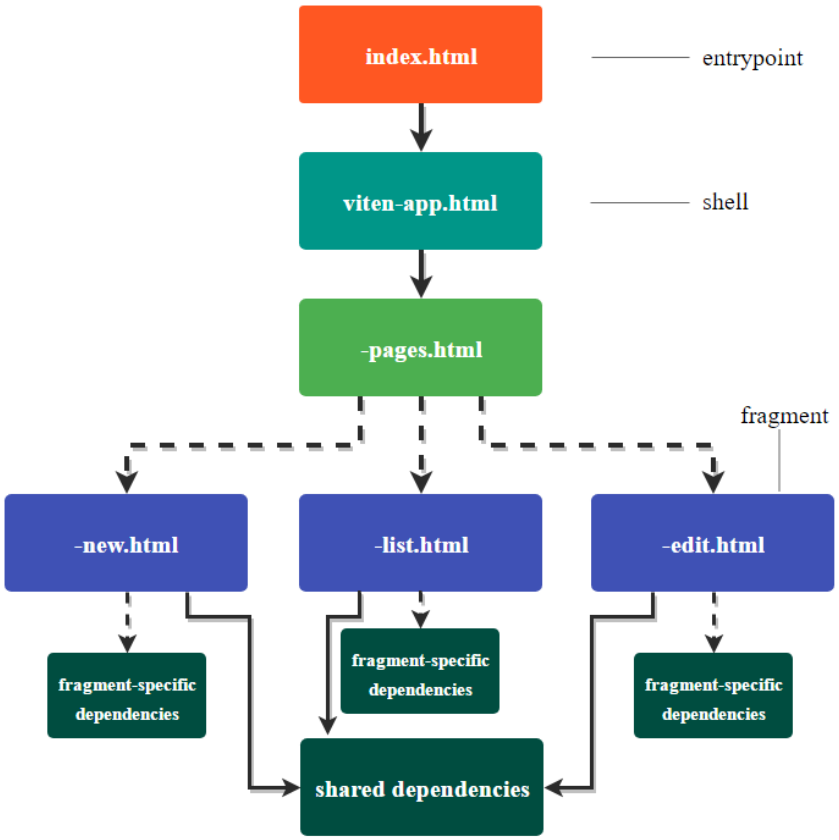


Figur 12: PRPL-patternet (laget i draw.io)

Figur 12 viser essensen i patternet.

1. De viktigste ressursene pushes via *entrypointet*.
2. Siden innstilt som standard vises (i vårt tilfelle en liste over eksperimenter).
3. De andre hovedsidene lastes inn i *cachen*.
4. De resterende sidene er definert og lastes inn ved behov (*lazy loading*[56]).

<app-route>, beskrevet i detalj i 5.2.2, definerer hvor i prosessen alle sider hører hjemme.



Figur 13: Struktur frontend (laget i draw.io)

5.2.4 Struktur

Figur 13 viser filstrukturen i webapplikasjonen vår.

- **Index.html** er *entrypoint* til applikasjonen. Den er bevisst gjort så liten som mulig, siden den blir brukt fra flere URL'er og dermed *cachet* flere ganger.
- **viten-app.html** er *shellet*. Her ligger all toppnivås app-logikk, <app-route>, menyer osv.
- **-pages.html** er hovedsidene for hver kategori av undersider/fragmenter som har som eneste oppgave å laste inn riktig fragment ut i fra aktiv URL. Eksperimenter er et eksempel på en slik kategori. Det er tre fragmenter som sorteres som eksperiment; experiment-new.html, experiment-edit.html og experiment-list.html, og experiment-pages.html har <app-route>-logikken for å vise brukeren riktig side.
- **-new.html**, **-list.html** og **-edit.html** er fragmenter som *lazy-loades* ved behov.

Når en bruker endrer *route*/URL, *lazy-load*'es de sidene som ikke allerede er cachet. Neste gang brukeren går tilbake til samme side, ligger den allerede i *cache*'n.

5.2.5 Kommunikasjon med backend

Alle forspørsler til, og svar fra, backend, er i JSON-format (se i neste seksjon (5.3) for detaljer om hvordan forespørslene blir behandlet på serveren).

Vi har brukt elementet <iron-ajax>[57] for å både sende til og motta fra backend. Dette elementet forenkler kommunikasjonen, blant annet ved å automatisk *parse* JSON-responsen til et objekt, eventuelt en *array*, som Polymer kan ta i bruk umiddelbart.

I kodelisting 5.5 er det et eksempel på en slik respons fra backend. Her har det blitt generert en forespørsel om informasjon om et spesifikt eksperiment som er sendt med <iron-ajax>.

```
{
  "scanId": "4ecc184f-d8a7-4b0d-8046-14f5effefeb5",
  "name": "Naermest null",
  "description": "Ved hjelp av sifferene 0 & 9 skal det ...",
  "hints": [
    {
      "hintNr": 1,
      "description": "Dersom du oppnar et godt resultat..."
    },
    {
      "hintNr": 2,
      "description": "Forholdet mellom den hoyeste faktoren..."
    }
  ],
  "extraFunctions": [
    {
      "name": "Naermest null spill",
      "actionAndroid": "no.ntnu.stud.mobile.fragments.MathGameFragment",
      "actionIOS": "ClosestToZero"
    }
  ],
  "imageId": "3ff2e3a5-deda-4b04-b8c1-5ef4d6d8c78c",
  "roomId": 1
}
```

Listing 5.5: Respons på forespørsel om informasjon om et eksperiment

Denne responsen blir automatisk *parset* til et objekt:

```
{
  _onSingleExperimentResponse: function(event) {
                let experiment = event.detail.response || [];
        }
}
```

Listing 5.6: Håndtering av respons fra backend

Polymer kan deretter lett hente ut informasjon, slik som f.eks. *experiment.name* for navnet, eller *experiment.hints.0.description* for beskrivelse av det første hintet.

5.3 Backend API

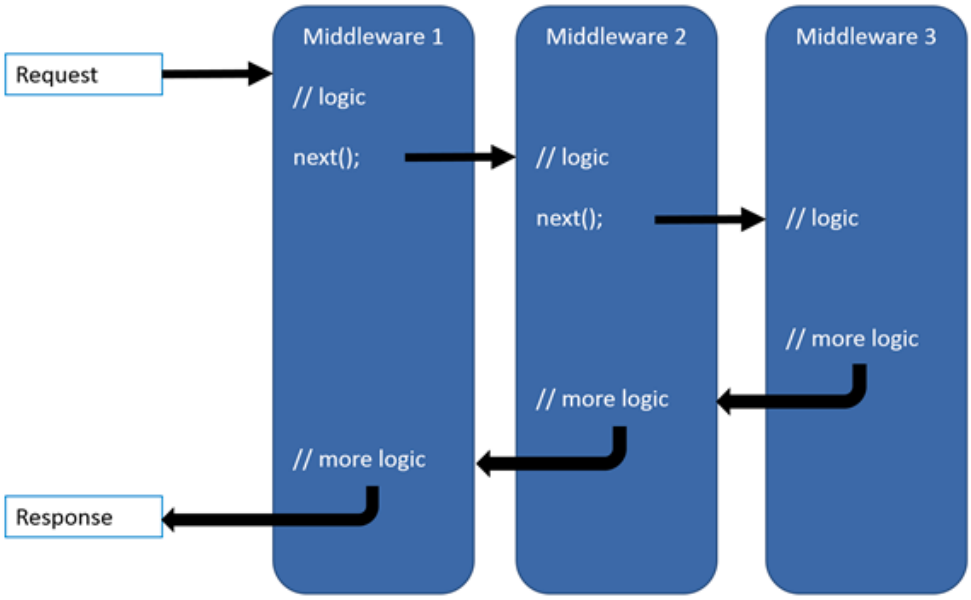
Før jul laget vi en liten test av API'et for prototypen vår, så vi hadde testet rammeverket før, men det var mer *serving* av statiske data og ikke et ferdig utviklet API. Da vi startet på bacheloroppgaven valgte vi derfor å starte helt på nytt. Vi hadde nå oppdaget andre måter å løse problemer og sette opp struktur på, som ville gjøre API'et til et fullverdig API som kunne skalere på en mer fornuftig måte. Erfaringene vi hadde tilegnet oss og som et av gruppemedlemmene hadde fra tidligere prosjekter gjorde det lettere å sette opp en fornuftig struktur som det ville være enkelt å bygge videre på i fremtiden. Vi fant også en god artikkel[58] som forklarte hvordan .NET Core kunne settes opp gjennom å bruke metoder som *repository pattern, dependency injection, request pipeline* og *MVC*. Rådene i denne artikkelen ble fulgt slavisk i starten av prosjektet, men ble tilpasset våre behov etterhvert som vi lærte mer.

Alle kodeeksemplene i denne seksjonen er hentet fra vårt prosjekt, og viser hvordan vi har anvendt de forskjellige teknologiene i backend.

5.3.1 Request pipeline

Når vi velger å bruke et rammeverk for å bygge løsningen, gjør dette at vi automatisk følger de samme retningslinjene som alle andre som koder i samme rammeverk. Dette fører igjen til at det blir lettere med vedlikehold, videreutvikling og feilsøking. Mange API-rammeverk idag gjør at utviklerne slipper å skrive nettverkslogikk selv. .NET Core tilbyr oss ikke bare nettverkslogikk, men en hel rekke av løsninger som vi kunne bygge videre på. Når et API mottar et kall fra en klient, ønsker vi å hente ut informasjon fra forspørselen, men også bearbeide kallet en eller flere ganger, før et resultat blir returnert til klienten. For å gjøre dette enklere for utviklerne har .NET Core laget et rekkeoppsett der hver *request* (forespørsel) går gjennom mange lag med bearbeiding og prosessering før det returneres til klienten. Dette kalles en *request pipeline*, og hvert ledd i denne rekken kalles en *middleware*. Som vi ser på figur 14 kommer en request fra klienten inn til serveren, går inn i den første middlewaren før den sendes videre til neste. Til slutt, når den har vært igjennom alle leddene, returneres en respons. Rammeverket har laget flere slike lag ferdig som for eksempel autentisering, CORS-støtte og logging. Dette er veldig fint for oss, da vi i starten ønsket å fokusere på funksjonalitet. Dette lar seg gjøre ved et slik oppsett, for når vi da vil legge på autentisering legges det bare på en ekstra *middleware* som autentiserer *request*'en i *pipeline*'n.

Det er enkelt å legge til flere lag i *pipeline*'n, enten ferdige eller egenutviklede *middleware*'s. For å legge til et steg i prosesseringen av en request angir du dette i metoden *Configure* som kjøres i startup. Denne metoden kalles *runtime*, og mottar *interface*'t IApplicationBuilder som definerer stegene i *request pipeline*'n.



Figur 14: Pipeline middleware .NET Core. Diagrammet er hentet fra [1]

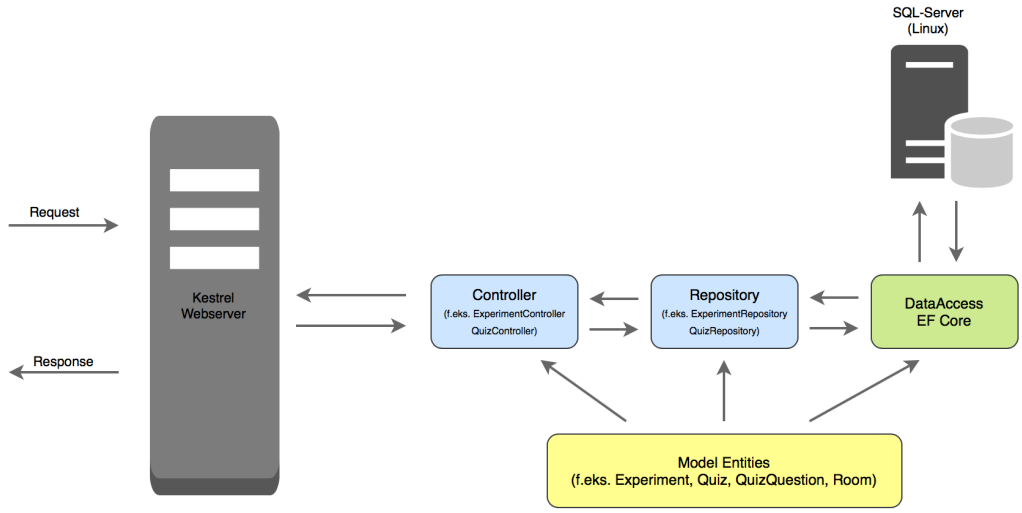
```
1  public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
2          {
3              app.UseCors(builder =>
4                  builder.AllowAnyOrigin()
5                  .AllowAnyHeader()
6                  .AllowAnyMethod());
7              app.UseSwagger();
8              app.UseSwaggerUi();
9              app.UseStaticFiles();
10             app.UseMvc();
11         }
```

Listing 5.7: *Configure*-funksjon som setter opp *pipeline*

I listing 5.7 ser vi hvordan vi satte opp *pipeline*'n i *backend*-applikasjonen, blant annet satte vi her opp *CORS*. Dette gjør at alle adresser kan gjøre forespørsler mot serveren vår. Merk at dette ikke gjøres ut mot nettet, men mot lastbalansereren vi har foran serveren - på denne settes de reelle *CORS*-reglene. Videre settes swagger opp som en del av *pipeline*'n noe som gjør at vi får autogenerert dokumentasjon som kan brukes av utviklerne.

Generell struktur

Når *request*'en har passert gjennom deler av *pipeline*'n, og kommet frem til kontrolleren starter logikken som skal behandle dataene lagret i databasen. Siden dette er kjernen i hva en *backend* skal utføre, blir det mange kall opp mot databasen. Hvis disse kallene ikke sentraliseres ville resultatet ha blitt mye duplisering av kode, med mange kodesnutter som gjør det samme. For å løse disse problemene anvender vi patternet "Repository Pattern"[59]. Dette patternet sentraliserer logikken som utfører databasekall i egne *repository*-klasser. I figur 15 kan vi se hvordan repoene brukes av kontrollerklassene til å oppdatere data. Dette gjør at kontrollerne ikke trenger å vite hvordan dataene er lagret eller hvordan selve spørringene opp mot databasen utføres, den kaller bare funksjonene i repoene. Alle klassene bruker de samme modellklassene, som betyr at de deler en felles representasjonen av dataene.



Figur 15: *Repository pattern*-implementasjon (laget i draw.io)

Dette gjør at systemet vårt oppnår en løs kobling, slik at utviklerne faktisk underveis i prosjektet kan implementere en helt ny databaseimplementasjon, uten å røre annen kode. Så lenge kontrakten(*interfacene*) ut mot de andre klassene forblir det samme, vil ikke dette ødelegge for koden som ligger ellers i systemet.

Ved å bruke Repository Patternet er det også enkelt å skrive tester som tester controllerlogikken, da repositoriene kan *mockes*(fakes). På denne måten kan det skrives tester som utelukker feil i databasekallene, og bare tester logikken i selve *controlleren*. I kapittel 9 skriver vi mer om testing.

Alle modellene trenger et *repository* tilknyttet seg, men funksjonene som skal utføres er veldig like og den eneste forskjellen er ofte datatypen. For eksempel skal alle modellene ha funksjoner som henter ned et objekt med en gitt id. Derfor var det ønskelig å lage et felles *repository* med de aller vanligste operasjonene. Dette løste vi ved å lage en generisk klasse, som ikke var knyttet til en bestemt type. For å forstå hvordan dette er implementert og hvordan *controllerne* får tilgang til disse repositoriene må vi se på en annen sentral teknologi .NET Core benytter seg aktivt av: *Dependency Injection*.

Dependency Injection

Dependency injection(DI) er en metode for å oppnå løs kobling mellom objekter. Istedenfor å initialisere objekter i klassen, eller refererer statisk til dem, skytes objektene inn som en *dependency* under runtime[8]. Det er flere forskjellige måter å skyte inn objektene, men i .NET Core er det vanlig å bruke det som kalles *constructor injection (CI)* som vi også bruker her. For å holde rede på alle objekter som tilbys gjennom DI brukes en container. .NET Core tilbyr en enkel DI-container: IServiceProvider. For å lage en service som skal tilbys gjennom CI registreres objektet hos *IServiceProvideren* under oppstart. .NET Core lar deg ikke bare unytte fordelene med DI, det brukes også aktivt i implementasjonen av selve rammeverket[60].

Objekter som skal tilbys gjennom DI må implementere et interface. Dette interfacet registreres hos service provideren, som igjen legger dette i DI-containeren. I listing 5.8 ser vi hvordan en ExperimentRepositoriet med tilhørende interface ble registrert som en service.

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    services.AddScoped<IExperimentRepository, ExperimentRepository>();
    ...
}
```

Listing 5.8: Registrering av ny service

For å få tilgang til et objekt av ExperimentRepository under *runtime* defineres bare IExperimentRepository som et parameter til constructoren til den klassen du ønsker tilgang fra. Et eksempel ser vi i listing 5.9. Da vil det under runtime kjøres en spørring mot containeren IServiceProvider satte opp, og servicen vil skytes inn i containeren automatisk.

```
public ExperimentsController(IExperimentRepository experimentRepository)
{
    _experimentRepository = experimentRepository;
}
```

Listing 5.9: Utdrag fra ExperimentController, oppsett av DI

Generisk *repository*

Nå vet vi hvordan kontrollerne får tilgang til *repository*'ene som tar seg av all databehandlingen opp mot databasen. Som nevnt over er de fleste av disse kallene opp mot databasen like, og avhenger kun av typen objekt det skal utføres på. Det var derfor fornuftig å definere et felles *repository* som en generisk klasse. Slik at alle de forskjellige repoene kan ta i bruk disse fellesfunksjonene. Vi lager derfor en generisk klasse kalt EntityBaseRepository, som kan behandle alle objekter som implementerer *interface*'t IEntityBase - noe alle modellene våre gjør. I listing 5.10 kan vi se hvordan noen av disse funksjonene i denne klassen er skrevet. Her ser vi at ved initialisering av denne klassen kan vi definere hvilken type vi opererer på, som igjen vil si at det kan brukes med alle modellobjektene våre.

```
public class EntityBaseRepository<T> : IEntityBaseRepository<T> where T : class, IEntityBase, new()
{
    ...
    public T GetSingle(int id)
    {
        return _context.Set<T>().FirstOrDefault(x => x.Id == id);
    }


    public void Add(T entity)
    {
        EntityEntry dbEntityEntry = _context.Entry<T>(entity);
        _context.Set<T>().Add(entity);
    }
 ...
}
```

Listing 5.10: Utdrag fra EntityBaseRepository

Etterhvert oppdaget vi at i noen tilfeller trengte spesifikke modeller egne og tilpassede funksjoner. Disse funksjonene var det ikke hensiktsmessig å legge i det generelle repoet. Derfor definerer vi egne *repositories* for alle modellene. Et eksempel på dette er modellklassen Experiment som trengte to egne

funksjoner som bare gjelder for objekter av typen Experiment. Da var det ingen grunn til at dette skulle ligge i den generiske klassen. Disse funksjonene ligger da i ExperimentRepository, som igjen implementerer EntityBaseRepository med Experiment som type. Dette gir tilgang til de generelle funksjonene også. Under kan vi se interfacet som definerer ExperimentRepository:

```
1  public interface IExperimentRepository : IEntityBaseRepository<Experiment>
2  {
3      IQueryable<Experiment> BuildExperiment();
4      Experiment GetSingleByScanId(IQueryable<Experiment> query, string scanId);
5  }
```

Listing 5.11: Interfacet IExperimentRepository

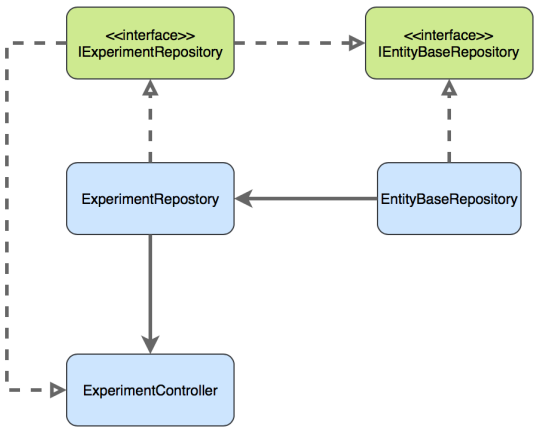
Her kan vi se at dette *interface*'t implementerer det generelle *interface*'t EntityBaseRepository, men forteller at det vil bruke det med Experiment som type. Derfor vil alle generelle funksjonene som befinner seg inne i dette, bli funksjoner som behandler modeller av type Experiment.

Vi bruker så *dependency injection* til å gi tilgang til alle repoene i kontrolleren, som gjør det mulig å kalle på funksjoner i repoet. For eksempel vil vi i RoomsController hente ned et Room fra databasen med en spesifikk id. Et eksempel på hvordan dette kallet ser ut ser vi i listing 5.12.

```
1  var room = _roomRepository.GetSingle(newExperiment.RoomId);
```

Listing 5.12: Eksempelkall til room *repository*

Ved å strukturere applikasjonen vår på denne måten får vi separert logikk på en fornuftig måte, og det er enkelt å skrive om en del av applikasjonen så lenge *interface*'ne forblir de samme. Alle modellene har egne repoer hvor egne funksjoner kan defineres, men for å unngå duplisering av kode bygger alt på et generisk *repository*. Hvordan dette henger sammen på typen Experiment kan vi se på figur 16. *Dependency injection* er en veldig fin måte å sette opp en slik struktur på. Siden .NET Core er bygget opp på denne tankemåten var det ingen grunn til å ikke følge det på samme måten i prosjektet vårt.



Figur 16: Oppsett av dependency injection på Experiment

EF Core

Det finnes flere måter å opprette databaser når vi utvikler et system. Microsoft har et veldig bra og veldokumentert rammeverk kalt Entity Framework Core[61], som er en omskriving av Entity Framework for .NET. Dette er et Object-Relation Mapper (ORM) som støtter en metode som kalles *Code First*[62].

Code First vil si at databasegenereringen starter med å skrive modellklasser og *context*'er, ikke med å manuelt lage en database. Når applikasjonen kjører genereres denne databasen utfra hvordan modellene og *context*'en er definert. Derfor styres alle koblinger og referanser i databasen av hvordan modellene er koblet til hverandre.

```
public class Experiment : IEntityBase
{
    public int Id {get; set;}
    ...
    public ICollection<Hint> Hints {get; set;}
    public ICollection<ExtraFunction> ExtraFunctions {get; set;}
    ...
}
```

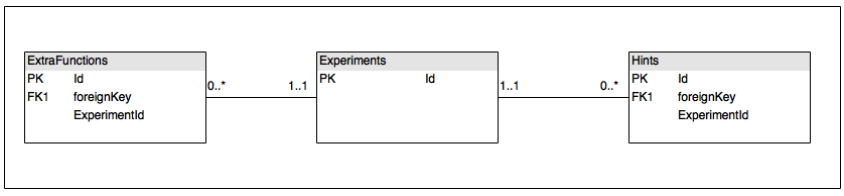
Listing 5.13: Utdrag fra modellklassen Experiment

```
public class Hint : IEntityBase
{
    public int Id {get; set;}
    public int HintNr {get; set;}
    public string Description {get; set;}

    // Link to experiment one to many
    public int ExperimentId {get; set;}
    [JsonIgnore]
    public virtual Experiment Experiment{get; set;}

    ...
}
```

Listing 5.14: Utdrag fra modell klassen Hint

```
public class ExtraFunction : IEntityBase
{
    public int Id{get; set;}
    public string Name {get; set;}
    public string Action{get; set;}

    public int ExperimentId{get; set;}
    [JsonIgnore]
    public virtual Experiment Experiment{get; set;}
    ...
}
```

Listing 5.15: Utdrag fra modellklassen ExtraFunction



Figur 17: Eksempel på databaseskjema, generert av *code first*

La oss se et eksempel. Listing 5.13 viser et utdrag fra klassen Experiment. Et eksperiment kan ha mange hint og mange ekstrafunksjoner. Listing 5.14 viser oss modellen Hint. Denne har en kobling tilbake til eksperimentet den tilhører, det samme har modellen ExtraFunction (listing 5.15). Dette oppsettet ville resultert i en databasetabellstruktur som i figur 17. Dette er *code first*, vi definerer hvordan

modellene våre henger sammen, så vil det automatisk bli omgjort til en database. For å få enda mer kontroll kan vi definere disse koblingene mer manuelt. Vi må også beskrive hver tabell i databasen. Dette gjøres i *context*'en som beskriver databasen vår. Hvordan dette gjøres ser vi i listing 5.16. Her beskrives alle koblinger mellom tabellene.

```
class BackendContext : IdentityDbContext<ApplicationUser>
    {
        public DbSet<Experiment> Experiments {get; set;}
        public DbSet<ExtraFunction> ExtraFunctions {get; set;}
        public DbSet<Hint> Hints {get; set;}
        ...
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        foreach (var relationship in  modelBuilder.Model.GetEntityTypes().SelectMany(e => e.GetForeignKeys()))
        {
            relationship.DeleteBehavior = DeleteBehavior.Restrict;
        }

        modelBuilder.Entity<Experiment>()
            .ToTable("Experiments");

        modelBuilder.Entity<Hint>()
            .ToTable("Hints");

        modelBuilder.Entity<ExtraFunction>()
            .HasOne(h => h.Experiment)
            .WithMany(e => e.ExtraFunctions);

        modelBuilder.Entity<Hint>()
            .HasOne(h => h.Experiment)
            .WithMany(e => e.Hints);

        ...
    }
}
```

Listing 5.16: Utdrag fra modell klassen ExtraFunction

Autentisering

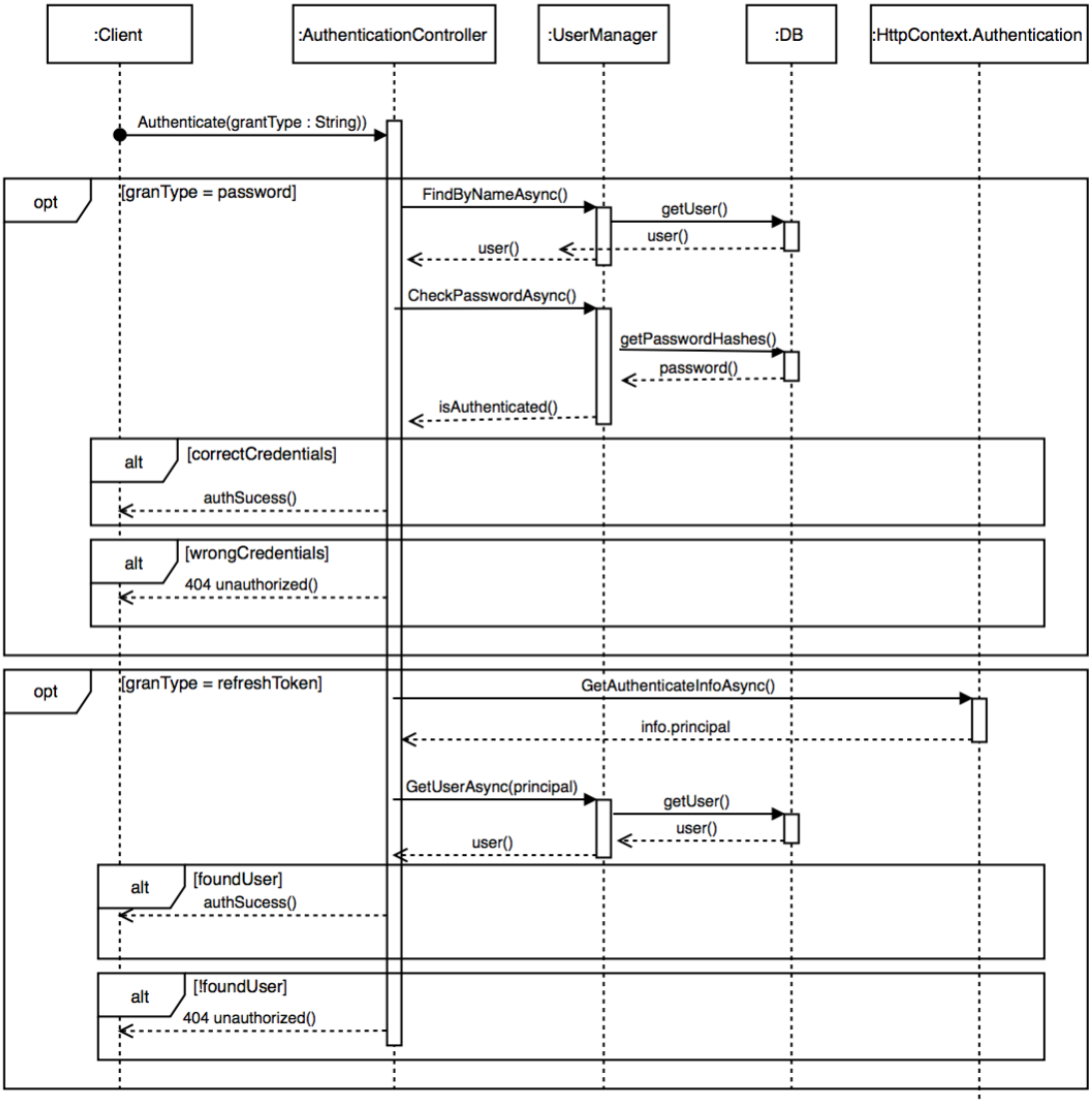
Siden det bare er ansatte på Vitensenteret som skal ha tilgang til å endre og legge til informasjon gjennom webapplikasjonen, må endel av *endpoint*'ene i backend-API'et ha autentisering. Det er flere måter å løse autentisering av API'er på. En av de anbefalte måtene er ved å bruke JSON Web Tokens (JWT) [63], dette er *tokens* generert av serveren, og brukes av klienten for å verifisere at den har gjennomført autentisering. Fordelen ved å bruke *tokens*, er at serveren ikke trenger å lagre noe informasjon om hvem som er logget inn, og dette gjør løsningen rask og effektiv. JWT er ment til å være en en enkel løsning å sette opp, minuset er at *token*'ene ikke er kryptert, og alle kan dekryptere de og få mye informasjon om serveren. En annen mer fleksibel løsning som også baserer seg på *tokens* er OAuth[64]. OAuth er en løsning som ble laget for å bruke andre autnentiseringsmetoder for dine egne løsninger, for eksempel kan man logge inn gjennom sin egen Facebook-profil. Løsningen kan også utvides til å støtte en rekke andre ting, som epostverifisering, toveisautentisering osv.

Microsoft har allerede et veletablert rammeverk for å lagre informasjon om brukere, deres identi-

teter og rettigheter. Disse kalles Roles og Claims. Nylig laget Kévin Chalet et rammeverk designet for .Net Core som heter *OpenIddict-Core*[65]. Dette er et rammeverk som ved hjelp av Microsoft sitt rammeverk Identity implementerer en variant av OAuth. Ideen bak rammeverket er å gjemme en del av det kompliserte rundt autentiseringsprosessen. Microsoft skrev om rammeverket[66] og anbefaler det som en implementasjon av *token*-basert autentisering. Biblioteket dekker vårt behov på en god måte. Vitensenteret hadde ønske om ulike tilgangsnivåer til de ulike brukerne. Ved å bruke *openiddict* fikk vi disse tilgangsnivåene gjennom *Microsoft Identity*, uten at vi måtte manuelt sette opp et eget rollesystem.

Underveis i implementasjonen merket vi at dette var et tema vi ikke hadde mye erfaring med, og vi brukte mye tid på å forstå systemet. Vi kontaktet også Kévin Chalet for å stille noen oppklarende spørsmål (vedlegg H), og om vår måte å implementere dette på var ok. Dette gjorde at vi fikk en bekreftelse på at det vi gjorde ikke var helt feil, og det ble lettere å fortsette på en riktig måte.

Openiddict går inn i *pipeline*-systemet til .NET core. Dette gjorde at vi kunne skru på autentisering enkeltvis for de ulike *endpoint*'ene etterhvert som webapplikasjonen implementerte autentisering.



Figur 18: Flow av autentisering (laget i draw.io)

Flyten i autentiseringen kan vi se i figur 18. Når en klient vil autentisere seg, sendes det et HTTP

POST kall til *endpoint*'et */api/users/connect/token*. Vi har satt opp to mulige autentiseringsmetoder, brukernavn/passord eller *refresh tokens*.

Refresh Tokens

Et problem med *token*-basert autentisering får vi når en server har signert en *access token*. Serveren har etter at en *token* er utsendt ingen mulighet for å trekke den tilbake. Dette er en potensiell sikkerhetstrussel hvis noen skulle få tak i en av *token*'ene. For å løse denne problemstillingen ønsker vi å ha kortest mulig levetid på *tokens* utgitt av serveren. Brukeren må da autentisere seg på nytt så fort *token*'en slutter å være gyldig, men det er ikke ønskelig å bruke brukernavn/passord hver gang. Det kan da være en fristende mulighet å lagre brukernavn/passord kombinasjonen lokalt på klienten - noe vi ikke ønsker.

For å løse dette har vi to typer *tokens* som klienten mottar når den autentiserer seg, en *access token*, og en *refresh token*. *Refresh token* har lang levetid, og kan brukes til å reautentisere brukeren, uten at brukernavn og passord er påkrevd. Det er derfor to måter å autentisere mot dette *endpoint*'et, og det bestemmes av variabelen *grantType* som settes i HTTP-forespørselen.

Ved passord sendes en HTTP POST forespørsel sammen med brukernavn og passord. Dette kallet mottas av AuthenticationController (se figur 18). Det første kontrolleren gjør er at den prøver å finne brukeren, dette verifiserer at brukernavnet er reelt. UserManager sjekker opp om det finnes en bruker med dette brukernavnet i databasen. Finnes brukeren vil et brukerobjekt returneres til kontrolleren. Neste steg er å verifisere passordet. Kontrolleren kaller funksjonen CheckPasswordAsync i UserManager, som først hasher passordet med samme algoritme som det lagrede passordet i databasen, og sjekker om den eksisterende hashen i databasen er lik den genererte hashen. Hvis dette er tilfellet kan brukeren anses som autentisert og det retureneres en *access token* og en *refresh token*. *Access token* er da klar til å brukes mot de beskyttede *endpoint*'ene.

Det fine med disse *token*'ene som genereres er at de ikke er JSON *web tokens*, men kryperte *tokens* som følger Microsoft sin standard. Det vil si at en potensiell angriper ikke vil kunne hente ut noe informasjon fra disse. Siden vi har kort levetid på *access tokens* vil en angriper ikke kunne oppnå permanent tilgang til serveren, kun ved å sniffe opp denne.

5.4 iPhone applikasjon

5.4.1 Arkitektur

Model-View-Controller

iOS-applikasjonen er skrevet i Swift 3 og arkitekturen er basert på Apple sin variant av Model View Controller-patternet. Dette er en av Apple sine anbefalte arkitekturer for iOS applikasjoner[2]. I et program som er basert på MVC så gis hvert objekt en rolle. Et objekt er enten en modell, en kontroller eller et *view*. Videre vil vår implementasjon av MVC illustreres. Kommunikasjonen i arkitekturen eksemplifiseres ved å vise interaksjonen mellom komponentene i det en bruker skanner et eksperiment.

5.4.2 Apple sitt MVC-pattern

I iOS-miljøet snakkes det om at Apple sitt MVC-pattern ikke er ekte MVC, men en undervariant[67][68]. Det argumenteres også for at Apple sin implementasjon av MVC egentlig følger MVP-patternet(Model-View-Presenter)[68].

MVP er et pattern hvor man har "dumme" views uten logikk og en *presenter(controller)* som håndterer UI-logikken og setter data i *view*'et. *Presenter*'en er en mellommann som håndterer interaksjonen mellom *View* og Modell. All *state* lagres i *presenter*'en[68] og modell og *view* er helt adskilt og snakker aldri sammen direkte. Definisjonen av MVP passer godt med hvordan vi har implementert iOS-applikasjonen - *views* er "dumme" og inneholder ingen logikk, *view controller*'ne håndterer interaksjon og *view*-logikk, modellen holder data. Samtidig er det en vesentlig forskjell mellom vår arkitektur og MVP at vi lagrer all *state* i modellen til forskjell fra MVP hvor *state* lagres i *presenter*.

Vår arkitektur er basert på Apple sin måte å definere MVC på, og vi definerer derfor arkitekturen

vår som MVC. Det kan argumenteres for at den viktigste faktoren for at man skal kunne definere arkitekturen som MVC er at man har skilt objektene i modeller, *views* og kontrollere og at det videre finnes underkategorier av MVC[69], for eksempel MVP.

Arkitektur som velges må være basert på applikasjonen man skal lage. Et godt argument for å bruke MVC er at man oppnår lav kobling mellom objektene noe som gjør at de blir gjenbrukbare. Spesielt modellene som er ren databehandling er objekter som kan være aktuelt å gjenbruke med andre kontrollere og *views*. Modellen Quiz kan for eksempel brukes som modell for quiz i et annet program.

Modell

Modellene er klasser og *struct*'er med data og funksjoner. *Struct*'er er valgt der objektene kun representerer enkle datatyper og hvor respektive objekter ikke har behov for komplekse funksjoner som jobber på objektene. På Apple sin utviklerkonferanse i 2015 fremhevet utviklingsteamet bak Swift hvordan *struct*'er er mye brukt i oppbyggingen av selve språket og at utviklere også bør bruke det mer[70]. *Struct*'er er ikke referanseoverført og de er derfor "tryggere" å bruke enn klasser[71]. Listing 5.17 viser skallet av datamodellen for et eksperiment. For et eksperiment er det valgt å bruke en klasse da et Eksperiment behøver flere komplekse funksjoner som jobber opp mot *backend*-API'et.

```
1  class Experiment {
2    var id: String?
3    var scanId: String?
4    var name: String?
5    var description: String?
6    var hints = [Hint]()
7    var extraFunctions = [ExtraFunction]()
8    var imageId: String?
9    var roomId: Int?
10 }
```

Listing 5.17: Eksperimentmodell

View

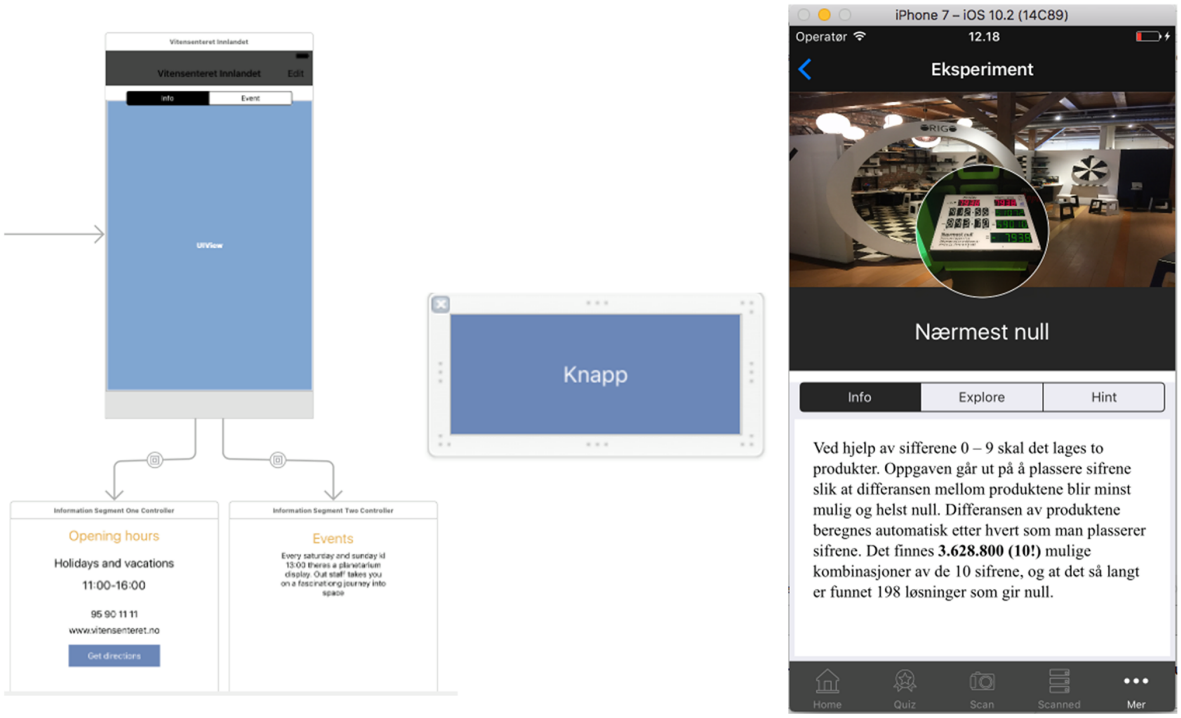
I applikasjonen er *views* representert gjennom *storyboards*, *ViewControllere*, *xib*-filer og tilhørende Swift-klasser som konfigurerer disse. Et storyboard inneholder en eller flere *ViewController*'e og illustrerer hvordan de interagerer på høynivå. En *view controller* er en enkelt side på applikasjonen og inneholder flere *view*-komponenter - knapper, tekst, bilder og mer. En xib-fil inneholder et enkelt *view* som er en del av en *view controller*. Eksempel på komponenter som kan ligge i xib-filer er en enkelt knapp, en header eller et bilde. Alle views i en *view controller* trenger ikke ha en egen xib-fil. Figur 19 viser henholdsvis eksempler på et storyboard, et xib-view og en *view controller*.

Figur 20 viser koden for referanse til views gjennom outlets. En outlet er referanse til én komponent i grensesnittet, for eksempel et bilde.

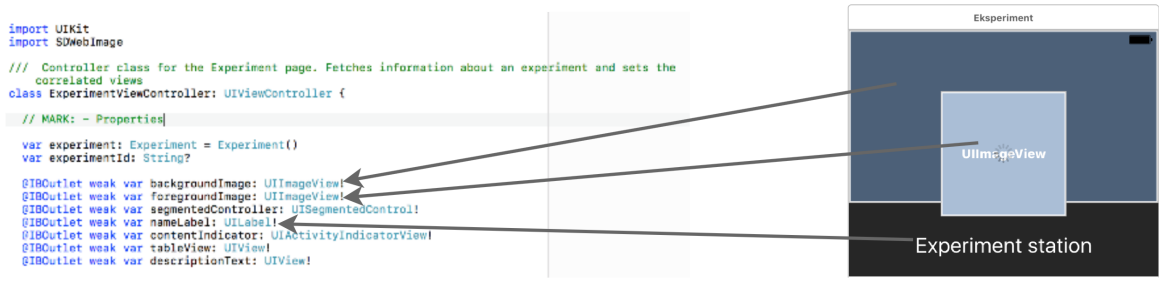
Controller

Controller-filer er Swift-klasser som er koblet til *view controller*'e i storyboard, men kontrollerne kan også være klasser som er *delegates* og/eller *data sources*. Klasser som er *delegates* og *data sources* håndterer *call back*-funksjoner for spesifikke *views*. En Swift-klasse kan både være en *view controller*, en *delegate* og en *data source* på samme tid.

Presisering: Det vi definerer som kontrollere heter i iOS-verden *view controller*'e. Det kan være forvirrende å lese at det vi definerer som kontroller heter *view controller* når *view* og kontroller helst skal være adskilt, men dette er slik arkitekturen er anbefalt for iOS fra Apple sin side[2]. *Views* og kontrollere er fortsatt adskilt på den måten at hvordan *views* ser ut er definert i egne filer, men all logikk ligger i *view controller*-filene.



Figur 19: Views



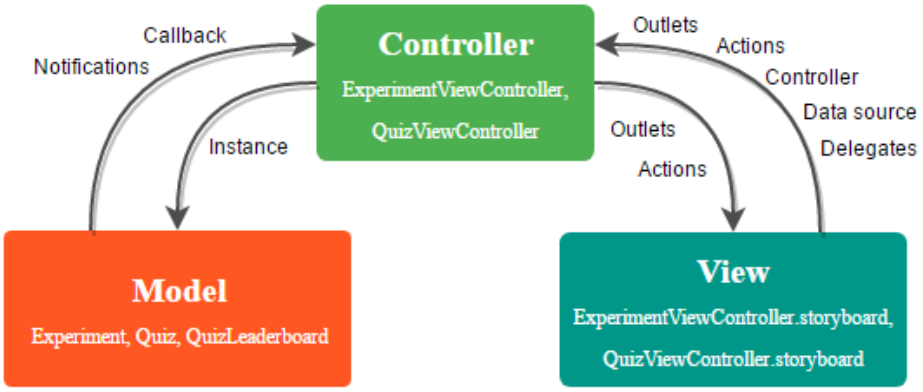
Figur 20: Outlets og controller

I applikasjonen vår har vi brukt en kombinasjon av *callbacks* og *notifications* for å gi beskjed om modellendringer. *Callbacks* er funksjoner i kontrollere som sendes med som parameter til modellene, *notifications* er *broadcasts* med data som sendes ut basert på en streng som klasser registrerer en *observer* på.

Implementasjon

Figur 21 illustrerer hvordan MVC er implementert i applikasjonen. Kontrolleren får beskjed om endringer fra *views* gjennom kontrollere, *outlets*, *actions*, *delegates* og *data sources* samt at den gir beskjed om endringer gjennom *outlets* og *actions*. Den gir beskjed til modellen gjennom å bruke en instans av

modell-objektet. Modellen gir beskjed om endringer til kontrolleren ved å kalle *callback*-funksjoner i kontrolleren samt ved å sende ut *broadcast notifications*, et meldingssystem i iOS som sender ut meldinger som man kan "abonnere" på i koden.



Figur 21: MVC Illustrasjon[2] (laget i draw.io)

5.4.3 Case: QR-skanning av eksperiment

Når appbrukeren er på senteret og skal skanne et eksperiment så starter interaksjonen med at brukeren åpner QR-skanneren(*view*'et). Når QR-koden er funnet gis kontrollen til QR-skanneren sin kontroller. Denne kontrolleren gir videre kontrollen til Eksperimentet sin *view controller*. Eksperimentkontrolleren bruker instansen av Eksperimentmodellen og kaller en funksjon fra modellen som starter å hente eksperimentdata fra API'et, kontrolleren sender med en egen *callback*-funksjon som modellen skal kalle senere når data er oppdatert. Kontrolleren sier altså i fra til modellen om at nå har brukeren bedt om noe i *view*'et.

Videre kaller modellen en hjelpefunksjon som gjør et nettverkskall og som returnerer et Eksperiment når den er ferdig. Når den er ferdig så får modellen beskjed og modellen kaller da funksjonen som ble parametisert fra kontrolleren i utgangspunktet. Når kontrolleren får beskjed via *callback*'et så henter den de nylig oppdaterte dataene fra modellen og setter de i *view*'et slik at brukeren ser de nye dataene - eksperimentnavn, bilde av eksperimentet og beskrivelse. Eksperimentsiden har flere kontrollere, *callback*'et som blir kallet ligger i ExperimentViewController og oppdaterer nevnte variable. Rombilde, hint og ekstrafunksjoner blir styrt av andre kontrollere og for å gi disse beskjed om modellendringer så har vi brukt *broadcast notifications*.

5.4.4 Nettverking

Kommunikasjon med REST-API'et er en svært sentral del av applikasjonen. Hvert *endpoint* i API'et har en tilhørende klasse i applikasjonen som kalles ("Endpointnavn")Service. Videre er det definert protokoller(*interfaces*) som deklarerer funksjoner som skal kunne brukes opp mot API'et. Ettersom en klasse kan implementere flere protokoller blir klassedeklarasjonene veldig lettleselige ved at man raskt får oversikt over hvilke operasjoner som kan utføres på et gitt *endpoint*. Listing 5.18 viser deklarasjon av protokollene Gettable og Uploadable og klassedeklarasjon knyttet til *endpoint* for et Eksperiment og et leaderboard.

```swift
protocol Gettable {
  associatedtype Data
  static func get(id: AnyObject, completionHandler: @escaping ([Data], Int) -> Void)
}

protocol Uploadable {
  associatedtype Data
  static func upload(id: Int, obj: Data, completionHandler: @escaping (Data, Int) -> Void)
}

class ExperimentService: Gettable {

}

class LeaderboardService: Gettable, Uploadable {

}
```

Listing 5.18: Protokoller

En alternativ implementasjon av nettverkslogikken ville vært å lage et arvehierarki, i dette tilfellet er det like praktisk og leselig å bruke protokoller. En klasse kan kun arve fra én enkelt klasse, men den kan implementere flere protokoller. Denne måten å implementere koden på er også mer i tråd med prinsippene i det Apple kaller protokollorientert programmering. Protokollorientert programmering går ut på at man bruker protokoller og protokollutvidelser(her kan man gi funksjoner en standardimplementasjon på samme måte som i en baseklasse i et arvehierarki) istedenfor å bruke arvehierarkier[72].

5.5 Android applikasjon

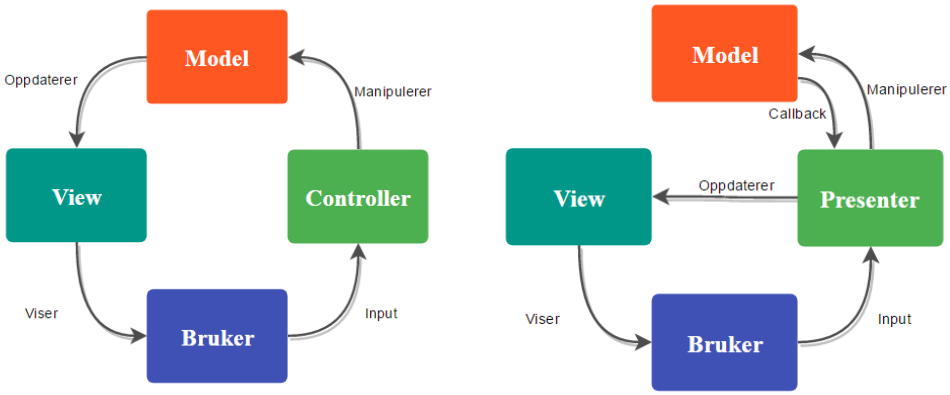
5.5.1 Arkitektur

I forprosjektet (se seksjon 1.9.2) lagde vi en fungerende pilotversjon av Androidapplikasjonen med grunnleggende funksjonalitet. Da vi skulle fortsette på utviklingen i hovedprosjektet, innså vi at en struktur som passer i en liten pilotapplikasjon ikke nødvendigvis fungerer like bra når størrelse og kompleksitet skaleres opp. For å gjøre alt ryddigere og enklere å vedlikeholde, bestemte vi oss for å skille logikken fra hverandre etter prinsippene til MVC/MVP, og samtidig følge retningslinjene på developers.google.com.

5.5.2 Model-view-presenter

I tradisjonell MVC [73] kommuniserer modellen med *view*'et direkte (se figur 22a). På iOS-applikasjonen, har vi fulgt Apples versjon av MVC, der modellen er passiv og får kontrolleren til å oppdatere *view* med *callbacks* (se seksjon 5.4 for mer detaljer om iOS-implementasjonen). På Android-applikasjonen har vi valgt å bruke MVP-patternet (*model-view-presenter*[74]), som er en avart av denne MVC-versjonen (se figur 22b).

Implementasjonen er veldig lik den på iOS, med noen plattform-spesifikke forskjeller.



(a) (b)

Figur 22: Tradisjonell MVC (a) og MVP (b) (laget i draw.io)

Model: Klasser

```
1  public class Experiment implements Getable<String, Experiment>, Saveable {
2      private String name;
3      private String description;
4      private String scanId;
5      private String imageId;
6      private ArrayList<Hint> hints;
7      private ArrayList<ExtraFunction> extraFunctions;
8      private String roomId;
9  }
```

Listing 5.19: Modellen til “Experiment”

Modellene er klasser som er satt opp slik at de speiler den databaseenheten de skal representere, og har de samme variabelnavnene. I kodelisting 5.19 kan man se at variabelnavnene er identiske med de man får i svaret på et databasekall via REST-*API*’et (se kodelisting 5.5 i Polymer-delen av dette kapitlet).

Alle modellene kan også implementere *interface*’ene *Gettable* og *Postable*. Disse *interface*’ne indikerer om modellene kan henholdsvis hentes fra databasen eller poste til den.

Hvis en modell er *Gettable* og kjører sin GET-metode, genereres en forespørsel via API-klienten. Hvis forespørselen lykkes, kommer responsen tilbake i JSON-format. Vi bruker Googles Java-bibliotek GSON[75] til å konvertere JSON-strengen til en klasse på én kodelinje (kodelisting 5.20).

```
1  Experiment experiment = new Gson().fromJson(response.toString(), Experiment.class);
```

Listing 5.20: JSON-konvertering

API-klienten sender deretter et callback til modellen, som sender enda et callback til *controller/presenter* (fragmentet).

View: Layoutfiler

Alt det som er synlig for brukeren er representert som *view*. Dette er alle ressursfiler, som layout (xml), ikoner, bilder osv.

Presenter: Fragmenter

Activity- og fragmentfilene har rollen som *presenter*. Dette er laget som sørger for kommunikasjonen mellom *view* og modell.

Siden vi har satt opp applikasjonen som en SPA (*single page application*, har vi en hovedaktivitet som alltid er aktiv, som oppdaterer informasjon i topptekstfeltet og lytter etter klikk i navigasjonsmenyen, og styrer hvilket fragment som til enhver tid er aktivt. Hvert fragment er knyttet til en layout-fil, som sammen viser alt innhold i applikasjonen til brukeren.

```
1  public void replaceFragment(Class<?> actClass, int id) {
2      try {
3          mFragment = (Fragment) actClass.newInstance();
4      } catch (Exception e) {
5
6          Log.e(TAG, "Error", e);
7      }
8
9      Bundle args = new Bundle();
10     args.putInt("id", id);
11     mFragment.setArguments(args);
12
13     FragmentManager fragmentManager = getSupportFragmentManager();
14     fragmentManager.beginTransaction().replace( R.id.content_home_screen, mFragment).
       ↪  addToBackStack("tag").commit();
15 }
```

Listing 5.21: Fragmentstyring

For å sette et spesifikt fragment aktivt, f.eks. ved at noen velger et navigasjonspunkt i menyen, brukeren er ferdig med en quiz og skal sendes til resultatsiden o.l., har vi lagd en generell funksjon i hovedakvititeten (se kodelisting 5.21).

Denne funksjonen har to parametere, klassen til det fragmentet som skal settes inn, og id.

Først lages en ny instans av valgt fragmentklasse, deretter settes id som et argument slik at fragmentet kan hente ut dette ved behov (det kan f.eks. være quiz- eller eksperimentID, som fragmentet må vite for å hente ned riktig data fra databasen). Til slutt byttes det nåværende fragmentet ut med den nye instansen ved hjelp av *FragmentManager*, og det gamle legges til i *stack*'en (slik at en bruker kan komme tilbake til dette fragmentet ved å trykke på tilbakeknappen).

5.6 Beacons

Som diskutert tidligere i rapporten vurderte vi flere ulike måter å scanne eksperimentstasjoner på(dypere diskusjon rundt dette i seksjon 4.6). Selv om vi valgte QR-skanning for identifikasjon av eksperimenter ville gjerne Vitensenteret at vi skulle se nærmere på *beacons* og ideer rundt hvordan dette kunne bli brukt i senteret.

Vi foreslo å bruke *beacons* til å gi besøkende en velkomstmelding ved ankomst samt spesifikke meldinger når de beveger seg rundt i senteret. Disse meldingene skulle kunne endres av Vitensenteret gjennom webapplikasjonen. Ettersom dette var et ønske som oppsto sent i prosessen fikk vi ikke tid til å implementere alle ønskene, men vi fikk implementert mobilsiden av løsningen. Under følger en beskrivelse av implementasjonen.

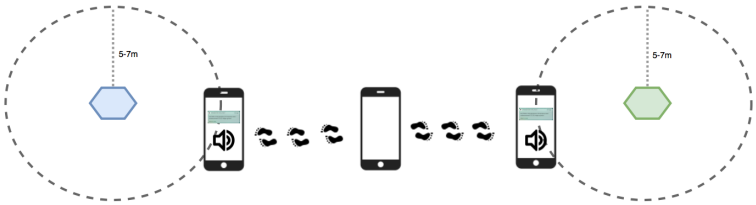
5.6.1 Teknologi

Som forklart i seksjon 4.6 er *beacons* små enheter som ved hjelp av bluetooth-signaler sender ut "her er jeg"-meldinger. Disse meldingene kan brukes til å identifisere hvilken *beacon* en mobil er i nærheten av.

Det finnes mange leverandører av *beacons*, men vi fikk anbefalt av studiekamerater å se nærmere på beacons fra Estimote. Vi valgte i samarbeid med Vitensenteret å bestille tre beacons fra Estimote.

De forskjellige beacons'ene kan stilles inn til å sende ut signaler av en gitt styrke - hvor nære mobilen må være før den oppdager signalet. Vi valgte å stille inn på *low*, noe som betyr at mobilen vil oppdage beacon'et fra en avstand på ca 5 - 7 meter.

Én beacon ble plassert ved inngangspartiet og denne skulle sende ut en melding som ønsket besøkende velkommen. I tillegg skulle det være en ved planetariet som minnet besøkende på når det er planetearievisning.



Figur 23: Beacons illustrasjon (laget i draw.io)

Meldingene fra beacons popper opp på telefonen som push-varsler når den besøkende kommer innenfor dens radius. Det er verdt å merke seg at brukeren må tillatte varsler fra beacons, man kan reservere seg ved installasjon av appen men kan også endre senere gjennom innstillinger.

Til selve implementasjonen brukte vi Estimote sitt SDK. Dette gjorde at vi kunne registrere varslene selv uten at appen kjører på telefonen. I listing 5.22 og 5.23 kan vi se hvordan funksjonene som lytter til om mobilen har *entered region*.

```
1  func beaconManager(_ manager: Any, didEnter region: CLBeaconRegion) {
2    print("User did enter " + region.identifier)
3
4    let dateFormatter = DateFormatter()
5    dateFormatter.dateFormat = "yyyy-MM-dd"
6
7    switch region.identifier {
8    case Constants.KEY_USER_ENTERED_CENTER_BEACON:
9
10     let yourDate = UserDefaults.standard.object(forKey: Constants.KEY_USER_ENTERED_CENTER_BEACON) as? Date
11
12     // If user did not yet recieve welcome message this day.
13     if yourDate == nil || dateFormatter.string(from: yourDate!) != dateFormatter.string(from: Date()) {
14
15       sendOutNotification(message: NSLocalizedString("notification-beacon-text", comment:""),
16                           title: NSLocalizedString("notification-beacon-title", comment:""))
17
18       UserDefaults.standard.set(Date(), forKey: Constants.KEY_USER_ENTERED_CENTER_BEACON)
19     }
20     ...
```

Listing 5.22: Funksjon som setter opp beacons-monitorering

Testing ble gjennomført av utviklerne ved at vi plasserte *beacons* rundt i huset til et av gruppemedlemmene, og gikk rundt med mobilen. Under denne testingen oppdaget vi at når vi gikk inn og ut av sonene til *beacons*'ene flere ganger, fikk vi varsler hver gang. Dette er ikke ønskelig, da Vitensenteret

```
@Override
public void onEnteredRegion(Region region, List<Beacon> list) {
    Calendar calendar = Calendar.getInstance();
    SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd-MMM-yyyy", Locale.getDefault());
    String currentDate = simpleDateFormat.format(calendar.getTime());
    String oldDate;
    String identifier = region.getIdentifier();
    SharedPreferences storedDates = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());

    if (identifier.equals(ENTER_CENTER)) {
        oldDate = storedDates.getString(DATE_ENTER_CENTER, null);
        if (oldDate == null || !oldDate.equals(currentDate)) {
            showNotification(getString(R.string.text_welcome_message_title),
 ↪  getString(R.string.text_welcome_message));
        }
        storedDates.edit().putString(DATE_ENTER_CENTER, currentDate).apply();
```

Listing 5.23: Callback-funksjon for beacons

ønsket at meldingene bare skal dukke opp ved ankomst. Derfor måtte vi legge inn en sjekk på om brukeren allerede har motatt et varsel samme dag. Forskjellige løsninger ble utprøvd, som å legge dette i databasen, men vi kom frem til at dette kunne lagres lokalt på telefonen. Løsningen ble at datoen for når varselet ble sent lagres på telefonen, og dette sjekkes hver gang den ønsker å sende ut et nytt varsel. Er det allerede sendt ut et varsel samme dag, vil det ikke sendes ut et til.

Både vi og Vitesenteret var veldig fornøyde med *beacons* og mulighetene dette ga. Det fungerte bra og ga mobilapplikasjonene en ekstra dimensjon. Vi anbefaler at dette utvikles videre og at det implementeres en måte å styre når meldinger sendes ut og hva som står i de fra webapplikasjonen da dette var noe vi ikke fikk tid til å implementere. Derimot er alt av mobilutvikling ferdig, så hvis en bruker installerer applikasjonen vil den motta en velkomstmelding, og en melding når den går forbi planetariet.

6 Brukergrensesnitt

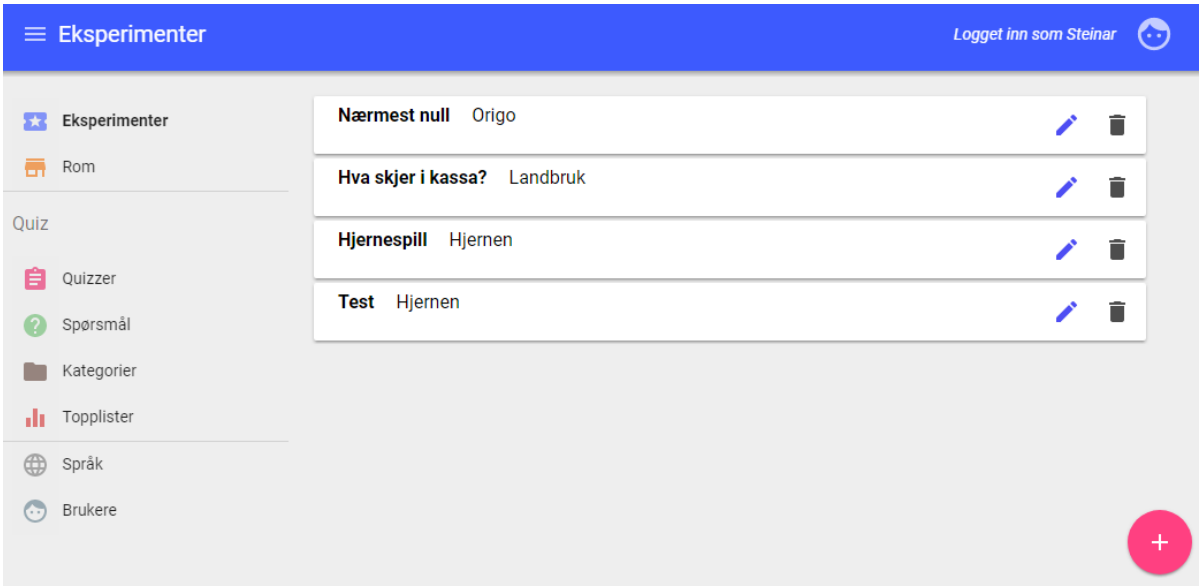
6.1 Frontend

Administrasjonsverktøyet har naturligvis kun de ansatte på Vitensenteret og eventuelle systemadministratorer som brukergruppe, noe som gjorde at brukervennlighet og funksjonalitet ble prioritert foran design. Gjenkjennbarhet fra andre systemer skal bidra til at det er enkelt å forstå hvordan man finner den informasjonen man er ute etter, eller hvordan man skal utføre en endring.

Vi har brukt Google Polymer som rammeverk (se mer om dette i seksjon 5.2), og siden mange av webkomponentene i Polymer er basert på Googles egne design-prinsipper, *Material Design*, valgte vi å bygge videre på det. Ved valg av grunnleggende struktur og oppbygning av grensesnittet, så vi til mange av Googles egne websider, som YouTube, Drive, News, Gmail, samt andre sider[76] basert på *Material Design*. Vi så etter en side med det å presentere informasjon på en enkel og ryddig måte som hovedfokus, og mener det beste alternativet var *Google Inbox*[77]. Dette er Gmail revitalisert med *Material Design*, og oppfyller kravene om brukervennlighet og gjenkjennbarhet. Vi har spesielt latt oss inspirere av Inbox skifte av farge på toppfelt ut i fra valgt navigasjonselement, lister der enkeltlinjer kan klikkes på for å utvide med mer informasjon og en *floating action button*. Se de neste delseksjonene for mer informasjon om hvordan vi har brukt dette i vår applikasjon.

6.1.1 Layout

Grunnelementene i layouten vår er navigasjonsmenyen på venstre side, toppfeltet med sidetittel og innloggingsmuligheter, en *floating action button* (FAB) nederst i høyre hjørne og en liste med elementer fra databasen (se figur 24).



Figur 24: Frontendlayout

Toppfelt

Denne har en knapp for å minimere/utvide navigasjonsmenyen, hvilken kategori siden man er inne på tilhører ("Eksperimenter" kan være både siden med liste, redigering av et enkelt eksperiment, eller

detaljer om et nytt), og en knapp helt til høyre for en liten nedtrekksmeny med inn/ut-logging.

Siden hver side har et ganske likt oppsett, og derfor lett kan forveksles, har hver kategori fått en unik farge. Toppfeltet skifter farge etter aktiv kategori, slik at brukeren får enda et holdepunkt for intuitivt å vite hvilken side han er inne på.

Navigasjonsmeny

Hvert element i navigasjonsmenyen representerer en kategori, og er lenket til listevisningen av alle elementene i den kategorien. Hver kategori har et unikt ikon med en unik farge, for å øke gjenkjennbarheten.

Floating Action Button

FAB'en nede i høyre hjørne åpner en meny, med alternativer for å lage nye elementer innenfor hver kategori. Hver kategori har en merkelapp med navn ved siden av samme ikon og samme farge som i navigasjonsmenyen (se figur 25).

Liste med elementer

Listen er alle elementer funnet i databasen innenfor valgt kategori, og hvert element har en knapp for å redigere og å slette. Ved å klikke på et av elementene, utvides hele linjen vertikalt for å vise eventuell informasjon som er registrert. Ved å f.eks. trykke på et eksperiment i en liste, kommer QR-kode, bilde, beskrivelse, hint og ekstrafunksjonalitet opp (så lenge det er registrert på akkurat det eksperimentet).

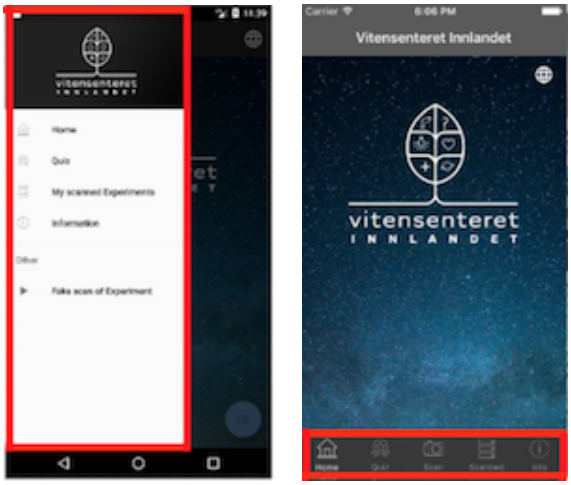
Figur 25: FAB-meny

6.2 Mobilapplikasjonene

Denne seksjonen beskriver ulike aspekter knyttet til applikasjonenes brukergrensesnitt - generelt, hva som skiller designet i applikasjonene, hva som er likt samt hensyn til universell utforming.

Universell utforming er svært viktig for Vitensenteret og dette prosjektet, av mange årsaker. Som hovedregel skal alle mobilapplikasjoner som er utgitt i Norge og som har personer i Norge som målgruppe være universelt utformet[78].

For Vitensenteret er det viktig at mobilapplikasjonene er universelt utformet fordi de mottar statlige midler og fordi Vitensenteret skal være et inkluderende sted å være. Aldersgruppen til mobilapplikasjonenes målgruppe er en alder hvor det er ekstra viktig å føle seg inkludert, at alle kan bruke mobilapplikasjonene uansett funksjonsevne er derfor svært viktig.



(a) Navigation drawer Android (b) Tab Bar Controller iOS

Figur 26: Eksempler på navigasjonsmekanismer

6.2.1 Generelt

Android-applikasjonen sitt design er basert på prinsippene i *Material Design for Android*[42]. iOS-applikasjonen sitt design er basert på Apple sine egne designprinsipper og standardkomponenter.

Apple og Google søker med sine retningslinjer for grensesnitt å skape en standard for hvordan applikasjon bør se ut på respektive plattformer. Hvis brukeren gjenkjenner grensesnitt og navigasjonsmåte så går det raskere å bli vant med nye applikasjoner[43].

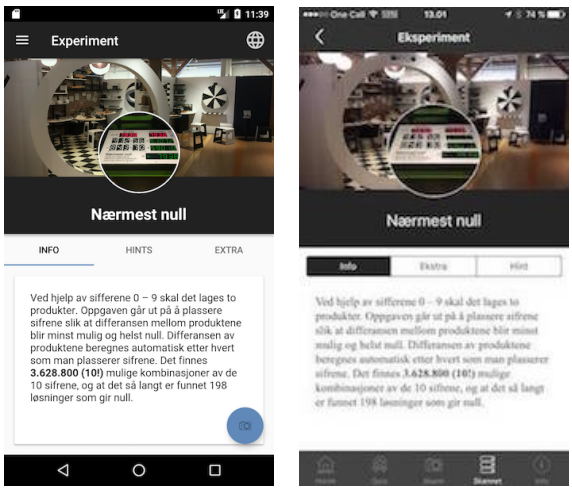
Helt tilbake til 2015 så var veldig mange av Google sine egne applikasjoner for Android designet med Material Design - Google Drive, Gmail, YouTube, Google Maps og Google Innboks[79]. Fra Android

sitt UX team er det anbefalt å bruke Material Design til Android-applikasjoner[80]. Vitensenteret synes at kombinasjonen av argumentasjonen om gjenkjennbarhet og Android sine egne anbefalinger var gode nok grunner til å velge Material Design.

6.2.2 Hva skiller de?

En av de største forskjellene mellom applikasjonenes grensesnitt er navigasjonmåten. På Android har man en *Navigation drawer*, mens på iPhone har man en *Tab Bar*. Dette er illustrert i figur 26. Figur a illustrerer Android, Figur b illustrerer iOS.

Figur 27 illustrerer forskjellen mellom applikasjonene på eksperimentsiden, den illustrerer også hvor knappen for bruk av QR-skanneren ligger på respektive applikasjoner. Figur a er Android-applikasjonen, Figur b er iOS-applikasjonen.



(a) Eksperimentside Android (b) Eksperimentside iOS

Figur 27: Eksperimentsider

På Android-applikasjonen er det brukt et *card* med bakgrunnskygge hvor informasjonsteksten ligger[81]. Teksten på iOS har ingen *card*. Navigasjonen på eksperimentsiden iOS gjøres gjennom en *segmented controller* - en av standardkomponentene som er mye brukt i iOS-applikasjoner[82]. Navigasjonen på eksperimentsiden til Android er en *view pager*[83].

Knappen for tilgang til QR-skanneren ligger på Android nederst til høyre som en *Floating Action Button*, tilsvarende på iOS ligger som en egen tab i midten av tab baren.

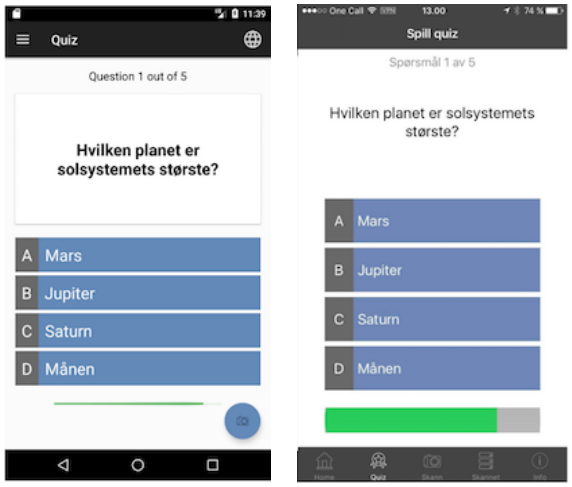
6.2.3 Hva er likt?

Informasjonssiden, tidligere skannede eksperimenter og quiz er veldig like. Illustrert i henholdsvis figur 28 og figur 29.

6.2.4 Universell utforming

I forprosjektet i Objekt-orientert systemutvikling mappe 4(se vedlegg 11.5) gjorde vi en vurdering av hensyn vi måtte ta i forhold til universell utforming av applikasjonen. Følgende subseksjon er en oppsummering og videreføring av denne vurderingen.

Applikasjonen må designes for personer med synsutfordringer, fargeblindhet, hørselsproblemer, motoriske utfordringer og turister.



(a) Quiz Android (b) Quiz iOS

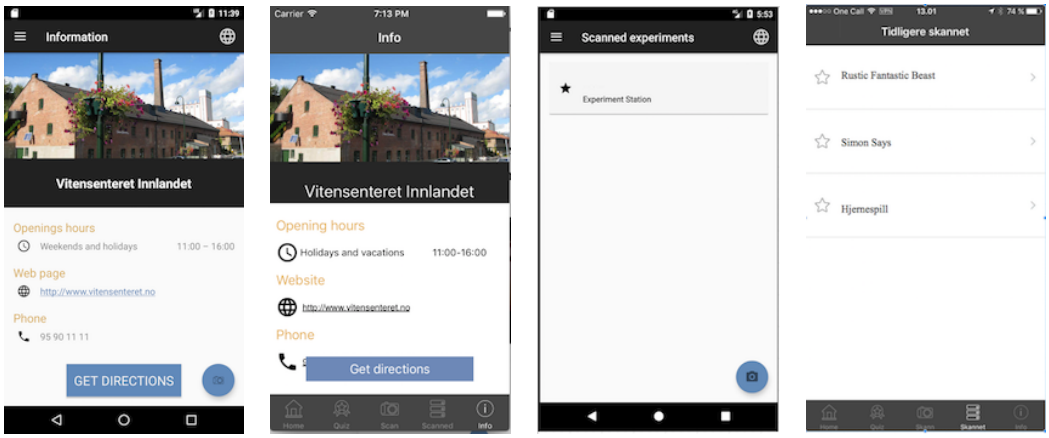
Figur 28: Quizsider

Språk

En potensiell brukergruppe er turister og andre som ikke kan norsk. Applikasjonen vil bli internasjonalisert slik at det er mulig å bruke den på flere språk. Gjennom administrasjongrensesnittet skal det være mulig å legge til nytt språk for innhold. Knapper o.l på applikasjonen følger språket på telefonen, i første omgang kun norsk og engelsk.

Synsutfordringer

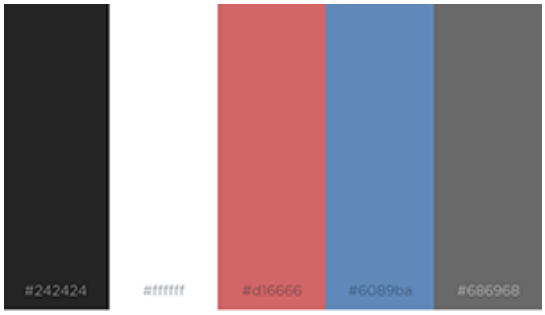
En viktig ting å ta hensyn til er svaksynthet. Applikasjonene er kompatible med innebygde funksjoner for svaksynthet som forstørrer tekst. I tillegg er applikasjonen kompatibel med funksjonalitet i både iOS og Android som leser opp informasjonen som står på skjermen.



(a) Informasjonsside Android (b) Informasjonsside iOS (c) Tidligere skannet Android (d) Tidligere skannet iOS

Figur 29: Informasjonssider og tidligere skannet sider

Fargeblindhet

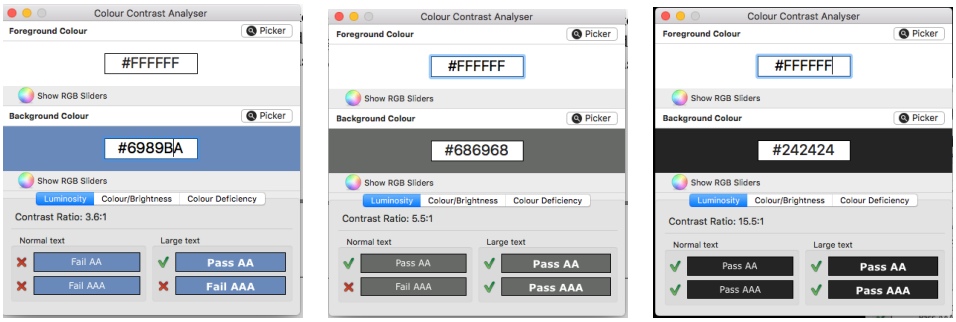


Figur 30: Fargepaljett, laget med verktøyet coolors.co[3]

Vitensenteret ville at fargene på applikasjonen skulle være så like de offisielle fargene som mulig uten å diskriminere mennesker som kan ha problemer med å skille ulike farger.

Fargepaljetten vi har brukt i applikasjonen er illustrert i figur 30.

Nyanser av blå, svart og hvit er hovedfargene. For nettløsninger og mobilapplikasjoner i Norge er det ulike krav til universell utforming, et av kravene er at fargekontraster i applikasjonen minst skal ligge på Nivå A definert av WCAG 2.0-standarden[84]. Figur 28 viser quiz-siden på applikasjonen og figur 31 viser test av fargekontraster på quiz siden. Vi har testet kontraster for den svarte/hvite kontrasten som går igjen i applikasjonen samt kontraster på knappene. Som man ser i figur 31 figur b og c så passerer kontrasttesten på nivå AA både for svart/hvit og grå/hvit, ser man videre på figur a blå/hvit ser man at testen kun passerer på nivå AA med stor skrift. Vi regner skriften på knappene som stor, og mener dermed at dette er tilstrekkelig. Minstekravet er nivå A, vi har lagt oss på nivå AA.



(a) Fargekontrast blå/hvit (b) Fargekontrast grå/hvit (c) Fargekontrast svart/hvit

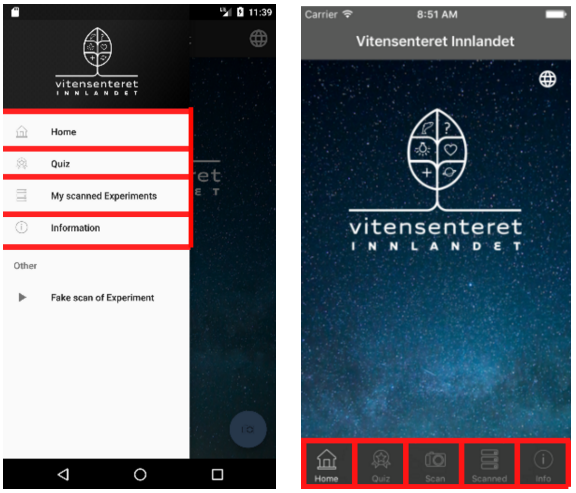
Figur 31: Tester fargekontrast

Motoriske problemer

Brukere med motoriske problemer kan trenge spesialtilpasninger av applikasjonen for en god opplevelse. Knapper og menyer er designet med dette i tankene med store knapper og styreflater. Figur 32 illustrerer størrelsen på styreflater i applikasjonen. Figur a er Android, figur b er iOS.

Hørselsproblemer

Brukere med hørselsproblemer kan trenge spesialtilpasninger, vi har ingen lyd i applikasjonen og trenger derfor ikke å ta noen spesielle hensyn her. Ved videreutvikling og eventuell bruk av lyd måtte hensyn til hørselsproblemer blitt utredet.



(a) Knappestørrelse Android (b) Knappestørelse iOS

Figur 32: Illustrasjon av knappestørrelser

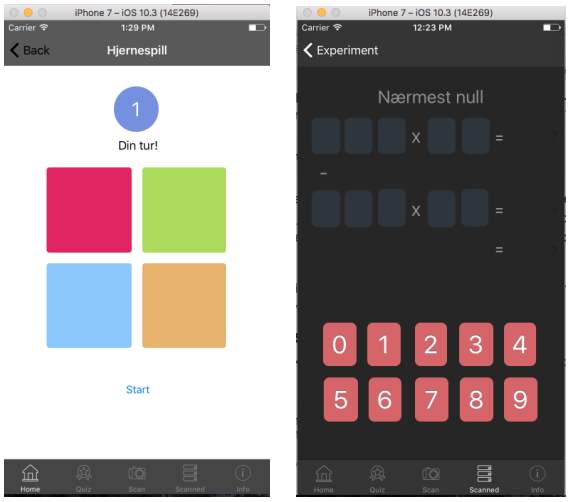
6.2.5 Spillenes grensesnitt

Som nevnt i seksjon 1.6 så er det blitt utviklet to digitale implementasjoner av eksperimenter på Vitensenteret, nærmere bestemt to spill kalt *Hjernespill* og *Nærmest null* illustrert i figur 33.

Hjernespill er et hukommelsesspill hvor det er fire knapper med forskjellige farger som lyser i en bestemt rekkefølge, brukeren skal så gjenta denne sekvensen. Hver gang brukeren gjentar riktig øker antallet i rekkefølgen med en. Spillet fortsetter så lenge brukeren gjentar riktig rekkefølge.

Nærmest null er et mattespill hvor man plasserer ut tall fra 0 - 9, tallene multipliseres og subtraheres og målet er å plassere tallene slik at resultatet blir lik null.

Spillene er representative for eksperimenter på Vitensenteret da de er eksakte speilinger av reelle spill lokalisert på senteret.



(a) Hjernespill (b) Nærmest null spill

Figur 33: Mobilspill

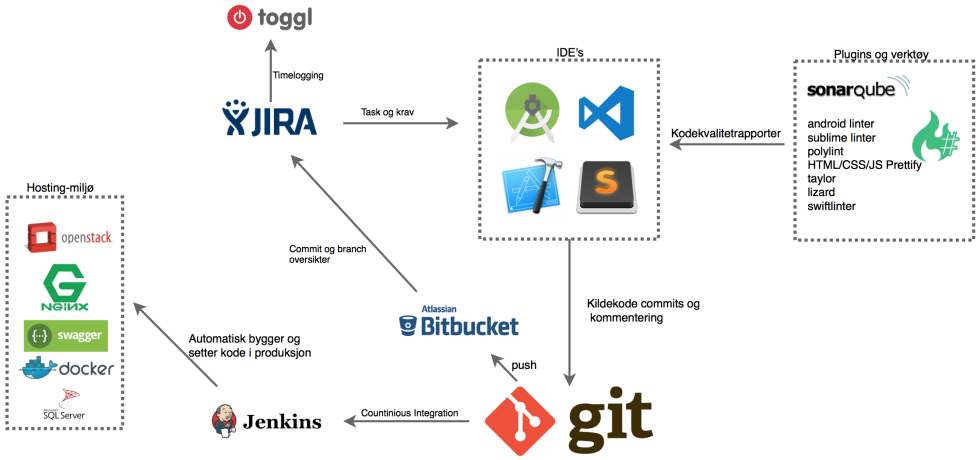
6.2.6 GUI workshop

I sprint 4 hadde vi GUI workshop der gruppedeltakerne brukte en hel dag på å utarbeide hvordan brukergrensesnittet skulle se ut både på iOS og Android. Vi lagde mockups av de ulike skjermbildene med tilhørende fargepaljett for farger vi skulle bruke.

Mockups av grensesnitt er grove skisser av hvordan designet skal se ut. Se appendix I for illustrasjon av alle mockups.

7 Utviklingsmiljøer

I dette kapittelet skal vi se på hvordan vi satte opp utviklingsmiljøene. Siden vi hadde begrenset tid til koding under selve bacheloroppgaven, så var det viktig å gjøre prosessen så effektivt som mulig. Derfor ønsket vi å sette opp profesjonelle miljøer som gjør at utrulling av ny kode til de andre systemene skulle gå raskt. Parallelt med bacheloroppgaven, hadde vi også et emne kalt Professional Programming som fokuserte på hvordan et utviklingsteam kan jobbe profesjonelt. Dette emnet bidro til at vi hadde fokus på hvordan vi kunne sette opp et effektivt utviklingsmiljø. Siden dette var noe vi ikke hadde drevet med mye før, lærte vi masse underveis i prosessen. I figur 34 ser vi en oversikt over alle verktøyene vi brukte og hvor i prosessen disse ble brukt. I dette kapitlet vil vi fokusere på *backend*, og de automatiske løsningene vi etterhvert implementerte her. Vi vil ikke gå inn på mobilmiljøet, da dette mer var et standardoppsett. I seksjon 8 går vi nærmere inn på hvilke verktøy som ble brukt for å sikre kodekvalitet.



Figur 34: Oversikt over kodeverktøy (laget i draw.io)

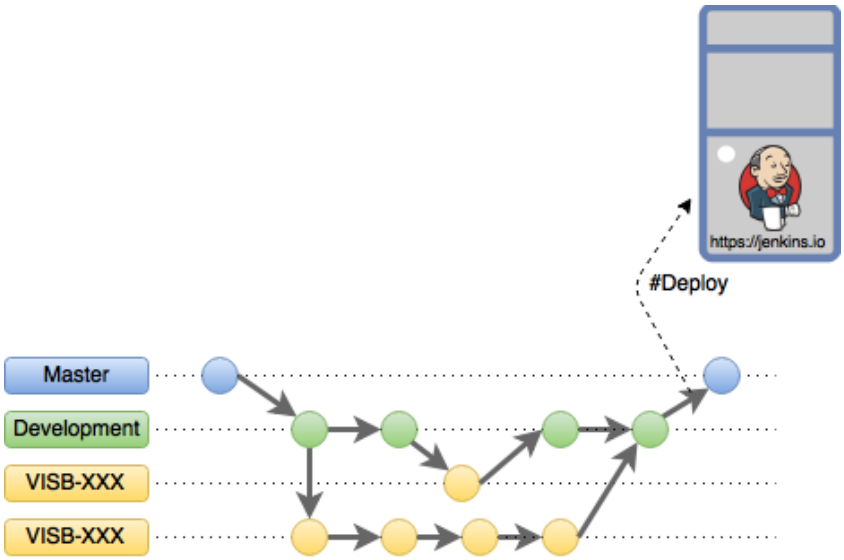
7.1 Backend

Siden vi jobbet iterativt med sprinter på to uker, var det gunstig å ta for seg en og en funksjonalitet. Dette resulterte i at vi f.eks. i en sprint bestemte oss for å implementere ferdig eksperimentsiden på mobil og web. Siden både mobil og webapplikasjonen skulle kommunisere med *backend* måtte API'et prøve å ligge litt foran på hvilke *endpoint*'s som ble utviklet. Slik sikret vi oss at mot slutten av sprinten, når web og mobil skulle implementere kommunikasjonen mot det reelle API'et kunne de teste kommunikasjonen mot utviklingsserveren. Vi hadde på forhånd blitt enige om hvordan vi ønsket at JSON-dataene skulle se ut. Dette ble en slags kontrakt mellom *frontend* og *backend*, og kommunikasjonen kunne *mockes* helt til *backend* fikk implementert funksjonaliteten.

I den første sprinten satte vi opp en enkel server, hvor vi manuelt bygget *backend* hver gang vi hadde en endring. Vi så raskt at dette stjal mye tid. Det ble ofte oppdaget *bugs* og forbedringer som måtte utbedres, og raskt rulles ut i utviklingsmiljøet slik at de andre kunne kode mot den nyeste versjonen av API'et. Derfor bestemte vi oss for å sette opp et CI (*continuous integration*) system som kunne bygge dette automatisk hver gang en ny endring ble lastet opp til repoet. Til å utføre denne byggingen benyttet vi en *open source* løsning kalt Jenkins.

Jenkins

Jenkins er et kryssplattform *open source* byggeverktøy[85]. Her kan vi sette prosjekter til å bygge automatisk når fastsatte kriterier er oppfylt. Jenkins varsler også om bygg feiler og/eller at systemet er ustabilt.



Figur 35: Illustrasjon over Jenkins arbeidsflyt (laget i draw.io)

Vi ønsket å følge vår normale arbeidsflyt, ved at vi koder i *branchen development* og *brancher* ut i egne *brancher* med JIRA-tag før det merges inn i *development* igjen. Når vi ønsker å rulle ut en ny versjon til serveren *merges* endringene inn i *branchen master*. Et slik oppsett er fullt mulig i Jenkins. Vanligvis settes det opp en *git-hook* i Bitbucket, slik at hver gang det pushes nye endringer, sender denne et signal til Jenkinsserveren om at nå er det på tide å bygge systemet på nytt.

Dette kunne ikke vi benytte oss av, da vår utviklingserver sto bak skolens brannmur, noe som gjorde at den ikke var tilgjengelig utenfor skolens nett. Dermed ville aldri Bitbucket kunne nå serveren vår.

Vi løste dette ved å sette opp et *git-hook* lokalt på maskinen. *Git-hook* er et script som skal kjøres på forskjellige stadier i *git-flyten*. I *.git*-mappen ligger det en mappe som heter *hooks*, hvor vi kan legge slike script. Vi benyttet oss av *hooket* kalt *pre-push*, som kjøres rett før en *push*. Scriptet (listing 7.24) sjekker hvilken melding som er lagt ved den siste *commit*'en, noe vi kan se på figur 35. Hvis denne meldingen inneholder frasen "#Deploy" kjøres et kall opp mot serveren som forteller Jenkins at byggingen skal starte.

Når serveren får beskjed om å starte et bygg, kjører Jenkins et script som setter i gang byggingen av API'et. Siden dette er en utviklingserver bruker vi bare .NET sine byggeverktøy, skulle dette produksjonsettes bør andre byggemetoder vurderes, noe vi skriver mer om i kapittel 10.

I scriptet (listing 7.25) avsluttes først alle prosessene relatert til API'et som kjøres. Så bygges det en ny database (dette er kommentert ut, da vi ikke ønsket å renske databasen for hvert bygg utover i prosjektet). Vi bruker programmet Supervisor[86] til å kjøre serveren, dette programmet kjører serveren som en bakgrunnsprosess, men tar vare på alle logger - pluss at man kan se *livefeed* av terminalen hvor det kjøres - som gjør det enkelt å hoppe inn i prosessen og se feilmeldinger. Det er så lagt inn en *sleep*, som gjør at serveren venter 3 minutter før den bygger videre. Siden vi kjører et *pre-push* script lokalt før selve pushen er gjennomført, kan vi være uheldige ved at byggingen starter før hele *push*'en er ferdig. *Sleep*'en sørger for at Jenkins venter så lenge at vi kan være sikre på å ha siste versjon.

Det gjøres så klart for at byggingen skal starte. Etter byggingen er fullført, starter Supervisor prosessen igjen og serveren skal fungere som normalt. Hvis byggingen skulle mislykkes kan Jenkins varsle på

```
#!/bin/bash

master_branch="master"
current_branch=$(git rev-parse --abbrev-ref HEAD)

[ "master" != $(git rev-parse --abbrev-ref HEAD) ] && exit 0

commit_regex=".*#Deploy.*"
message="$(git log -1 --pretty=%B)"

shopt -s nocasematch

if  ! [[ $message =~ $commit_regex ]] ; then exit 0 ; fi

echo "Commit message includes #Deploy, build instruction is getting sent to JENKINS..."

# Everything under here will run if commit message match.
curl -u bruker:password 128.39.141.233:19111/job/viten-i-senter-backend/build?token=thescrumdolls
```

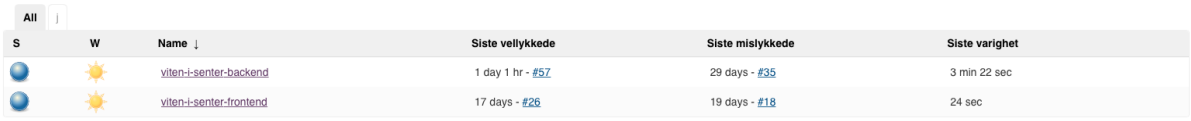
Listing 7.24: Pre-push script som iverksetter bygging

```
# Kill processes using files
sudo killall dotnet || true

# Stop and clean up old docker files
# docker stop $(docker ps -a -q) || true
# docker rm $(docker ps -a -q) || true
# docker rmi $(docker images -f "dangling=true" -q) || true
# docker volume rm $(docker volume ls -qf dangling=true) || true

# Run SQL server
# docker run -e 'ACCEPT_EULA=Y' -e 'SA_PASSWORD=<YourStrong!Passw0rd>' -p 1433:1433 -d
 ↪  microsoft/mssql-server-linux

# Wait to make sure push has finished
sleep 3m

# Stop current server from running, due to block on files
sudo service supervisor stop

cd src/BackendAPI

# Image folder needs to be created, or else it won't work with image endponits
mkdir -p wwwroot/images

# Build project
sudo dotnet restore --configfile Nuget.Config
sudo dotnet publish --framework netcoreapp1.1 --runtime "ubuntu.16.04-x64" --output /home/ubuntu/dotnet

# Start server
sudo service supervisor start
```

Listing 7.25: Byggescriptet som kjøres på CI-server

mail, og det vil synes på statussiden.

Figur 36 viser hvordan Jenkins viser oss at begge byggene var velykkede, og sola indikerer at de siste byggene har vært stabile uten feil. Jo dårligere værtegn jo fler ustabile bygg den siste perioden.



Figur 36: Jenkins statusoversikt

Docker

Som vi kan se i listing 7.25 bruker vi Docker for å kjøre databaseserveren. Docker er et containersystem som pakker og bygger en service til en selvstendig container[87]. Denne har mye av de samme fordelene som en virtuell maskin med ressursisolasjon, applikasjonsseparasjon og separat minneallokering. Fordelen ved å kjøre databasen i en Docker container er at den kjører i sitt eget private miljø, selvom vi bruker bare en virtuell maskin til å kjøre flere systemer. Det er også lett å bygge systemet om igjen, da skrapes hele containeren og det lages en ny.

```
1 docker run -e 'ACCEPT_EULA=Y' -e 'SA_PASSWORD=<PASSORD>' -p 1433:1433 -d microsoft/mssql-server-linux
```

Listing 7.26: Docker kommando

Kommandoen i listing 7.26 starter opp en container med en Microsoft SQL Server[88]. Microsoft lanserte akkurat sin SQL-server til kryssplattform. Dette er en god løsning for oss da vi ønsker å bevare mest mulig valgfrihet for senere produksjonsetting. Ved å bruke en Docker container, kan det senere avgjøres om produksjonserveren skal være Windows eller Linux. Vi setter opp Docker slik Microsoft anbefaler[89]. For å starte serveren kjøres kommandoen i listing 7.26.

For å koble til denne containeren definerte vi en *connection_string* i backendsystemet, som forteller at dette er databasen vi ønsker å benytte oss av. Definisjonen av denne strengen ser vi i listing 7.27.

```
1 "ConnectionStrings": {
2     "DefaultConnection":
  ↪   "Server=127.0.0.1;UserId=SA;Password=<PASSORD>;Database=ApplicationDb;Trusted_Connection=False;"
3   }
```

Listing 7.27: Tilkoblingsstreng til databasen

I kapittel 10 forklarer vi mer hvorfor vi anbefaler at en eventuell produksjonsetting av denne applikasjonen kjører både API'et og databasen som Dockercontainere.

7.1.1 Statusside

Etter Jenkins var satt opp til å automatisere bygging sparte vi mye tid fra det ble oppdaget en *bug* eller et ønske om en forbedring, til dette var rullet ut til utviklingsserveren. Likevel hendte det fra tid til annen, at når API'et ble gjort tilgjengelig for frontend og mobil oppsto det feil som ikke ble oppdaget under testingen. Av og til kunne API'et også settes helt ut av funksjon - av forskjellige årsaker som ofte er barnesykdommer tidlig i en prosess. Selvom dette skjedde veldig sjelden, hadde vi noen episoder der en eller flere av utviklerne trodde de hadde en *bug* hos seg, da det viste seg at det var serveren som ikke responderte eller returnerte feil. I starten ble dette løst ved kommunikasjon mellom gruppemedlemmene, men ønsket om en enkel måte å sjekke status for API'et og eventuelt kunne enkelt rense databasen meldte seg.

Vi bestemte oss da for å utvikle en webside som viste statusen til de forskjellige komponentene i systemet. Det gir også muligheten for alle til å bygge systemene på nytt. Dette løser ofte problemer som har blitt skapt av for eksempel korrupte data, eller det kan være et ønske fra noen av utviklerne om å få rensket databasen. Selve statussiden ble laget i grunnleggende JavaScript, som prøver forskjellige kall på API'et og viser en status ut i fra responsen som returneres. Denne indikerer om det er noe feil med API'et.

I Figur 37 ser vi at siden viser statusene på det siste bygget Jenkins har gjennomført, og skulle det være noen problemer med at systemet ikke ble bygget korrekt vil dette vises her. Under vises et bilde som sier noe om status på API'ets respons.

Nederst kan utviklere iverksette en ombygging av API'et eller frontend, som også ble lagt til statussiden etterhvert. Dette er noe som gjorde det mulig for alle, uten dypere kjennskap til hvordan Jenkins var satt opp, kunne bygge systemene på nytt.

Figur 37: Statusside som viser status på de forskjellige komponentene i systemet

Statussiden ble mye brukt, og eliminerte mye kveldskommunikasjon om serverstatus. Vi er veldig glade for at vi brukte den korte tiden det tok å sette opp denne siden.

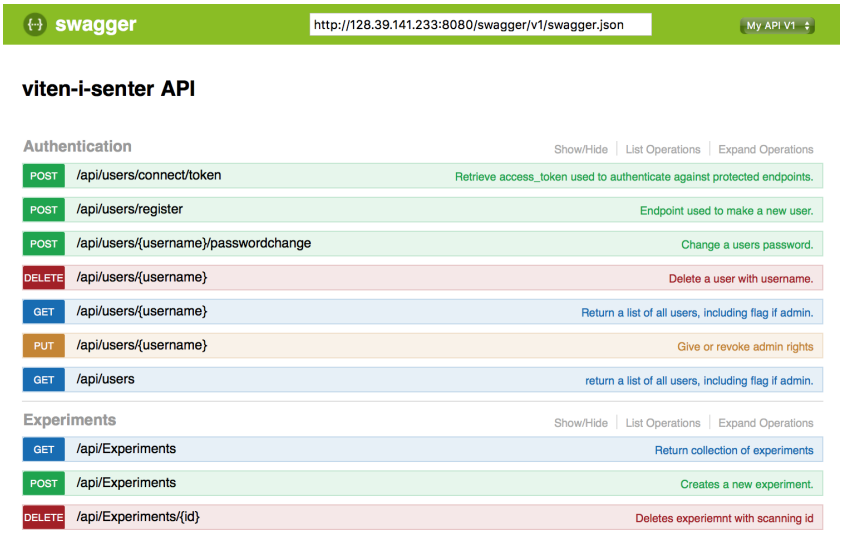
7.1.2 .NET Core RestAPI

Et API som skal brukes av andre utviklere bør dokumenteres. Dette er en tidkrevende jobb hvis det skal gjøres manuelt. Når man koder opp mot et API, ønsker også utviklere å teste funksjonalitet underveis. Dette for å få en fin oversikt over mulighetene og kunne se kravene rundt de forskjellige kallene til API'et. Dette inkluderer forskjellige attributter, *forms* eller hvilke HTTP-kall som støttes.

Vi identifiserte dette behovet helt i starten av bachelorprosjektet, mye på grunn av tidligere erfaringer fra prosjekter underveis i studiet.

Et populært rammeverk når det kommer til dokumentering av API'er er Swagger[90]. Dette er også et velbrukt rammeverk i .Net Core miljøet. Swagger er et rammeverk som genererer en dynamisk nettside, med oversikt over hvert enkelt *endpoint*, hvilke HTTP-kall som støttes, og hvordan en forespørsel skal utformes. Det tilbyr også et innebygd verktøy som lar utviklere generere kall, og teste responsen. På denne måten er det enkelt å teste ett kall - for etterpå å implementere det i kode. Figur 38 viser et oversiktsbilde av dokumentasjonen til vårt API.

Swagger er avhengig av at prosjektet drar nytte av *model*- og *viewmodel*-klasser. *Viewmodels* er det som skal presenteres ut til klienten som resultatet fra et kall. For å gi Swagger en forklaring på et *endpoint*, brukes forhåndsdefinerte attributter definert av Swagger sitt rammeverk.



Figur 38: Oversiktside Swagger dokumentasjon.

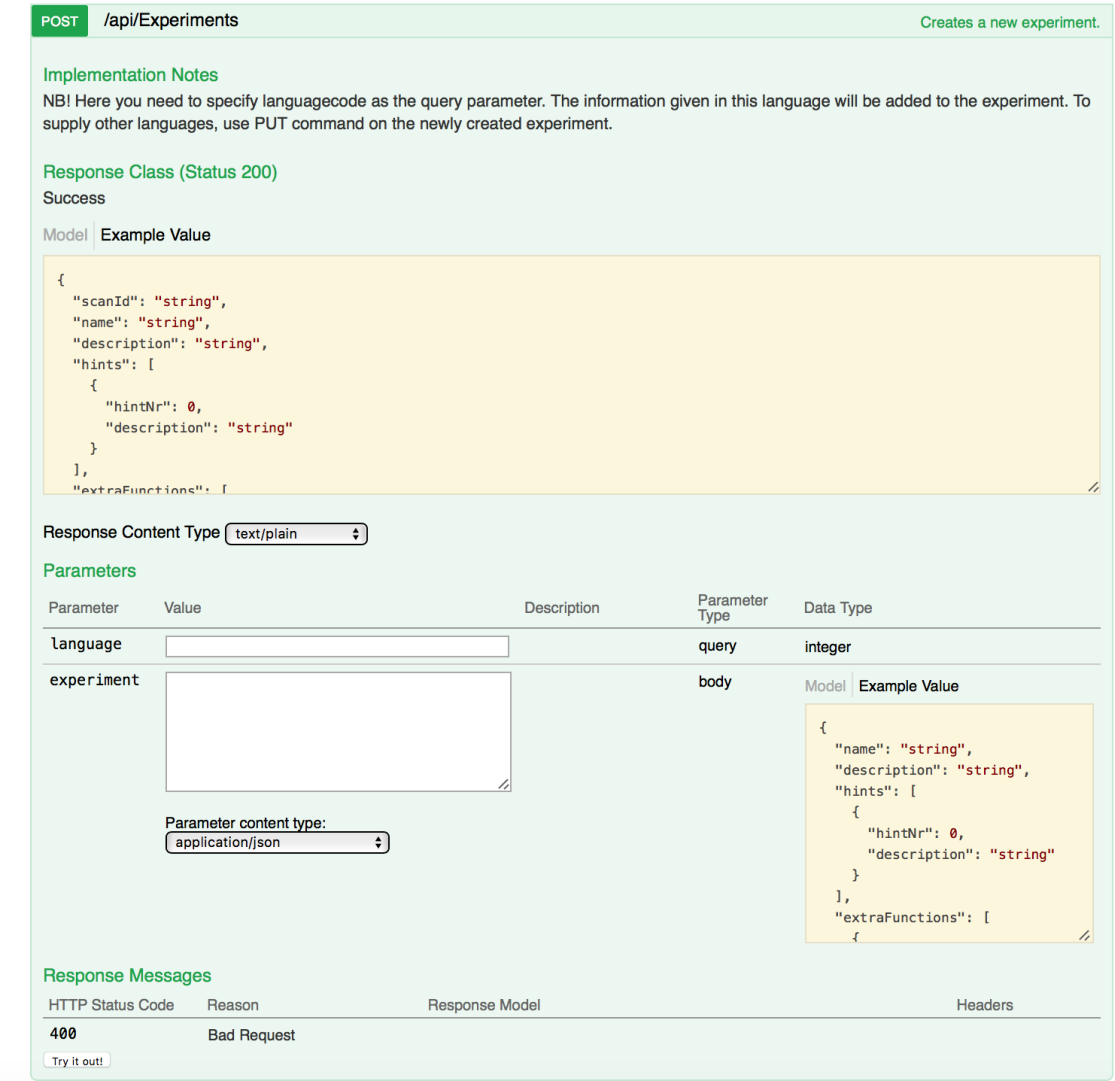
```
1  /// <summary>
2  /// Creates a new experiment.
3  /// </summary>
4  /// <remarks>
5  /// NB! Here you need to specify languagecode as the query parameter. The information given in this language
    ↪  will be added to the experiment.
6  /// To supply other languages, use PUT command on the newly created experiment.
7  /// </remarks>
8  [Authorize, HttpPost]
9  [HttpPost]
10 [Produces(typeof(ExperimentSingleLanguageViewModel))]
11 [SwaggerResponse((int)System.Net.HttpStatusCode.OK, Type = typeof(ExperimentSingleLanguageViewModel))]
12 [SwaggerResponse((int)System.Net.HttpStatusCode.BadRequest)]
13 public IActionResult Post([FromQuery] int language, [FromBody]ExperimentPostViewModel experiment)
14 {
15     ...
```

Listing 7.28: Swagger deklarasjon

I listing 7.28 ser vi et eksempel på en slik definisjon. Her forklarer vi til Swagger hvordan et *HTTP POST* kall til *endpoint*'et */api/experiment* fungerer. Kommentarene over selve funksjonen betrakter Swagger som dokumentasjon. Her vil alt innenfor *<summary></summary>* betraktes som en kort beskrivelse av funksjonalitet, mens *<remarks>* bruker for å gi utfyllende informasjon. Videre defineres de ulike HTTP-kodene som utvikleren kan forvente som svar. I dette eksempelet kan OK (kode 200) og *BadRequest* (kode 400) være mulige responskoder. Dette gir utvikleren muligheten til å legge inn feilhåndtering på feilkoder som kan forventes av API'et. Dette swagger oppsettet gir en dokumentasjonside som vi ser i Figur 39

På denne dokumentasjonsiden kan vi se et eksempel på formatering av JSON-dataene som skal sendes inn til API'et burde være formatert, og også hvordan JSON-dataene som vi får som svar er formatert. Nederst gir den også muligheten for å generere et kall til dette *endpointet*. Swagger siden generes automatisk ved hvert bygg, og den mappes opp og gis tilgang til fra url'en */swagger*.

Bruken av Swagger gjorde det mye lettere for frontend og mobil å teste API'et, samt at det ga en veldig oversiktlig presentasjon av hvilke funksjonalitet som tilbys.



Figur 39: Dokumentasjon til *endpoint*'et *api/Experiments*

7.2 Frontend

7.2.1 Polymers byggeverktøy

Byggingen av *webinterfacet* ble også utover i prosjektet lagt til som en byggejobb og kunne startes på samme måte som med bygging av API'et. Siden denne fremgangsmåten er veldig lik, går vi ikke mer inn på det her, men prosessen er den samme som i seksjon 7.1.

Som webserver for *webinterfacet* bruker vi per i dag Polymer CLI, som er Google sine verktøy for bygging, testing og hosting av Polymer. Siden dette er en utviklingserver som ikke skal kjøre i produksjon funker dette fint. Skulle vi derimot lansert dette i produksjon burde nok webapplikasjonen blitt pakket som en selvstendig applikasjon - slik at den kan ligge på en vanlig webserver. For å starte Polymer's innebygde webserver brukes kommandoen i listing 7.29 Denne kjøres på samme måte av Jenkins som API'et. Også dette bygget kan startes fra statussiden.

```
1 polymer serve --port 9000 --hostname 0.0.0.0
```

Listing 7.29: Polymer serve kommando

8 Kodekvalitet

Vi har i hele bachelorprosjektet hatt fokus på kvalitet i koden, og har gjort flere grep for å sikre dette utover i prosjektet. Målet har vært å jobbe som et profesjonelt utviklingsteam.

Siden vi har fire forskjellige kodebaser og derfor måtte jobbe mye separat med forskjellige deler av prosjektet ble vi nødt til å tenke på hvordan vi best kunne sikre god kvalitet.

8.1 Code Review

En av måtene vi sikret kvalitet var å plassere utviklingsoppgaver som var fullførte, men som utvikleren mente trengte en gjennomgang av andre i teamet i QA-kolonnen i scrumboardet vårt (se seksjon 2.3.4). Da kunne vi på slutten av uka ha en gjennomgang av de taskene som var ekstra kompliserte, eller hvor utvikleren var usikker på løsningen.

Hver fredag holdt vi en felles *code review*, hvor den enkelte utvikleren gikk igjennom deler av ukas kode på storskjerm, med spesielt fokus på taskene som var lagt i QA-kolonnen. De andre gruppemedlemmene ga tilbakemeldinger og pekte på eventuelle forbedringspotensialer som ble identifisert under disse seansene.

Dette fungerte bra på flere måter, den enkelte utvikleren fikk feedback på kodekvalitet og reflekterte også over sin egen kode på en annen måte når fremgangsmåten måtte forklares til andre. De andre gruppemedlemmene fikk også muligheten til å få et oppdatert innblikk i de andre systemene, som sikret oss ved eventuell sykdom, da det ville bli lettere å gå inn som utvikler på andre deler av systemet.

Code review med faglærer

I emnet Professional Programming tilbød faglærer Simon McCallum en felles *code review*-seanse med han, og alle deltagerne i kurset. Vi hadde en slik *review*-seanse 24. mars hvor vi presenterte *backend* og iOS koden på storskjerm. Vi fikk gode tilbakemeldinger på kode og struktur av McCallum, men også tips til forbedringer som at kommentarene helst skulle ligge på *interfacene* og ikke på selve funksjonene. At vi burde så langt det lar seg gjøre bruke konstanter og ikke magiske numre.

Det var veldig verdifullt underveis i prosjektet å få tips og feedback på systemer, og samtidig veldig moro når det også var positive tilbakemeldinger.

Code review 17/03

I denne gjennomgangen snakket vi om hvordan vi kunne refaktorere nettverkskallene på Android-applikasjonen da vi hadde en følelse at dette var komplisert løst, noe som resulterte i unødvendig mye kode. Dette var det konsensus for i gruppa, og det ble bestemt at vi skulle gjøre research om det fantes biblioteker som kunne gjøre mye av konverteringen fra JSON-data til modellklasser automatisk. Et slik bibliotek fantes for Java og heter GSON[75]. I listing 8.30 ser man hvordan funksjonen som *parset* JSON-data om til et eksperiment så ut før vi refaktorerte mens i listing 8.31 ser man hvordan resultatet ble.

```java
@Override
public void onSuccess(int statusCode, Header[] headers, JSONObject response) {
    ArrayList<Hint> hintsDecoded = new ArrayList<>();
    ArrayList<ExtraFunction> extraFunctionDecoded = new ArrayList<>();
    String roomId = null;
    try {
        JSONArray hints = response.getJSONArray("hints");
        for (int i = 0; i < hints.length(); i++) {
            JSONObject hint = (JSONObject) hints.get(i);
            hintsDecoded.add(new Hint(hint.getString("hintNr"), hint.getString("description")));
        }
    } catch (JSONException e) {
        Log.d("No hints", "Experiments has no hints");
    }
    try {
        JSONArray extraFunctions = response.getJSONArray("extraFunctions");
        for (int i = 0; i < extraFunctions.length(); i++) {
            JSONObject extraFunction = (JSONObject) extraFunctions.get(i);
            extraFunctionDecoded.add(new ExtraFunction(extraFunction.getString("name"),
            extraFunction.getString("action")));
        }
    } catch (JSONException e1) {
        Log.d("No Extra", "Experiments has no Extra functions");
    }
    try {
        roomId = response.getString("roomId");
    } catch (JSONException e) {
        e.printStackTrace();
    }
    try {

        String imageId;
        imageId = response.getString("imageId");

        Experiment experiment = new Experiment(response.getString("name"),
                response.getString("description"), response.getString("scanId"),
                imageId, hintsDecoded, extraFunctionDecoded, roomId);
        apiCallback.onSuccess(experiment);
    } catch (JSONException e) {
        Log.e("ERROR", e.getMessage());
        apiCallback.onFailure();
    }
}
```

Listing 8.30: Kode før refaktorering

```java
@Override
public void onSuccess(int statusCode, Header[] headers, JSONObject response) {
    Experiment experiment = new Gson().fromJson(response.toString(), Experiment.class);
    apiCallback.onSuccess(experiment);
}
```

Listing 8.31: Kode etter refaktorering

Som vi kan se ble det ikke bare en kraftig forbedring i lesbarhet, men også i feilhåndtering og optimalisering. Det er lurt å ikke finne opp hjulet på nytt, og heller dra nytte av bibliotek. Slik kan tiden bli brukt for å kode det som er unikt for hvert prosjekt.

Slike eksempler på refaktorering har vi mange av, og det viser at disse seansene med *code review*'s var veldig nyttige og noe vi hadde god nytte av.

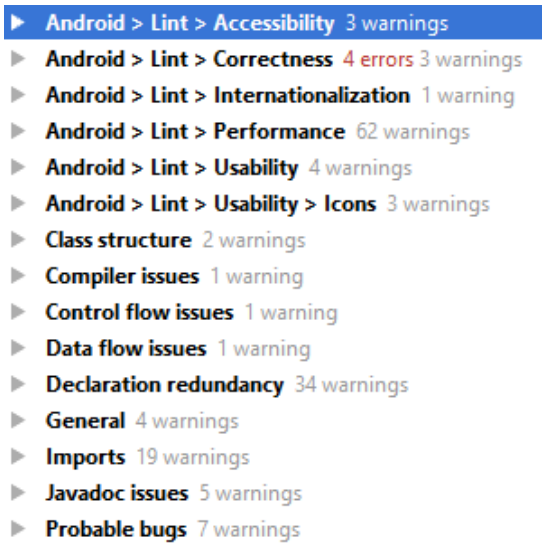
8.1.1 Workshops

Siden ulike gruppemedlemmer fikk dypere innsikt i nye teknologier, ønsket vi å spre kunnskapen mest mulig innad i gruppen. Derfor bestemte vi oss for at i løpet av prosjektet skulle vi sette av flere dager der forskjellige gruppemedlemmer kunne holde en workshop for de andre utviklerne. Den første workshoppen ble holdt av Steffen Granberg den 27/01 i .Net Core. Her gikk han igjennom hvordan .NET Core prosjektet var strukturert og ga en innføring i C#. Den neste workshoppen ble holdt av Ole Andre Slettum den 08/03, hvor det ble gitt enn innføring i iOS utvikling og Swift 3.

8.2 Android

8.2.1 Lint

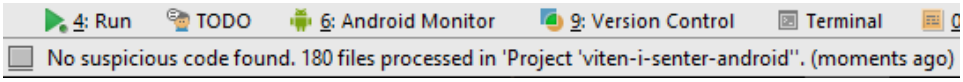
"Lint" eller "Lintere" er samlebegrep som omfavner verktøy som generelt kjører statisk analyse av kildekode for å flagge mistenkelig kode[91] som f.eks. kan bruke foreldete elementer, API kall som ikke støttes av nåværende versjon, potensielle bugs, ubrukte ressurser, manglende internasjonalisering av strenger, for å nevne noe. Android Studio har et slikt Lint-verktøy innebygd[92] som vi har brukt for å sikre kvalitet på kildekoden. Vi begynte aktivt å rette feil i sprint 5 (figur 40 viser resultatet av Lint-analyse kjørt tidlig i sprinten), og hadde i løpet av sprint 6 null feil i følge analysen. Se figur 41 for resultatet av analyse på *master branch* etter at all hovedfunksjonalitet var implementert og feilkilder fjernet.



Figur 40: Lintanalyse kjørt i sprint 5 på Androidkoden

Det er enkelte advarsler som vi ikke var enige i, eller som bare ikke gjaldt for vårt oppsett, og disse er ignorert (se appendix G.1 for liste over alle advarsler som er ignorert).

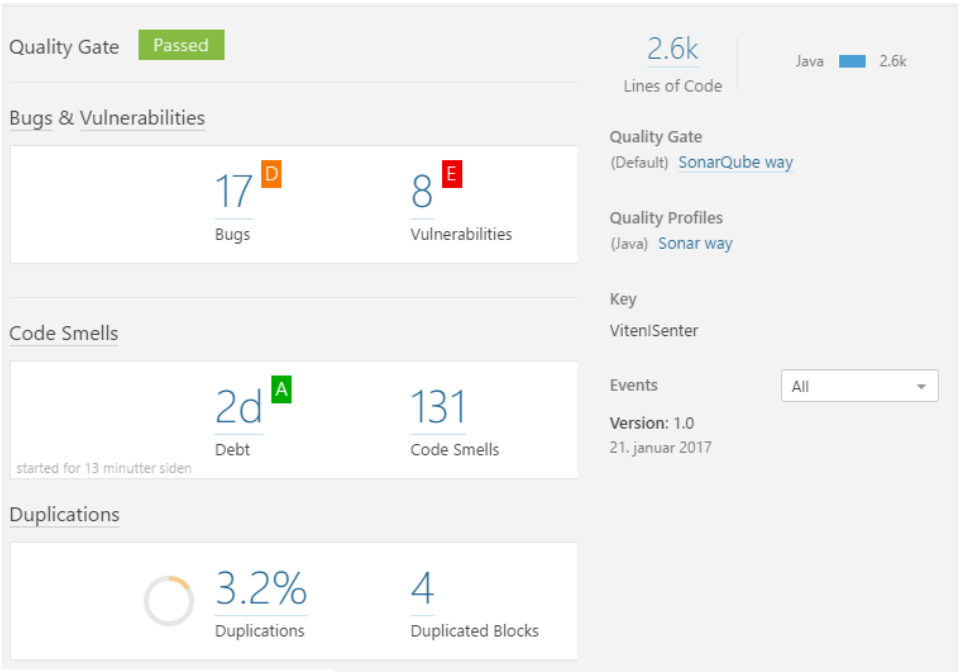
8.2.2 SonarQube



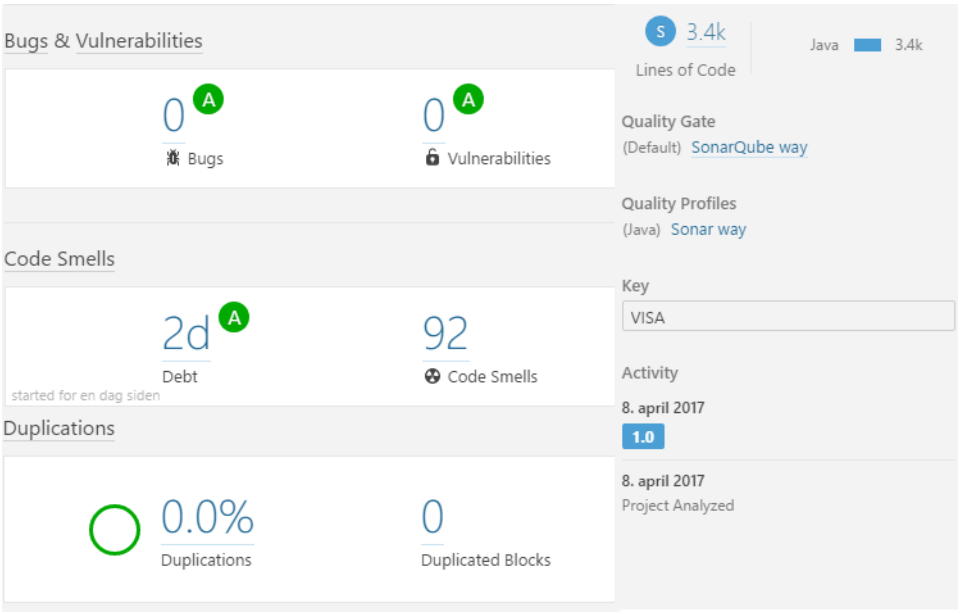
Figur 41: Lintanalyse kjørt i sprint 6 på Androidkoden

SonarQube er en *open source*-plattform for kontinuerlig inspeksjon av kodekvalitet. Den kontrollerer mye av det samme som Lint, men går mer i detalj, og kjører også analyser basert på duplisering av kode, kodestandarder, kompleksitet, bugs og sårbarheter[93].

Vi har brukt SonarQube både som et eksternt analyseverktøy[94] og i form av en plugin i Android Studio/IntelliJ (SonarLint [95]). Sistnevnte advarer kontinuerlig mot dårlig kode direkte i Android Studio, mens det eksterne verktøyet går mer i detalj og har rapporteringsmuligheter (se figur 42).



(a) Analyse i sprint 1



(b) Analyse i sprint 6

Figur 42: Resultat av Sonar-analyse

8.3 iOS

8.3.1 SonarQube

Vi hadde i starten planer om å bruke SonarQube for statisk kodetesting av Swift-kodebasen. Den offisielle Swift-*pluginen* til SonarQube koster 5000£, kjøp av denne ble raskt skrinlagt. Det finnes en *open source* Swift-*plugin*, som vi undersøkte raskt, men ikke fikk til å fungere. *Pluginen* er egentlig bare en samling av flere open source-verktøy for å analysere kodekvalitet, formatering osv. Derfor har vi i steden satt opp disse verktøyene separat for å imitere en SonarQube løsning, sammen med jevnlige *code review*'s følte vi dette sikret kodekvaliteten.

8.3.2 Swiftlinter

Pluginen Swiftlinter[96] ble brukt for å sikre at Swift-koden fulgte gode standarder og sedvaner som finnes i språket. Vi begynte å bruke dette verktøyet ganske sent i prosjektet. Dette fordi vi tenkte at IDE'en (XCode) sikret det meste av slike ting, men siden Swift er relativt nytt viste det seg at XCode ikke tok formateringsfeil og lignende. Da vi kjørte Swiftlinter for første gang, signaliserte den over 600 *warnings*. De fleste av disse var tomme linjer, parametre med ekstra mellomrom og lignende, dermed kunne disse fjernes med kommandoen som følger med Swiftlinter kalt Autocorrect. Gjenstående da var *warnings* som gikk på for mange parametre i funksjoner, switch istedenfor *nested* if's osv. På slutten av prosjektet hadde vi 0 *warnings* fra denne linteren, og prosessen med å fjerne de, hjalp oss også med å refaktorere koden. Noen *warnings* valgte vi å ignorere da dette var noe vi var uenig i eller ikke prioriterte, listen over ignorerte finnes i Appendix G.2.

8.3.3 Taylor

Taylor[97] er en annen *lint plugin* som sikrer at koden følger de anbefalte standarene til Swift. Den sjekker struktur på koden, og kommer også med *warnings* når den oppdager ting som ikke er anbefalt. Vi hadde som Swiftlinter her en del *warnings*, men er på leveringstidspunktet nede i 10, dette er *warnings* vi har tatt en vurdering på at vi ikke gjør noe med, fordi vi mener Taylor ikke forstår oppsettet eller signaliserer feil som ikke har stor betydning.

8.3.4 Lizard

Statisk analysering av kode er en annen måte å undersøke om koden er for komplisert skrevet, og bør refaktoreres. Innenfor statisk analysering har man noe som heter *cyclomatic complexity number* eller CCN. Dette er et tall som gir en pekepinne på hvor kompleks en funksjon er, det betyr i realiteten hvor mange "veier" det er gjennom denne koden. Andre ting som lengde spiller også inn. Taylor har definert at en CCN under 15 er greit, mens høyere enn dette vil generere en *warning*. Dette støttes også opp av flere andre som har analysert CCN-skalaer[98]. Alt over en CCN på 15 bør refaktoreres da dette indikerer at koden er for komplisert.

Lizard[99] er et program som kjører denne typen kompleksitetsanalyse. Vi kjørte en Lizard analyse av alle kodefilene på iOS prosjektet, og Lizard sa at det var ingen funksjoner som ga en CCN på over 15. Testen ble kjørt på alt av kode som vi har skrevet (se figur 9 seksjon 5.1 for oversikt) det vil si ca 3500 linjer kode. Dette mener vi indikerer at koden er enkel å vedlikeholde, og at vi har modularisert koden på en god måte. Siden resultatet var så bra, lurte vi på om det faktisk var en reell test, derfor kjørte vi testen på hele iOS kodebasen, det vil si alle inkluderte biblioteker og *dependencies*. Da ble det totalt 13 *warnings*, noe som viser oss at testen faktisk finner komplekse kodestrukturer, som igjen verifiserer at vår kode møter kravene for lav CCN.

8.4 Frontend

8.4.1 Plugins til Sublime

Vi har ikke brukt et eget IDE for utvikling av frontend, men kildekoderedigeringsverktøyet Sublime Text 3[100]. En rekke *plugins* ble brukt for å sikre kodekvalitet.

- **SublimeLinter**[101] - Rammeverk for lint-støtte i Sublime.
  - Plugins til SublimeLinter:
    - jshint (JavaScript)
    - csslint (CSS)
    - html-tidy (HTML)
- **Polylint**[102] - Linter spesielt for Polymer.
- **Polymer Snippets for Sublime**[103] - Kommer med forslag til kodesnutter/maler som kan legges

inn ved et tastetrykk.
- **LiveReload**[104] - Oppdaterer automatisk nettleseren ved kodeendringer.
- **HTML/CSS/JS Prettify** - Rydder opp koden for økt lesbarhet.

8.5 Backend

Siden .NET Core er et nytt rammeverk finnes det ikke all verden med ferdige *plugins*. Det finnes en del i Visual Studio, men siden utvikleren som utviklet dette jobbet på Mac, var ikke disse tilgjengelig. Derfor ble det også her viktig med jevnlige *code reviews* og diskusjoner for å sikre kodekvalitet.

Det finnes derimot noen verktøy som vi kan bruke for å analysere koden - disse er gratis, og vi synes det fungerte bra.

8.5.1 Omnisharp

Omnisharp[105] er Microsoft sin linter, som gir funksjonalitet som kodeforslag, viser *warnings* og *errors*. Dette verktøyet ble brukt for å sikre en god kodestandard. Det effektiviserte kodingen, ved at man slipper å slå opp i dokumentasjon hele tiden.

8.5.2 Lizard

Som beskrevet i punkt 8.3.4 brukte vi Lizard for å sjekke kompleksiteten til koden i iOS. Dette verktøyet støtter også c# kode, derfor brukte vi det også her.

Etter å ha kjørt en Lizard test på prosjektet viste dette at heller ikke her var det noen funksjoner som ifølge Lizard krevde refaktorering. Dette viser at ved hjelp av de andre verktøyene og med jevnlige *code reviews* har vi klart å oppnå god og vedlikeholdbar kode på backend.

9 Testing

9.1 Brukertester

9.1.1 Ansatte

I sprint 3 holdt vi brukertest av administrasjonsgrensesnittet med produkteier. Testen ble gjennomført ved å gi produkteier tilgang til administrasjonsgrensesnittet med tilhørende oppgaver hun skulle gjennomføre. Underveis skulle produkteier tenke høyt om hvordan hun opplevde systemet - både det som var positivt og det som var negativt.

Første oppgave gikk ut på å legge til et nytt rom. Nederst til høyre i grensesnittet er det knapper for de ulike funksjonene - legg til nytt eksperiment/nytt rom etc. Vi fikk umiddelbart tilbakemelding på at knappene ikke var så lett identifiserbare, det burde også ha vært tekst der som beskrev hva knappene gjorde. Under denne oppgaven fikk vi også tilbakemelding om at knappene bør minimeres etter at de er trykket på.

Andre oppgave var å legge til et nytt bilde på rom. I grensesnittet lå det allerede to rom uten bilder. Det viste seg at ved opplasting av et bilde tilknyttet et rom så ble bildet lagret som bildet på alle rommene som var registrert.

Tredje oppgave var å legge til et nytt eksperiment. Her fikk vi tilbakemelding på at produkteier kunne tenke seg farger og understreking i HTML-editor for ny beskrivelse av eksperiment. I tillegg - når man hadde laget ferdig et eksperiment og prøvde å lage et nytt igjen eller trykket ”Avbryt” så ble ikke inputfelter tømt. Vi fikk også tilbakemelding om at hintene tilknyttet et eksperiment burde nummereres.

Fjerde oppgave var å legge til et nytt quizspørsmål. Her fikk vi tilbakemelding om at svar på spørsmålet bør synes på oversiktslisten over alle spørsmålene.

Femte oppgave var å legge til en ny quiz. Her ble det ikke påpekt noen spesielle mangler.

Andre generelle tilbakemeldinger:

- Når det spørres om et ”objekt” skal slettes bør navnet på objektet vises
- Midtstillingsknapp i HTML-editor for eksperimentbeskrivelse burde reintroduseres

Se tabell 1 for liste over status på problemene avdekket i brukertesten.

Problem	**Løsning**	**Alvorlighets-grad**	**Prioritet**	**Lagt i backlog?**	**Status**
Knapper for å legge til nytt eksperiment, rom, osv. var vanskelig å kjenne igjen.	Legge til tekst-etikett ved siden av knappene.	Problematisk	Middels	Ja	Utbedret
FAB-meny blir ikke lukket ved valg	Legge til close() på on-tap-metode.	Uproblematisk	Lav	Ja	Utbedret
Nylig opplastet rombilde ble automatisk lagt til på andre rom uten bilde.	Fikse kode.	Problematisk	Høy	Ja	Utbedret
Input-felter ble ikke tømt etter at man la til noe nytt eller endret eksisterende	Fikse kode.	Uproblematisk	Lav	Ja	Utbedret
Svaralternativer ble ikke vist på oversikts-liste over Quiz-spørsmål	Legge til visning.	Uproblematisk	Lav	Ja	Utbedret
Dialog for sletting sa ingenting om akkurat hva som skulle slettes	Legge til informasjon i slettedialog	Uproblematisk	Lav	Ja	Utbedret

Tabell 1: Problemer avdekket ved brukertest av webapplikasjonen

9.1.2 Besøkende

Grunnet ombygging på Vitensenteret, ble det vanskelig å gjennomføre brukertest med en skoleklasse slik vi hadde tenkt. Vi valgte derfor å invitere noen medstudenter ned til senteret for en omvisning og test av applikasjonene. De fikk alle en testmobil og oppgavene nevnt under, med beskjed om å beskrive hva de gjorde og problemer som oppsto underveis, samtidig som en av oss noterte i et testskjema.

Testobjektene fikk i oppgave å utføre følgende:

1. Sjekke åpningstider til senteret.
2. Skanne et eksperiment
3. Lese hint om eksperimentene
4. Prøve å spille Nærmest null, etter at det rette eksperimentet er skannet.
5. Spille quiz
6. Registrere poengsum
7. Endre språk
8. Finne frem igjen tidligere skannet eksperiment

Allerede ved den første oppgaven ble det avdekket en svakhet i systemet. Informasjonssiden er ikke responsiv nok i forhold til telefoner med små skjermer. Applikasjonen ble for første gang testet på en eldre Android-telefon. Knappen for veibeskrivelse ble for stor, slik at den dekket over telefonnummeret.

Ved skanning av eksperiment ble det avdekket en ny svakhet i tillegg til forbedringsmuligheter. Skanningen med iPhone-applikasjonen gikk raskt. Faktisk **for** raskt. QR-skanneren var så følsom og kjapp at testobjektet ikke umiddelbart forsto at skanningen var vellykket og at han hadde kommet videre. Han foreslo at det kunne være en slags respons som ga tilbakemelding om vellykket skanning.

Testobjektene som hadde Android-telefoner hadde problemer med å få skannet og fikk avdekket et problem vi ikke var klar over. Da vi testet skanningen selv under utviklingen, var det i optimale lysforhold og sittende. Det vil si at det ikke var avstandsforskjell mellom QR-kode og kamera fra da vi gikk inn på skanneren til vi faktisk skulle skanne. Vi hadde derfor ikke fått med oss at autofokus bare slo inn i det skanner-aktiviteten ble aksessert, og ikke kontinuerlig. Testobjektene gikk inn på skanneren først, og flyttet seg deretter nærmere QR-kode for å skanne, og da med helt feil fokus.

Etter å ha skannet eksperimentet "Nærmest null", gikk de inn på fanen med hint, og fikk deretter i oppgave å starte selve spillet. Den ene Android-brukeren opplevde litt problemer med å dra tallene ut i boksene, og måtte bruke to forsøk på flere av dem. En annen bruker savnet en reset-knapp for å lettere prøve nye kombinasjoner av tall.

Neste oppgave var å spille ferdig en quiz, og registrere poengsum. Her kom samme problem som med informasjonssiden - nederste delen av skjermen ble borte på de mindre telefonene på Android (noe av timeren). Ellers hadde brukerne ingen problemer.

Neste oppgave var å endre språk fra norsk til engelsk. Det ene testobjektet brukte litt tid på å finne ut hvordan, kanskje 5-10 sekunder, men ikke nok til å bli frustrert.

Den siste oppgaven var å finne igjen tidligere skannede eksperimenter. Alle testobjektene fant lett frem til den menyen, men Android-brukerne fikk avdekket en svakhet. På den ene testtelefonen kræsjet applikasjonen hvis brukeren gikk inn på et tidligere skannet eksperiment annet enn "Nærmest null". Årsaken til dette er at vi ikke hadde tenkt på å fjerne eksperimenter fra listen over tidligere skannede ved nullstilling av databasen. Når databasen nullstilles, blir alle eksperimenter slettet bortsett fra "Nærmest null", som er hardkodet i backend. Det vil si at når andre eksperimenter opprettes på nytt, vil de få nye, unike scan-ID'er, som ikke lenger stemmer overens med de som eventuelt ligger i listen over tidligere skannede eksperimenter på forskjellige telefoner. Hvis en bruker av en av disse telefonene prøver å gå inn på et tidligere skannet eksperiment, vil dette gjøre oppslag mot databasen med en gammel, ikke-eksisterende scan-ID, og appen vil kræsje. Dette vil likevel ikke være det største problemet, siden databasen ikke vil resettes i et produksjonsmiljø.

Se tabell 2 for liste over status på problemene avdekket i brukertesten.

Plattform	Problem	Løsning	Alvorlighets-grad	Prioritet	Lagt i backlog?	Status
Android/ iOS	Info-side ikke responsiv	Endre oppsett i layoutfilen	Problematisk	Middels	Nei	Anbefaler utbedring
iOS	QR-skanning for rask.	Respons ved vellykket skanning	Uproblematisk	Lav	Nei	Bør vurderes
Android	Fikk ikke skannet	Fikse kontinuerlig autofokus	Kritisk	Høy	Ja	Utbedret
Android	Quiz ikke responsiv	Endre oppsett i layoutfilen	Problematisk	Middels	Nei	Anbefaler utbedring
Android	Kræsj hvis bruker prøver å gå inn på eksperiment skannet før reset av database	Legge inn en sjekk på fragmentet som viser tidligere skannede eksperimenter, slik at listen tømmes ved databasereset	Uproblematisk	Lav	Nei	Gjelder kun i utviklingsstadiet

Tabell 2: Problemer avdekket ved brukertest av applikasjonene

9.2 iOS

9.2.1 Nettverkstester

```
/**
 Tests QuizCategory.getAll-method. Tests method with statuscode 200 and proper json
 */
func testQuizCategoryGetAllSuccess() {
  //Read from file
  if let url = Bundle.main.url(forResource: "allquizcategoriesjson", withExtension: "json") {
    do{
      let data = try Data(contentsOf: url)
      let jsonData = try JSONSerialization.jsonObject(with: data, options: [])
      //Block GET-requests
      stub(condition: isMethodGET()){ _ in
        return OHHTTPStubsResponse(
          jsonObject: jsonData,
          statusCode: 200,
          headers: ["Content-Type": "application/json"]
        )
      }
      //Check object equality
      QuizCategoryService.getAll(id: 1) { quizcategories, statuscode in
        XCTAssertEqual(quizcategories[0].id, 1)
        XCTAssertEqual(quizcategories[0].name, "Blanding")
        XCTAssertEqual(quizcategories[1].id, 2)
        XCTAssertEqual(quizcategories[1].name, "Eksempel")
        XCTAssertEqual(quizcategories[2].id, 3)
        XCTAssertEqual(quizcategories[2].name, "eksempel2")
      }
    } catch {
      print("unable to read from file")
    }
  }
}
```

Listing 9.32: Eksempel på *unit testing* av nettverksfunksjon

For iOS-applikasjonen så har vi skrevet *unit test*'er for nettverkslogikken. Vi har skrevet tester for alle nettverksfunksjonene hvor vi simulerer HTTP-trafikk med statuskode 200 og 404.

Ved statuskode 200 sjekker vi at objektet som er blitt opprettet på bakgrunn av data returnert fra *backend*'en har like data som dataene som lå i det returnerte JSON-objektet. Ved statuskode 404 sjekker vi at objektet som er blitt opprettet etter svar fra *backend* er et tomt objekt.

Vi simulerer et nettverkskall ved å bruke et bibliotek som heter OHHTTPStubs[106]. Ved hjelp av OHHTTPStubs blokkerer vi alle GET-*request*'s og sender et JSON-objekt med parametre til nettverksfunksjonen.

Listing 9.2.1 viser koden for testing av QuizCategori-endpointet hvor funksjonen som testes skal hente ned alle quizkategorier fra API'et. Først leses JSON-data fra fil, GET-metoden blokkeres slik at man kan simulere et nettverkskall og deretter gjøres nettverkskallet. Videre er det lagt til en *closure* som kjører når nettverkskallet er ferdig, *closure*'n sjekker om objektets data er lik svaret fra API'et.

9.2.2 UI-tester

På iOS er det skrevet UI-tester med XCode sitt innebygde "XCTest"-rammeverk[107]. Testene er laget for å sjekke at navigasjonen og knappene i applikasjonen fungerer. Et eksempel ligger i listing 9.33. Her navigerer UI-testen seg fra hovedsiden til valg av quizkategori og quiz før den gjør en quiz helt automatisert.

```
1  /// Tests quiz navigation and doing a quiz
2  func testQuizNavigationAndQuiz() {
3      let app = XCUIApplication()
4      app.tabBars.buttons["Quiz"].tap()
5
6      let cellsQuizCat = app.tables.cells
7      let firstCell = cellsQuizCat.element(boundBy: 0)
8      firstCell.tap()
9
10     let cellsQuiz = app.tables.cells
11     let firstQuizCell = cellsQuiz.element(boundBy: 0)
12     firstQuizCell.tap()
13
14     app.buttons["playQuizButton"].tap()
15
16     let altacellButton = app.buttons["altACell"]
17     let altbcellButton = app.buttons["altBCell"]
18     let altccellButton = app.buttons["altCCell"]
19     let altdcellButton = app.buttons["altDCell"]
20
21     altacellButton.tap()
22     altbcellButton.tap()
23     altccellButton.tap()
24     altdcellButton.tap()
25     sleep(10)
26 }
```

Listing 9.33: UI-test på iOS, testing av quiz

9.3 Android

9.3.1 Nettverkstester

På Android har det blitt skrevet nettverkstester for alle kall som henter et JSON-objekt fra API'et. Alle disse testene har samme grunnleggende struktur, med noen variasjoner i forhold til hvilke objekt som hentes. Testen for *Experiment* er brukt som eksempel her.

```
@RunWith(AndroidJUnit4.class)
public class ExperimentNetworkTest extends ActivityInstrumentationTestCase2<HomeScreenActivity> {

String body = "{\n" +
            "  \"scanId\": \"4ecc184f-d8a7-4b0d-8046-14f5effefeb5\",\n" +   ...
```

Listing 9.34: ExperimentNetworkTest 1 (Android)

Testen er satt opp som en JUnit 4 testklasse med applikasjonens "*main*", HomeScreenActivity, som test case. En faktisk JSON-respons fra API'et på et spesifikt eksperiment er lagret som strengen *body*.

Den første testmetoden vises i kodelisting 9.35:

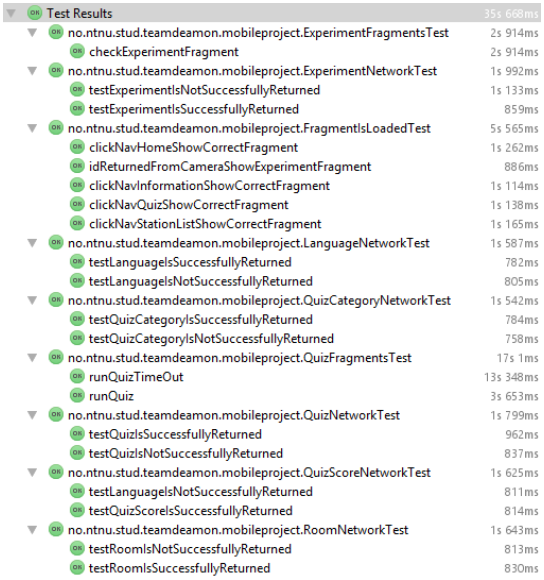
```
    @Test
    public void testExperimentIsSuccessfullyReturned() throws Throwable {
        setUpMockWebserver(200);
        final boolean[] onsuccessCalled = {false};
        Experiment originalExperimentModel = new Gson().fromJson(body, Experiment.class);
        final Experiment[] newExperimentModel = new Experiment[1];

        getActivity().runOnUiThread(new Runnable() {
            @Override
            public void run() {

                ExperimentAPIUsage.getExperiment(getActivity(), "id", new ResponseCallback<Experiment>() {
                    @Override
                    public void onSuccess(Experiment model) {
                        onsuccessCalled[0] = true;
                        newExperimentModel[0] = model;
                    }

                    @Override
                    public void onFailure() {
                        // Method implemented from interface.
                    }
                });
            }
        });

        SystemClock.sleep(500);
        assertEquals(originalExperimentModel.toString(), newExperimentModel[0].toString());
        assertEquals(true, onsuccessCalled[0]);
    }
```

Listing 9.35: ExperimentNetworkTest 2 (Android)

Først settes en *mocket* versjon av API'et opp, som sender JSON-dataene i *body* som respons med statuskode 200 ("OK"). Deretter lages originalExperimentModel direkte ut i fra *body*, og newExperimentModel via et kall til den *mockede* serveren.

Hvis kallet er vellykket settes den boolske variabelen onSuccessCalled til sann.

Etter kallet er det lagt inn en liten pause på et halvt sekund, så det er sikkert at det asynkrone kallet er ferdig kjørt før de to eksperimentmodellene sjekkes opp mot hverandre. Hvis de er like, sjekkes det om den boolske variabelen er sann. Den andre testmetoden er veldig lik den første, med den forskjellen at den tester hva som skjer ved et **feilet** kall. Dette gjøres ved å få den *mockede* webserveren til å returnere statuskode 500 ("Internal Server Error"). Da skal ExperimentAPIUsage respondere med onFailure().



Figur 43: Resultat av alle tester kjørt på Android

9.3.2 UI-tester

UI på Android er testet ved hjelp av biblioteket Espresso[108]. De åtte UI-testene har som hovedoppgave å kontrollere at navigeringen fungerer som forventet, slik som checkExperimentFragment (kodelisting 9.3.2).

```
1   @Test
2   public void checkExperimentFragment() {
3       mActivityRule.getActivity().showExperimentFragmentWithId("4ecc184f-d8a7-4b0d-8046-14f5effefeb5");
4
5       // Check that the correct fragment has been loaded.
6       onView(withId(R.id.header_cover_layout)).check(matches(isDisplayed()));
7
8       // Check that the correct tab has been loaded.
9       onView(withId(R.id.fragment_experiment_information_id)).check(matches(isDisplayed()));
10
11      // Navigates to a new webview via tab
12      Matcher<View> matcher = allOf(withText("Hints"), isDescendantOfA(withId(R.id.tabs)));
13      onView(matcher).perform(click());
14
15      // Check that the correct tab has been loaded.
16      onView(withId(R.id.fragment_hint_id)).check(matches(isDisplayed()));
17
18      // Navigates to a new webview via tab
19      matcher = allOf(withText("Extra"), isDescendantOfA(withId(R.id.tabs)));
20      onView(matcher).perform(click());
21
22      // Check that the correct tab has been loaded.
23      onView(withId(R.id.fragment_extra_functions_id)).check(matches(isDisplayed()));
24  }
```

Listing 9.36: UI-test for eksperiment-*fragment* (Android)

Denne testen navigerer først til eksperiment-siden og sjekker at riktig fragment er aktivt. Den sjekker deretter om riktig fane er valgt (det er tre faner på eksperiment, og "Info" skal alltid åpnes først).

Så navigerer testen applikasjonen til andre fane, “Hints”, og sjekker om denne er åpnet riktig. Til slutt sjekker den siste fanen.

9.4 Backend

9.4.1 Controller-testing

Controller-testene skal fortrinnsvis teste to ting: At valideringen av en modell som sendes inn til et *endpoint* fungerer som det skal. Samt at det returneres korrekt feilkoder når noe galt skjer. For at dette skal bli reelt, ønsker vi ikke å gjøre reelle databasekall, derfor må dette *mockes*.

```
1  public void setupMockRepos(){
2      mockMapper = new Mock<IMapper>();
3      languageCodeRepoMock = new Mock<ILanguageCodeRepository>();
4  }
5
6  [Fact]
7  public void GetLanguageCodeShouldReturnNotFound (){
8      setupMockRepos();
9
10     languageCodeRepoMock.Setup(mock => mock.GetSingle(It.IsAny<int>()))
11                         .Returns<LanguageCode>(null);
12
13     var controller = new LanguageCodeController(
14                                                 languageCodeRepoMock.Object,
15                                                 mockMapper.Object);
16
17     var result = controller.Get(2);
18     Assert.IsType<NotFoundObjectResult>(result);
19 }
```

Listing 9.37: Eksempel på controller test - *backend*

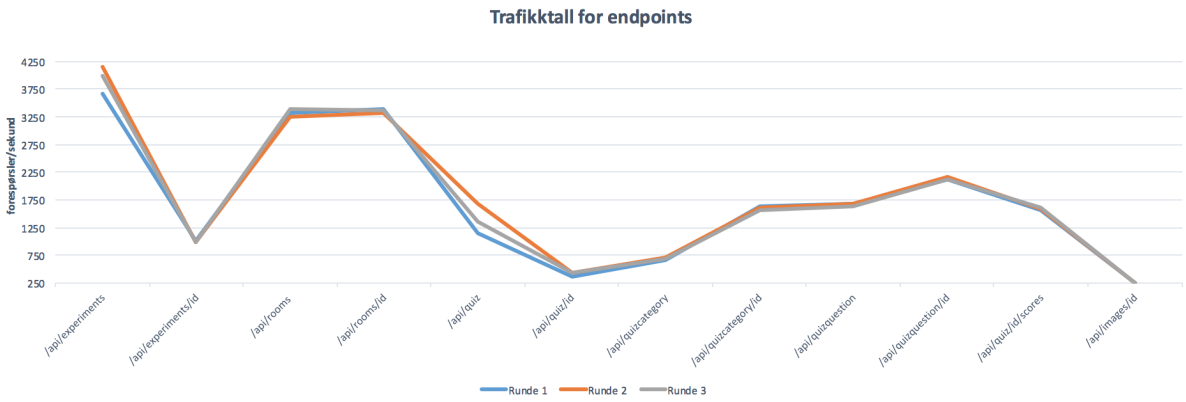
I listing 9.37 ser vi hvordan et eksempel på en slik så ut. Her setter vi opp *mocker* av alle *dependencies* en klasse har, som vi forklarte i seksjon 5.3 om design av *backend*-løsning skytes alle *dependencies* inn gjennom *constructor injection*. Derfor kan vi enkelt *mocke* disse nå. Så definerer vi hva som skal returneres når repositorien - som er *mocket* - kalles. Deretter kan vi verifisere at kontroller faktisk returnerer riktig, i dette tilfellet statuskode NotFound (404).

9.4.2 Trafikktesting av server

Siden det fra oppdragsgivers side var viktig at systemet kunne takle opp mot 500 brukere samtidig (se operasjonelle krav seksjon 3.4) ønsket vi å kjøre noen tester som ville gi oss en indikasjon på om vi var i nærheten av å nå dette målet. Til å utføre disse testene brukte vi et verktøy kalt Wrk[109], dette er et skript skrevet i Lua[110] som kan utføre testing av nettverkskall. Dette gjør den ved å lage veldig mange HTTP-tilkoblinger til en server, og parallelt kjøre kall mot den. Verktøyet passer godt til å teste API’er, som var det vi ønsket å gjøre.

Vi kunne ikke kjøre dette på skolens nett da det generer veldig mye trafikk. Vi kjørte det derfor fra *reverse proxy*’en til API-serveren. Dette gjør at det ikke testes bare lokalt på maskinen. Uansett vil man ved denne typen oppsett få *latencyen* som oppstår på det reelle nettet. Derfor brukes disse resultatene som en pekepinne, og ikke som en fasit på hvor mye serveren takler.

Serveren som kjørte API’et ved testing var en linux virtuell maskin med 16 GB ram, og 4 kjerner prosessor. Det ble kjørt tre tester på 30 sek med 1 minutts pause på alle *endpoint*’ene til API’et. Kommandoen som satter i gang en en slik runde ser vi i listing 9.38. Her ser vi at det kjøres to tråder med 100 tilkoblinger som sender så mange forespørsler som mulig i løpet av 30 sekunder.



Figur 44: Trafikktest av API (laget i MS Excel)

```
1 wrk -t 2 -c 100 -d 30 http://192.168.110.10:8080
```

Listing 9.38: Kommando som starter en wrk loadtest

I Figur 44 ser vi resultatet av denne testen. Her kan vi se at antall forespørsler per sekund varierer fra over 4000, til nede i 250. Dette er ikke så rart, da det varierer veldig på hva som skjer ved et slikt kall. Ting som tar tid er for eksempel databasekall der mange tabeller skal settes sammen og sammenlignes. Et eksempel på et slikt kall er henting av en spesifikk quiz. Siden *lazy loading* ikke finnes i .NET Core enda, må vi manuelt gå igjennom alle spørsmålene til en quiz og hente ned svarene til disse spørsmålene, som ligger i separate tabeller. Dette øker responstiden på denne typen kall.

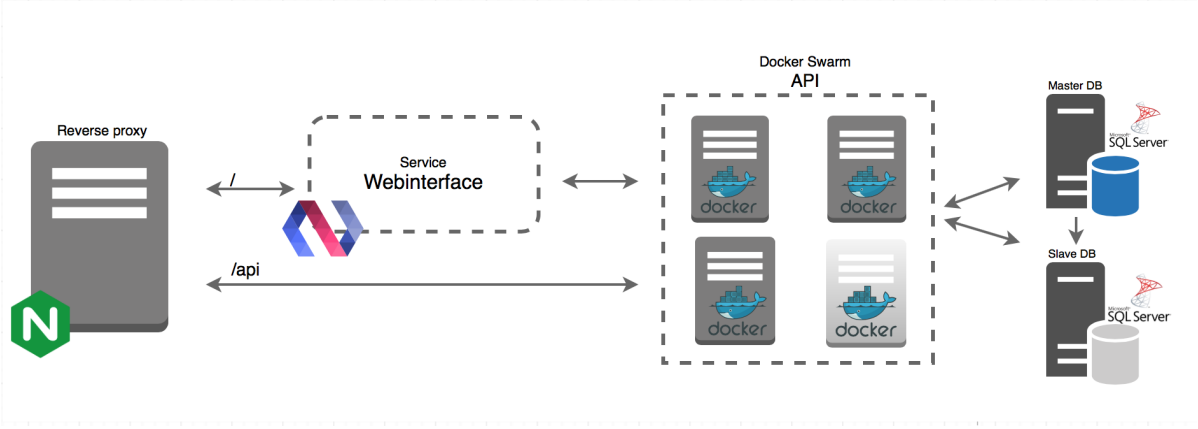
Likevel ser vi at de aller fleste kall ligger på over 1000 forespørsler per sekund. Dette mener vi er veldig bra, og vi vil med dette klare å tilfredsstille de operasjonelle kravene med god margin. Hvis applikasjonen blir mer populær enn forventet fra Vitensenteret sin side, og det er ønskelig å skalere den til andre Vitensentere vil man kunne takle flere forespørsler hvis det utvides med flere servere. Dette er mulig i det oppsettet vi foreslår(se kapittel 10)

10 Produksjonssetting

Under utviklingsprosessen satte vi opp et utviklingsmiljø som passet våre behov. Det vil i en produksjonsetting være andre krav som vi må ta hensyn til, derfor vil vi nå gå igjennom hvordan vi mener at en produksjonsetting av applikasjonene kan skje, og hvilke endringer som eventuelt må gjøres for at applikasjonene skal være produksjonsklare.

Det er ikke et mål i vår oppgave å produksjonsette løsningen, men vi kommer med en anbefaling på en mulig løsning, basert på research og våre erfaringer, fordi Vitensenteret ønsket vi skulle komme med et slikt forslag.

10.1 Oppsett av servermiljø



Figur 45: Produksjonsmiljø (laget i draw.io)

Det er flere måter å sette opp et moderne miljø for webtjenester. Vi har hatt fokus på å foreslå et oppsett som enkelt vil kunne skalere for å møte et voksende antall brukere. Dette fordi Vitensenteret har uttrykt en mulighet for at applikasjonen også kan passe for andre vitensentere. Det vil derfor være en mulighet at antall brukere kan øke over det antallet vi ser som realistisk bare for Vitensenteret Innlandet.

Viktige punkter å ta hensyn til i et slikt oppsett inkluderer:

- Sikkerhet
- Mulighet for skalering og dynamisk skalering basert på trafikk
- Backup av database og informasjon
- Enkelt å drifte

Vitensenteret har i dag ikke serverne som skal til for å sette opp dette, men de benytter et eksternt IT-firma som vil ha muligheten til å hjelpe med driftsetting. Vitensenteret har også sagt at det ikke er noe problem for dem å betale for en virtuell serverpark, som for eksempel Amazon EC2 skytjenester[111], OpenStack[112], Azure[113] eller lignende skulle dette være nødvendig.

I lys av dette, og andre punkter skrevet over, har vi foreslått et mulig produksjonsoppsett som driftsetter systemet.

10.1.1 Nginx

Siden API'et skrevet i .Net Core skal kjøres i Kestrel - Microsoft sin nye webserver, bør ikke denne stå direkte ut mot nettet (les mer om dette i 5.1). Derfor må det ut mot internett stå en *reverse proxy*, som dirigerer trafikken til de riktige lokale tjenestene. Dette kan også tilby tilstrekkelige sikkerhetstiltak, noe Kestrel mangler.

Det finnes flere forskjellige løsninger her, og populære webservere som vil kunne benyttes slik inkluderer Apache[114] og Nginx[50]. Nginx har i det senere blitt mer og mer populær, grunnet sin store kapasitet og raske prosessering. I en test gjort av Dreamhost ser vi at Nginx har lavere bruk av minne, og kan håndtere flere forespørsler per sekund enn Apache[115]. Nginx har også støtte for velkjente tjenester som SSL/TLS kryptering av trafikk, lastbalansering m.m.

I figur 45 ser vi at Nginx er plassert som en *reverse proxy* ut mot nettet. Den har som oppgave å dirigere trafikk inn til den riktige bakenforliggende tjenesten. På denne måten kan vi ha et eksternt domene, for eksempel *vitensenteret.no*, og utifra hvilken *prefix* eller *url-route* sende trafikken til den riktige tjenesten.

Sikkerhet Nginx

Nginx støtter flere sikkerhetstiltak, som for eksempel oppsett av SSL/TLS sertifikater[116]. Dette er noe vi anbefales at settes opp i et produksjonsmiljø og Vitensenteret kan anskaffe seg sertifikater som settes opp med denne tjenesten. Selvom applikasjonen ikke behandler det som kan karakteriseres som sensitive data, vil brukernavn/passord sendes via webgrensenittet.

Det er i dag flere argumenter for å bruke SSL/TLS, ikke bare går dette på å hindre at informasjon lekker ut, men også hindre at andre kan sitte mellom klientene og serverne og manipulere informasjon. På denne måten kan en tredjepart videresende informasjon som ser ut som det kommer fra Vitensenterets sine servere. Ved bruk av sertifikater og SSL/TLS, kan brukerne være trygge på at ingen har manipulert informasjonen som blir presentert til dem.

Vi testet aldri et oppsett av SSL/TLS, men med oppsettet vi anbefaler her vil dette kunne settes opp på Nginx uten at det kreves ytteligere endringer av kode eller applikasjon. Derfor er dette noe som Vitensenteret kan ta med seg i videre vurderinger før en produksjonssetting.

10.1.2 Docker

Bak Nginx ligger alle tjenestene. Microsofts nye tjenester satser hardt på kryssplattform gjennom å tilby docker-containere som kan kjøre de ferdige tjenestene. Docker[87] tilbyr en funksjon som heter *Docker Swarm*[117], dette gjør at man kan sette opp mange containere som noder i et nettverk. En av nodene fungerer som *manager* og brukes til å sette opp og administrere *swarmen*.

Det fine med å sette opp en *Docker Swarm* er at man enkelt kan skalere opp tjenesten ved å starte opp nye containere, i nye, eller eksisterende, virtuelle maskiner. Disse kan så enkelt legges til i *swarmen*. Dette gjør at det er enkelt å scripte dynamisk skalering. De forskjellige nodene kan også tilby ulike tjenester. Noen kan for eksempel fungere som en webserver for web-applikasjonen, mens andre bare tilbyr API'et. Når en forespørsel for en spesifikk tjeneste kommer inn til *swarmen*, vil den automatisk dirigere forespørselen til en node som kjører akkurat denne tjenesten. Systemet støtter også oppsett av en ekstern lastbalanserer som fordeler trafikken mellom nodene, eller *swarmen* kan ta seg av dette selv.

Ved å bruke en slik *swarm* har man "ut-av-boksen" feilhåndtering og skalering. Hvis en node skulle slutte å virke blir den automatisk tatt ut av *swarmen* til den er funksjonell igjen. Det er denne dynamiske arbeidsmåten som gjør denne typen oppsett ypperlig til en rekke tjenester - som et API.

10.1.3 SQL server

Microsoft SQL server har også blitt lansert med Docker-støtte, og kan settes opp som en container. For å håndtere økende trafikk kan det være en idé og sette opp to SQL-servere i et master-slave oppsett. Dette betyr at alle operasjoner gjort på masteren speiles til slaven, og alle lese-operasjoner kan kjøres

mot begge. Dette vil i Vitensenteret sitt oppsett være en fordel da nesten alle kall til databasen er for å lese informasjon - og kan dermed fordeles ut til begge serverne.

10.1.4 Oppsummering

Det foreslåtte oppsettet beskrevet her krever at Vitensenteret ansetter noen til å drifte systemet, eller at et eksternt firma kan gjøre det. Oppsettet kan også pushes opp til en ekstern tjeneste. Microsoft tilbyr for eksempel å hoste .NET Core API'er med tilhørende SQL server i sine skytjenester. Det samme oppsettet kan også settes opp i Amazon EC2, hvor det betales per bruk. Applikasjonen vår er laget på en dynamisk og skalerbar måte, og den kan legges på en rekke forskjellige tjenester.

Oppsettet vi har beskrevet i denne seksjonen når de operasjonelle kravene angitt i seksjon 3.4.

10.2 Utrulling av mobilapplikasjonene

Denne seksjonen diskuterer produksjonssetting av mobilapplikasjonene på App Store og Google Play Store. Både Apple og Google har mange krav til nye applikasjoner.

Den største forskjellen mellom lansering av Android og iOS-applikasjoner er at Apple manuelt reviderer hver eneste applikasjon før den blir publisert[118] mens det i Google Play Store kjøres automatiske skanninger[119].

For å få lansert mobilapplikasjonene måtte man brukt tid på å sørge for at applikasjonene følger Google og Apple sine krav. Underkjenning av applikasjoner er gjerne ikke permanent, man kan utbedre den ved avslag. Ved alvorlige brudd på reglement kan man utestenges permanent fra Apple sitt utviklingsprogram[120].

10.2.1 iOS

Apple sine krav er delt inn i seks hovedseksjoner, listen under oppsummerer de ulike seksjonene:

- **Sikkerhet** - faktorer som forbud mot eksplisitt innhold, mulighet for modifisering av brukerinnhold fra utviklerens side og at applikasjonen er uten fare for å påføre skade på brukeren. iPhone-brukere skal være trygge på at det de laster ned fra App Store er trygt å bruke. [120]
- **Ytelse** - Krav til ytelse inneholder faktorer som krav til korrekte metadata, at den er ferdigtestet, effektiv batteribruk o.l.
- **Programvarekrav** - man kan kun bruke offentlige API, applikasjonen må være kompatibel med IPv6, forbud mot applikasjoner som kan skade enheten o.l.
- **Businesskrav** - krav om velorganisert businessplan, regler for in-app kjøp, abonnementer o.l.
- **Designkrav** - applikasjonen må ikke kopiere design fra andre applikasjoner, den må unngå spamming av brukeren o.l.
- **Juridiske krav** - mobilapplikasjoner må følge lover og regler i etterhvert land den distribueres i.

Listen er på ingen måte en uttømmende liste over hva Apple krever av applikasjoner som skal publiseres til App Store, men gir et innblikk i hva som kreves av applikasjoner som skal lanseres[120].

Noen punkter vi tror vi måtte utarbeidet for å lansere den på App Store

- Lagt til et filter med navn man ikke kan bruke i quizzens toppliste
- Mulighet til å blokkere brukere fra å bruke applikasjonene
- Lagt til metadata(navn, beskrivelse, kategori m.m)
- Ikke et absolutt krav, men muligheten for å kjøre applikasjonen iPad burde vært sett nærmere på
- Bestemme navn og pris.
- Evaluert applikasjonens batteribruk

10.2.2 Android

Som nevnt i seksjon 10.2.1 er det forskjell på hvordan Google og Apple reviderer nye applikasjoner. Ansatte i Apple reviderer hver eneste applikasjon mens Google bruker automatiserte skanninger til å revidere applikasjoner[119]. Applikasjoner som bryter med reglementet blir fjernet når de blir oppdaget.

Under følger en oppsummering av krav som stilles fra Google til nye applikasjoner:[121]

- Begrenset innhold - forbud mot eksplisitt innhold som pornografi, vold o.l
- Åndsverk, villedelse og nettsøppel - Krav til originalt innhold, forbud mot falsk identitet o.l
- Personvern og sikkerhet - åpenhet om bruk av brukerdata og sensitive data, krav til antivirus/sikkerhetsmekanismer.
- Inntektsgenerering og annonser - Applikasjoner må bruke Google sitt faktureringssystem, tydelig varsling om innhold som krever betaling o.l.
- Butikkoppføring og markedsføring - eksklusjon av applikasjoner som driver villedende markedsføring, uoppfordret markedsføring på SMS o.l.
- Familier og COPPA - spesifikke retningslinjer for applikasjoner som er myntet på familier og barn.
- Håndhevelse - håndtering av brudd på retningslinjene.
- Oppdatering og andre ressurser - Oversikt over endringer som er gjort ved nye oppdateringer

På samme måte som for iOS måtte vi gjort en vurdering av applikasjonen opp mot Google sine krav hvis vi skulle lansert den på Google Play Store.

Noen punkter vi tror vi måtte utarbeidet for å få lansert Android-applikasjonen på Google Play Store:

- Bestemt navn og pris
- Lagt til et filter med navn man ikke kan bruke i quizzens toppliste
- Mulighet til å blokkere brukere fra bruke applikasjonen
- Kreve at brukeren godkjenner vilkår for bruk av produktet før de sender inn innhold[122]
  Spesielle krav fordi applikasjonen er myntet på familer og barn:[123]
  - Redegjørelse av om brukere kan samhandle/utveksle informasjon
  - Redegjørelse av om applikasjonen deler innhold med tredjeparter
  - Redegjørelse av om applikasjonen deler brukerens fysiske posisjon med tredjeparter
  - Legge til link til personvernregler på applikasjonens side på Google Play Store
  - Samsvar med reglene i COPPA (Children's Online Privacy Protection Rule)

11 Konklusjon

11.1 Resultat

Vitensenteret ønsket en applikasjon som de har mulighet til å teste i et reelt produksjonsmiljø og som etterhvert kan lanseres. Dette har vi oppnådd, og ved å følge vår oppskrift for å sette applikasjonene ut i produksjon(se kapittel 10) vil mobilapplikasjonene kunne lanseres. Vi leverer kode av høy kvalitet samt et produkt det er enkelt å videreutvikle. Vitensenteret har mange muligheter for både å utvide, men også endre på komponenter i systemet.

Med applikasjonene vi har laget vil Vitensenteret Innlandet bli det første Vitensenteret som tar brukeropplevelsen til det neste nivået med mobilapplikasjoner. Vi vet at mange av de andre Vitensenterne allerede har hørt om vårt arbeid og er interessert i løsningen.

Alle ønskene i prosjektbeskrivelsen(se seksjon 1.4) er oppnådd, og Vitensenteret Innlandet er svært fornøyd med det endelige resultatet. Dette kom frem da vi i slutten av prosjektperioden holdt en presentasjon for alle ansatte på senteret.

11.2 Alternative muligheter og valg underveis

En av de viktigste avgjørelsene vi måtte ta underveis var om vi skulle utvikle mobilapplikasjonene som *native* eller kryssplattformapplikasjoner. Se seksjon 4.5 for dypere diskusjon rundt dette temaet.

Ettersom vi sett i ettertid ikke valgte å implementere noen ekstrafunksjoner som bruker sensorer, kunne nok den nåværende applikasjonen ha blitt utviklet i et kryssplattformspråk. Det betyr allikevel ikke at vi ikke gjorde et riktig valg. Vitensenteret var opptatt av at løsningen skulle være så utvidbar som mulig, og at den skulle følge brukergrensesnittprinsippene tilhørende de enkelte plattformene. Siden Vitensenteret ønsker å utvide med sensorer og vi bruker beacons og kamera på de nåværende applikasjonene mener vi at det var riktig valg å gå for *native* utviklng.

Grunnen til at vi ikke prioriterte sensorer var fordi vi så at dette ville ta for mye tid og gå utover de andre løsningene og hvor mye som ble ferdigstilt. Samtidig mente Vitensenteret at ettersom de nå gjennomgår en ombygging så var det usikkert hvilke eksperimenter som kunne være aktuelle for ekstrafunksjonalitet. Sensorbruk er noe vi anbefaler Vitensenteret å bygge videre på, mer om dette i seksjon 11.3.

Valget av rammverk for API'et var basert på at utvikleren som hadde ansvaret for dette hadde tidligere erfaringer med .NET. Derimot ser vi i ettertid at siden .NET Core er et veldig ferskt rammerverk, har det fortsatt en del barnesykdommer og uferdige løsninger, som har resultert i noen *workarounds*. Vi er uansett fornøyde med valget, og mener at når oppdatereringer kommer til .NET Core vil det være enkelt å oppgradere deler av koden som kan ta i bruk nye funksjoner.

11.3 Videre arbeid

Et av de første punktene Vitensenteret må se på i det videre arbeidet er produksjonssetting. Ettersom systemet vi har utviklet er stort, er det komplisert å sette seg inn i. Dette siden oppgaven inneholder mange teknologier. Vi har prøvd å forenkle videre arbeid ved å dokumentere mye underveis - både i og utenfor kode. Denne rapporten vil kunne fungere godt som dokumentasjon for eventuell videreutvikling.

Vi har som nevnt i seksjon 1.5 begrenset oppgaven noe for å imøtekomme alle ønsker. Dette resulterte i at det ble tid til å implementere to ekstrafunksjoner/spill - Nærmest Null og Hjernespill. Fokuset har ligget på å lage en god og gjennomarbeidet løsning hvor de ansatte på Vitensenteret kan legge til og endre nye eksperimentstasjoner, quizzer og annen informasjon. Siden vi vet at Vitensenteret gjerne øns-

ker å legge til flere ekstrafunksjoner etterhvert har vi gjort dette så enkelt som mulig - programmereren implementerer spillet på mobil, før det videre ved hjelp av web-applikasjonen enkelt linkes rett inn i et eksperiment.

Videre ønsker Vitensenteret å kunne forandre meldingene som *beacons*-enhetene sender ut samt legge til flere *beacons*. Alt ligger til rette for dette, mobilsiden er implementert - det som gjenstår er å lage webdelen.

Quizsvarene sendes med i det første kallet når man henter ned quizzen, derfor vil det være en mulighet for en som kjenner API-adressen til å hente ned alle svarene. Dette var noe vi identifiserte, men sammen med Vitensenteret bestemte vi at det ikke var en prioritert. Ønsker Vitensenteret å gjøre noe med dette, kan det vurderes i videre arbeid.

Det ligger mye potensiale i mobilapplikasjonene. Vi tror at etterhvert som Vitensenteret får testet applikasjonen vil ny type funksjonalitet være ønsket. Eksempler på dette er innhenting av statistikk slik at Vitensenteret kan få oversikt over hvilke stasjoner som brukes mer enn andre.

Vi mener vi har lagt godt til rette for denne type arbeid videre - en ny bacheloroppgave som kan videreutvikle den solide basen vi har laget er noe Vitensenteret kan vurdere.

11.4 Evaluering av gruppens arbeid

Samarbeidet mellom gruppedeltakerne har fungert veldig bra. Vi har sittet mye sammen og jobbet og alle har utført oppgavene de har fått tildelt til avtalte tider. Vi har brukt scrum og Jira som hovedverktøy i prosessen, dette har fungert godt. Jira har hjulpet oss med å følge med på hverandres arbeid, samtidig som det har vært et verktøy for å se om arbeidsmengden tildelt den enkelte har vært for liten eller for stor i løpet av en sprint. Dette er reflektert i prosessen hvor vi etterhvert ble flinkere til å estimere arbeidsmengde per sprint.

Gruppedeltakerne har hjulpet hverandre med motivasjonen og kvalitetssikret hverandres arbeid, for eksempel gjennom jevnlige *code reviews*. Vi har lært veldig mye av hverandre gjennom denne bachelor-oppgaven.

Rapporten har vi brukt mye tid på og vi synes vi har gjort et solid arbeid. Rapporten er refaktorert ved flere tilfeller, som for kodingen i prosjektet har vi lest over hverandres deler flere ganger.

11.5 Avslutning

Vi føler vi har oppnådd det vi ønsket med denne oppgaven og mere til, og vi er veldig fornøyd med resultatet. Videre har Vitensenteret mange muligheter for hvordan de vil bruke produktet videre, og både produksjonssetting og videreutvikling er aktuelt. Avslutningsvis vil vi takke Vitensenteret for et godt samarbeid og mulighetene de ga oss med denne oppgaven.

Bibliografi

[1] Lander, R. Asp.net core middleware fundamentals [Internettside]. Microsoft [sitert 21.02.17]. Tilgjengelig fra: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware.

[2] Apple. Model-view-controller [Internettside]. Apple [sist oppdatert 21/10-2015; sitert 26/3-2017]. Tilgjengelig fra: https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html.

[3] coolors.co. The super fast color schemes generator! [Internettside, sitert 05.05.2017]. Tilgjengelig fra: https://coolors.co/.

[4] Front and back ends [Internettside]. Wikipedia [sitert 08.05.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Front_and_back_ends.

[5] Scrum (software development) [Internettside]. Wikipedia [sitert 08.05.2017]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Scrum_(software_development).

[6] Bluetooth low energy beacon [Internettside]. Wikipedia [sitert 08.05.2017]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Bluetooth_low_energy_beacon.

[7] Callback (computer programming) [Internettside]. Wikipedia [sitert 08.05.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Callback_(computer_programming).

[8] Dependency injection [Internettside]. Wikipedia [sitert 22.02.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Dependency_injection.

[9] Service-oriented architecture [Internettside]. Wikipedia [sitert 08.05.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Service-oriented_architecture.

[10] Object-relational mapping [Internettside]. Wikipedia [sitert 22.02.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Object-relational_mapping.

[11] Qr code [Internettside]. Wikipedia [sitert 08.05.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/QR_code.

[12] Rational unified process: Overview [Internettside, sist oppdatert 2002; sitert 13.02.17]. Tilgjengelig fra: http://sce.uhcl.edu/helm/RationalUnifiedProcess/.

[13] Repository (version control) [Internettside]. Wikipedia [sitert 08.05.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Repository_(version_control).

[14] Representational state transfer [Internettside]. Wikipedia [sitert 08.05.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Representational_state_transfer.

[15] Sprint (software development) [Internettside]. Wikipedia [sitert 28.05.2017]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Sprint_(software_development).

[16] Unified modeling language [Internettside]. Wikipedia [sitert 08.05.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Unified_Modeling_Language.

[17] Use case [Internettside]. Wikipedia [sitert 08.05.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Use_case.

[18] Merriam-Webster. Workshop [Internettside]. Merriam-Webster [sitert 26.04.2017]. Tilgjengelig fra: https://www.merriam-webster.com/dictionary/workshop.

[19] Dashboards | android developers [Internettside, sitert 23.1.17]. Tilgjengelig fra: https://developer.android.com/about/dashboards/index.html.

[20] Part 1: Setting up [Internettside]. Estimote [sitert 18.04.2017]. Tilgjengelig fra: http://developer.estimote.com/android/tutorial/part-1-setting-up/.

[21] Mixpanel trends - mixpanel | mobile analytics [Internettside, sitert 23.1.17]. Tilgjengelig fra: https://mixpanel.com/trends/#report/ios_10.

[22] Vitensenteret - hjem [Internettside, sitert 23.1.17]. Tilgjengelig fra: http://vitensenteret.no/.

[23] Universell utforming [Internettside]. Wikipedia [sist oppdatert 20.3.16; sitert 23.1.17]. Tilgjengelig fra: https://no.wikipedia.org/wiki/Universell_utforming.

[24] Kodeklubben gjøvik [Internettside, sitert 26.1.17]. Tilgjengelig fra: http://kidsakoder.no/kodeklubb/gjovik/.

[25] Imt3102 - objektorientert systemutvikling [Internettside]. NTNU [sitert 26.1.17]. Tilgjengelig fra: https://www.ntnu.no/studier/emner/IMT3102#tab=omEmnet.

[26] Imt3672 - mobile development project [Internettside]. NTNU [sitert 26.1.17]. Tilgjengelig fra: https://www.ntnu.no/studier/emner/IMT3672/2016/1#tab=omEmnet.

[27] Sommerville, I. 2011. *Software engineering*. Pearson Education, Inc., Boston, 9th edition.

[28] Pointing poker [Internettside]. Matt Ruwe [sitert 23.1.17]. Tilgjengelig fra: https://www.pointingpoker.com/.

[29] Cohn, M. What are story points? [Internettside]. Mountain Goat Software [sitert 13.02.17]. Tilgjengelig fra: https://www.mountaingoatsoftware.com/blog/what-are-story-points.

[30] Fibonacci number [Internettside]. Wikipedia [sitert 13.02.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Fibonacci_number.

[31] Full featured web editing. featherweight download. [Internettside]. Ephox [sitert 24.04.2017]. Tilgjengelig fra: https://www.tinymce.com/.

[32] Javascript [Internettside]. Wikipedia [sist oppdatert 27.02.17; sitert 28.02.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/JavaScript.

[33] Polymer (library) [Internettside]. Wikipedia [sist oppdatert 08.02.17; sitert 27.02.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Polymer_(library).

[34] A collection of elements made by the polymer team [Internettside, sitert 28.02.17]. Tilgjengelig fra: https://www.webcomponents.org/collection/Polymer/elements.

[35] Node.js [Internettside]. Wikipedia [sitert 08.05.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Node.js.

[36] Java api for restful web services [Internettside]. Wikipedia [sitert 08.05.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services.

[37] Entity framework [Internettside, sitert 22.02.17]. Tilgjengelig fra: https://docs.microsoft.com/en-us/ef/.

[38] Stack overflow answer [Internettside]. Stack Owerflow [sitert 22.02.17]. Tilgjengelig fra: http://stackoverflow.com/questions/1279613/what-is-an-orm-and-where-can-i-learn-more-about-it.

[39] Swift vs objective-c: 5 reasons for using swift [Internettside, sitert 28.02.17]. Tilgjengelig fra: https://theappsolutions.com/blog/development/swift-vs-objective-c/.

[40] A., N. React native vs native ios/android [Internettside, sist oppdatert 25.01.2017; sitert 10.04.2017]. Tilgjengelig fra: https://www.coursereport.com/blog/so-you-want-to-build-a-mobile-app-react-native-vs-native-mobile.

[41] J., P. Myths about enterprise app development in modern scenario – native vs. cross-platform [Internettside, sitert 10.04.2017]. Tilgjengelig fra: http://www.configure.it/blog/mobile-app-development-native-vs-cross-platform-app/.

[42] Google. Material design - introduction [Internettside, sitert 10.04.2017]. Tilgjengelig fra: https://material.io/guidelines.

[43] Apple. ios human interface guidelines [Internettside, sitert 10.04.2017]. Tilgjengelig fra: https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/.

[44] K., A. Native vs cross-platform [Internettside]. Yalantis [sitert 10.04.2017]. Tilgjengelig fra: https://yalantis.com/blog/native-vs-cross-platform-app-development-shouldnt-work-cross-platform/.

[45] E., K. 99,6 prosent av mobilmarkedet er nå eid av to operativsystem [Internettside]. www.tek.no [sist oppdatert 17.02.2017; sitert 14.04.2017]. Tilgjengelig fra: https://www.tek.no/artikler/na-har-android-og-ios-tatt-nesten-i-hele-i-mobilmarkedet/376773.

[46] Estimote [Internettside]. Estimote [sitert 10.04.2017]. Tilgjengelig fra: http://estimote.com.

[47] Borowicz, W. How do beacons work? the physics of beacon tech [Internettside]. Estimote [sist oppdatert 02.01.2015; sitert 10.04.2017]. Tilgjengelig fra: http://blog.estimote.com/post/106913675010/how-do-beacons-work-the-physics-of-beacon-tech.

[48] Shirhatti, S. Publish to a linux production environment [Internettside]. Microsoft [sitert 12.03.2017]. Tilgjengelig fra: https://docs.microsoft.com/en-us/aspnet/core/publishing/linuxproduction.

[49] Tom Dykstra, Steve Smith, S. H. & Ross, C. Web server implementations in asp.net core [Internettside]. Microsoft [sitert 12.03.2017]. Tilgjengelig fra: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/.

[50] Nginx [Internettside]. Wikipedia [sitert 12.03.2017]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Nginx.

[51] Project, P. Polymer app toolbox [Internettside, sitert 13.03.17]. Tilgjengelig fra: https://www.polymer-project.org/2.0/toolbox/index.

[52] Project, P. Routing with <app-route> [Internettside, sitert 13.03.17]. Tilgjengelig fra: https://www.polymer-project.org/2.0/toolbox/routing.

[53] Project, P. Encapsulated routing with elements [Internettside, sitert 14.03.17]. Tilgjengelig fra: https://www.polymer-project.org/blog/routing.

[54] Single-page application [Internettside]. Wikipedia [sist oppdatert 23.02.17; sitert 13.03.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Single-page_application.

[55] Osmani, A. The prpl pattern [Internettside, sist oppdatert 13.02.17; sitert 14.03.17]. Tilgjengelig fra: https://developers.google.com/web/fundamentals/performance/prpl-pattern/.

[56] Lazy loading [Internettside]. Wikipedia [sist oppdatert 04.03.17; sitert 25.03.17]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Lazy_loading.

[57] PolymerElements. iron-ajax [Internettside, sitert 10.04.2017]. Tilgjengelig fra: https://www.webcomponents.org/element/PolymerElements/iron-ajax/iron-ajax.

[58] S., C. Building rest apis using asp.net core and entity framework core [Internettside]. chsakell's blog [sitert 12.03.2017]. Tilgjengelig fra: https://chsakell.com/2016/06/23/rest-apis-using-asp-net-core-and-entity-framework-core/.

[59] The repository pattern [Internettside, sitert 06.03.2017]. Tilgjengelig fra: https://msdn.microsoft.com/en-us/library/ff649690.aspx.

[60] Steve Smith, S. A. Dependency injection in asp.net core [Internettside]. docs.microsoft.com [sitert 22.02.17]. Tilgjengelig fra: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection.

[61] Entity framework [Internettside]. Wikipedia [sitert 12.03.2017]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Entity_Framework.

[62] What is code-first [Internettside]. Entity Framework Tutorial [sitert 12.03.2017]. Tilgjengelig fra: http://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx.

[63] Piros, T. Securing a rest api [Internettside]. marklogic.com [sitert 28.03.2017]. Tilgjengelig fra: https://developer.marklogic.com/blog/securing-a-rest-api.

[64] Parecki, A. Oauth 2 simplified [Internettside, sitert 28.03.2017]. Tilgjengelig fra: https://aaronparecki.com/oauth-2-simplified/.

[65] Chalet, K. openiddict-core [Internettside]. Github [sitert 10.04.2017]. Tilgjengelig fra: https://github.com/openiddict/openiddict-core.

[66] Rousos, M. Bearer token authentication in asp.net core [Internettside]. Microsoft [sitert 28.03.2017]. Tilgjengelig fra: https://blogs.msdn.microsoft.com/webdev/2016/10/27/bearer-token-authentication-in-asp-net-core/.

[67] A., M. Architectural pattern: Apple mvc's variation [Internettside, sitert 12.04.2017]. Tilgjengelig fra: http://www.albertomoral.com/2016/01/10/model-view-controller.html.

[68] Isn't cocoa mvc really mvp? [Internettside]. StackOverflow [sitert 12.04.2017]. Tilgjengelig fra: http://stackoverflow.com/questions/14943678/isnt-cocoa-mvc-really-mvp.

[69] E., G. ios design patterns [Internettside, sist oppdatert 04.09.2013; sitert 12.04.2017]. Tilgjengelig fra: https://www.raywenderlich.com/46988/ios-design-patterns.

[70] Apple. Protocol-oriented programming [Internettside]. Apple [sist oppdatert 2015; sitert 26/3-2017]. Tilgjengelig fra: https://developer.apple.com/videos/play/wwdc2015/408/.

[71] Why choose struct over class? [Internettside]. StackOverflow [sist oppdatert 15.06.2014; sitert 24.04.2017]. Tilgjengelig fra: http://stackoverflow.com/questions/24232799/why-choose-struct-over-class/24232845.

[72] Protocol-oriented programming in swift - apple wwdc 2015t [Internettside]. YouTube [sitert 12.03.2017]. Tilgjengelig fra: https://www.youtube.com/watch?v=g2LwFZatfTI.

[73] Model–view–controller [Internettside]. Wikipedia [sist oppdatert 31.03.2017; sitert 11.04.2017]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Model\T1\textendashview\T1\textendashcontroller.

[74] Model–view–presenter [Internettside]. Wikipedia [sist oppdatert 01.04.2017; sitert 11.04.2017]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Model\T1\textendashview\T1\textendashpresenter.

[75] Google. google/gson [Internettside]. Github [sitert 11.04.2017]. Tilgjengelig fra: https://github.com/google/gson.

[76] Shaikh, S. 14 websites and apps using google's material design [Internettside]. Code Condo [sitert 29.04.2017]. Tilgjengelig fra: http://codecondo.com/14-websites-and-apps-using-googles-material-design/.

[77] Google. Google inbox [Internettside]. Google [sitert 29.04.2017]. Tilgjengelig fra: https://inbox.google.com/.

[78] Difi. Kva seier forskrifta? | universell utforming [Internettside]. Difi [sitert 26.04.2017]. Tilgjengelig fra: https://uu.difi.no/krav-og-regelverk/kva-seier-forskrifta.

[79] Wikipedia. Materialdesign [Internettside]. Wikipedia [sist oppdatert 10.03.2017; sitert 26.04.2017]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Material_Design.

[80] Android. Design [Internettside]. Android [sitert 27.04.2017]. Tilgjengelig fra: https://developer.android.com/design/index.html.

[81] Google. Cards [Internettside]. Google [sitert 20.04.2017]. Tilgjengelig fra: https://material.io/guidelines/components/cards.html.

[82] Apple. Segmented controls - [Internettside]. Apple [sitert 20.04.2017]. Tilgjengelig fra: https://developer.apple.com/ios/human-interface-guidelines/ui-controls/segmented-controls/.

[83] www.materialdesignblog.com. Material design viewpager library for android [Internettside]. www.materialdesignblog.com [sist oppdatert 16.05.2015; sitert 20.04.2017]. Tilgjengelig fra: http://materialdesignblog.com/material-design-viewpager-library-for-android/.

[84] Difi. Wcag 2.0 - bruk av farger [Internettside]. Difi [sitert 20.04.2017]. Tilgjengelig fra: https://uu.difi.no/krav-og-regelverk/wcag-20-standarden/141-bruk-av-farge-niva.

[85] Kawaguchi, K. Meet jenkins [Internettside]. Jenkins-ci.org [sist oppdatert 07.01.2016; sitert 10.04.2017]. Tilgjengelig fra: https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins.

[86] Supervisor. Supervisor: A process control system [Internettside, sitert 03.05.2017]. Tilgjengelig fra: http://supervisord.org.

[87] What is docker? [Internettside]. opensource.com [sitert 10.04.2017]. Tilgjengelig fra: https://opensource.com/resources/what-docker.

[88] Microsoft. Sql server 2016 [Internettside, sitert 03.05.2017]. Tilgjengelig fra: https://www.microsoft.com/en-us/sql-server/sql-server-2016.

[89] Jason Roth, Luis Bosquez, B. K. Run the sql server vnext docker image on linux, mac, or windows [Internettside]. Microsoft [sist oppdatert 15.03.2017; sitert 10.04.2017]. Tilgjengelig fra: https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-setup-docker.

[90] Swagger [Internettside]. SmartBear [sitert 12.05.2017]. Tilgjengelig fra: http://swagger.io/.

[91] Lint (software) [Internettside]. Wikipedia [sist oppdatert 02.01.2017; sitert 16.04.2017]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Lint_(software).

[92] Improve your code with lint [Internettside]. Android [sitert 16.04.2017]. Tilgjengelig fra: https://developer.android.com/studio/write/lint.html.

[93] Sonarqube [Internettside]. Wikipedia [sist oppdatert 14.02.2017; sitert 17.04.2017]. Tilgjengelig fra: https://en.wikipedia.org/wiki/SonarQube.

[94] Sonarqube - the leading platform for continuous code quality [Internettside]. SonarSource [sitert 17.04.2017]. Tilgjengelig fra: https://www.sonarqube.org/.

[95] Sonarlint [Internettside]. SonarSource [sitert 17.04.2017]. Tilgjengelig fra: https://plugins.jetbrains.com/plugin/7973-sonarlint.

[96] A tool to enforce swift style and conventions [Internettside, sitert 20.04.2017]. Tilgjengelig fra: https://github.com/realm/SwiftLint.

[97] Measure swift code metrics and get reports in xcode, jenkins and other ci platforms. [Internettside, sitert 20.04.2017]. Tilgjengelig fra: https://github.com/yopeso/Taylor.

[98] Schneller, D. Why good metrics values do not equal good quality [Internettside]. CodeCentric. Tilgjengelig fra: https://blog.codecentric.de/en/2011/10/why-good-metrics-values-do-not-equal-good-quality/.

[99] A simple code complexity analyser without caring about the c/c++ header files or java imports, supports most of the popular languages. [Internettside, sitert 20.04.2017]. Tilgjengelig fra: https://github.com/terryyin/lizard.

[100] Sublime text [Internettside]. Wikipedia [sitert 17.04.2017]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Sublime_Text.

[101] Sublimelinter 3 [Internettside, sitert 17.04.2017]. Tilgjengelig fra: http://sublimelinter.readthedocs.io/en/latest/.

[102] Polylint [Internettside]. PolymerLabs [sitert 17.04.2017]. Tilgjengelig fra: https://github.com/PolymerLabs/polylint.

[103] Dodson, R. Polymer snippets for sublime [Internettside, sitert 17.04.2017]. Tilgjengelig fra: https://packagecontrol.io/packages/Polymer%20%26%20Web%20Component%20Snippets.

[104] Livereload for sublime text 3 [Internettside, sitert 17.04.2017]. Tilgjengelig fra: https://packagecontrol.io/packages/LiveReload.

[105] Omnisharp. Omnisharp [Internettside, sitert 28.04.2017]. Tilgjengelig fra: http://www.omnisharp.net.

[106] Github. Ohhttpstubs [Internettside]. Github [sist oppdatert 04.04.2017; sitert 11.04.2017]. Tilgjengelig fra: https://github.com/AliSoftware/OHHTTPStubs.

[107] Apple. User interface testing [Internettside]. Apple [sitert 02.05.2017]. Tilgjengelig fra: https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/testing_with_xcode/chapters/09-ui_testing.html.

[108] Developers, G. Testing ui for a single app [Internettside, sitert 26.04.2017]. Tilgjengelig fra: https://developer.android.com/training/testing/ui-testing/espresso-testing.html.

[109] Modern http benchmarking tool [Internettside]. Estimote [sitert 19.04.2017]. Tilgjengelig fra: https://github.com/wg/wrk.

[110] Lua (programming language) [Internettside]. Wikipedia [sitert 17.04.2017]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Lua_(programming_language).

[111] Amazon ec2 [Internettside]. Amazon [sitert 12.04.2017]. Tilgjengelig fra: https://aws.amazon.com/ec2/.

[112] Openstack [Internettside]. Openstack [sitert 12.04.2017]. Tilgjengelig fra: https://www.openstack.org.

[113] Microsft azure [Internettside]. Microsoft [sitert 12.04.2017]. Tilgjengelig fra: https://azure.microsoft.com/nb-no/free/?WT.srch=1&wt.mc_id=AID_SEM_B4prrWKd.

[114] Apache [Internettside]. Apache [sitert 12.04.2017]. Tilgjengelig fra: https://www.apache.org.

[115] Web server performance comparison [Internettside]. Dreamhost [sist oppdatert 23.12.2016; sitert 12.04.2017]. Tilgjengelig fra: https://help.dreamhost.com/hc/en-us/articles/215945987-Web-server-performance-comparison.

[116] Transport layer security [Internettside]. Wikipedia [sitert 17.05.2017]. Tilgjengelig fra: https://en.wikipedia.org/wiki/Transport_Layer_Security.

[117] Stoneman, E. Docker swarm mode walkthrough [Internettside]. Youtube [sist oppdatert 21.06.2016; sitert 12.04.2017]. Tilgjengelig fra: https://www.youtube.com/watch?v=KC4Ad1DS8xU.

[118] Apple. App review [Internettside]. Apple [sitert 14.04.2017]. Tilgjengelig fra: https://developer.apple.com/app-store/review/.

[119] www.readwrite.com. How google play security still falls short [Internettside]. www.readwrite.com [sist oppdatert 10.07.2015; sitert 14.04.2017]. Tilgjengelig fra: http://readwrite.com/2015/08/10/android-google-play-security-malware-scanner/.

[120] Apple. App store review guidelines [Internettside]. Apple [sitert 11.04.2017]. Tilgjengelig fra: https://developer.apple.com/app-store/review/guidelines/.

[121] Google. Developer content policy [Internettside]. Google [sitert 04.05.2017]. Tilgjengelig fra: https://play.google.com/intl/no_ALL/about/developer-content-policy/.

[122] Google. Brukergenerert innhold [Internettside]. Google [sitert 20.04.2017]. Tilgjengelig fra: https://play.google.com/intl/no_ALL/about/restricted-content/user-generated-content/.

[123] Google. Designed for families [Internettside]. Google [sitert 20.04.2017]. Tilgjengelig fra: https://play.google.com/intl/no_ALL/about/families/designed-for-families/program-requirements/.

[124] What is a story point [Internettside, sitert 23.1.17]. Tilgjengelig fra: https://agilefaq.wordpress.com/2007/11/13/what-is-a-story-point/.

A Prosjektplan

Prosjektplan

Viten i senter

Steinar Opphus	140174
Ole Andre Slettum	141329
Steffen Granberg	141647

NTNU Norges teknisk-naturvitenskapelige universitet

I samarbeid med:

vitensenteret INNLANDET

Våren 2017

Innhold

1 Mål og rammer

1.1 Bakgrunn

Vitensenteret Innlandet er et populærvitenskapelig opplevelses- og læringssenter innen matematikk, naturvitenskap og teknologi. Vitensenteret Innlandet er lokalisert på Gjøvik og er et av 10 offisielle vitensenter i Norge som etter oppdrag fra Kunnskapsdepartemenetet skal inspirere og lære barn og ungdom om realfag.

Senteret er åpent for publikum i helger og ferier, hvor både unge og voksne kan eksperimentere gjennom en rekke eksperimentstasjoner knyttet til emner som matematikk, astronomi, biologi, kjemi og landbruk.

Vitensenteret Innlandet ønsker nå å utvikle en mobilapplikasjon som vil guide og gi ekstra informasjon for de besøkende. Målet for applikasjonen er at den besøkende - ung som gammel - vil gå hjem med et ønske om å lære enda mer om realfag. Applikasjonen vil gi brukerne muligheten til å fortsette utforskningen også etter endt besøk.

1.2 Prosjektmål

Vi har valgt å dele målene for prosjektet inn i tre kategorier: resultatmål, effektmål og læringsmål. Resultatmål beskriver hva prosjektet konkret skal oppnå og er knyttet til prosjektets resultater og leveranser. Effektmål beskriver målsettingen til prosjektet og hvorfor vi har satt igang med det. Læringsmål er hva vi som studenter håper å lære av prosessen og oppgaven.

Resultatmål

Målet er å lage en ferdig mobilapplikasjon for iOS og Android, i tillegg til et webgrensesnitt med tilhørende server. Dette skal være fungerende komponenter ved prosjektslutt som skal kunne settes i drift av Vitensenteret uten større modifikasjoner, som innebærer at modulene skal kunne kjøres på servere hos Vitensenteret.

Effektmål

Målet er å gi besøkende på Vitensenteret en ekstra dimensjon til besøket. Gjennom eksperimentene prøver Vitensenteret å treffe så mange som mulig. Vi ser derimot at noen ønsker mer og dypere informasjon, mens andre liker mindre tekst og nøyer seg med å prøve seg frem. En mobilapplikasjon vil gi besøkende muligheten til å utforske temaene dypere på senteret eller hjemme etter endt besøk gjennom digitale versjoner av spill og eksperimenter.

Konkret har Vitensenteret som mål at applikasjonen skal:

- Øke antall besøkende til senteret i helger og ferier.
- At besøkende skal komme igjen flere ganger.
- At besøkende skal utforske eksperimentene hjemme etter endt besøk.

Læringsmål

Vi ønsker å lære mest mulig om å arbeide i team ved utvikling av nye systemer. Fokuset vårt vil være på:

- Lære bruk og nytten av flest mulig verktøy.
- Bruke SCRUM-metodikk i et reelt prosjekt.
- Lære oss bruk av testing og kvalitetskontroll
  - Forstå viktigheten av testing.
  - Bruke unittester på viktige komponenter.
  - Bruke kvalitetssikringsverktøy som øker kvaliteten på koden.
  - Gjennomføre og se nytten av brukertester.
- Lære oss hvordan vi kan lage intuitive grafiske brukergrensesnitt.

1.3 Rammer

- Webgrensesnittet skal fungere på nettlesere som støtter HTML5. Dette vil si nyere versjoner av Chrome, Firefox, Safari og minimum Internet Explorer 9.
- Android applikasjonen skal fungere fra og med telefoner med API level 17.
- iOS applikasjonen skal fungere fra og med telefoner med iOS 8.0.
- Servere skal settes opp slik at det i fremtiden kan settes opp på en lokal server hos Vitensenteret.

2 Omfang

2.1 Fagområde

Noe av det viktigste på Vitensenteret er eksperimentstasjonene. Interaksjon mellom applikasjon og eksperiment er en av de viktigste funksjonene i løsningen. Eksperimenter er knyttet til ulike tema innenfor realfag. Eksperimentene søker å formidle kunnskap gjennom lek og moro.

”Mindball” er et eksempel på en eksperimentstasjon. Mindball er et eksperiment hvor en ball starter på midten av et bord, to deltakere spiller mot hverandre og man skal prøve å få ballen over til motstanderen. Deltagerne bruker et headsett som måler hjerneaktivitet. Jo lavere hjerneaktivitet en deltager har, jo sterkere trekker ballen mot motstanderens sone. For å senke hjerneaktiviteten kan man bruke teknikker som å fokusere på pusten eller et punkt i veggen.

Et eksempel på en interaksjon mellom applikasjonen og Mindball er at man starter med å skanne eksperimentet, videre kan brukeren få hint om hvordan han kan bli bedre i spillet og informasjon om hvordan slike teknikker fungerer fysiologisk.

Applikasjonen vi skal utvikle søker hovedsaklig å lage en god base for at pedagoger og ansatte på Vitensenteret skal kunne digitalisere eksperimentene som er på senteret. Hovedfokuset for utviklerne er strukturen i løsningen, ikke det pedagogiske rundt læring. Den pedagogiske delen vil det være ansatte på Vitensenteret som utarbeider.

Oppgaven vår dekker en rekke teknologier innenfor programmering, disse inkluderer:

- Responsive webapplikasjoner med bruk av HTML, CSS og Polymer
- RESTFul backend API utviklet i .NET Core
- Fleksible og moderne mobilapplikasjoner
- Unit testing
- Databasemodellering og databasedesign
- Oppsett av server og serveradministrasjon
- Internasjonalisering av applikasjoner
- QR scanning og sensorbruk
- Grafiske brukergrensenitt og ergonomi

2.2 Avgrensning

Vi skal som utviklere ikke ha ansvaret for det pedagogiske opplegget rundt hvert enkelt eksperiment, men her høre og støtte oss til pedagogene på Vitensenteret Innlandet.

2.3 Oppgavebeskrivelse

Vår oppgave er å levere en mobil applikasjon som besøkende på Vitensenteret Innlandet kan bruke både under og etter besøket. Det skal også utvikles et webgrensesnitt som gjør det mulig for ansatte ved Vitensenteret å oppdatere, slette og endre informasjon på en enkel måte. Mobilapplikasjonen skal ha følgende funksjonalitet:

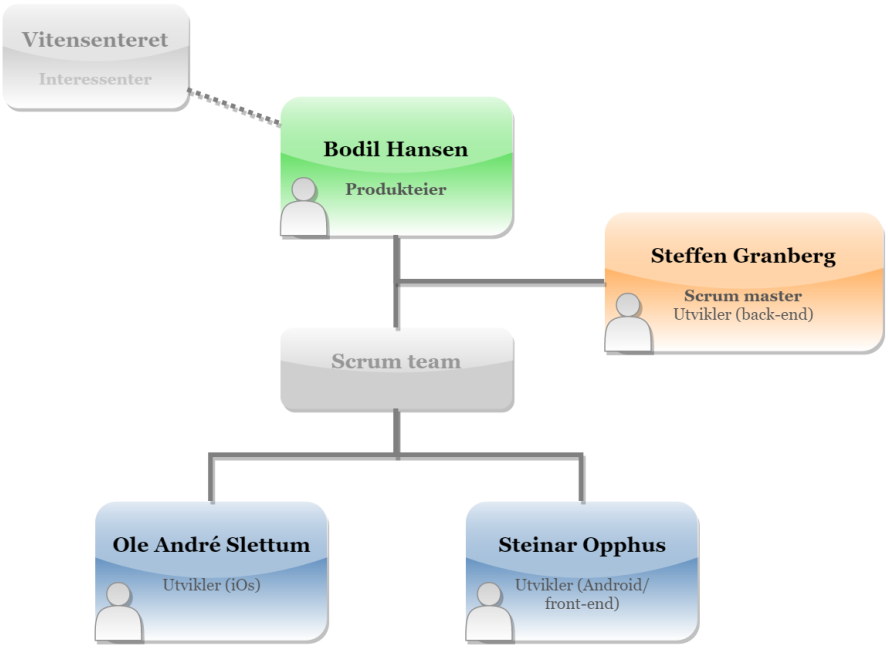
- Brukeren skal kunne skanne et eksperiment og få informasjon om muligheter knyttet til dette eksperimentet.
  - Informasjon om eksperimentet som navn, bruk og hint.
  - Være mulig å markere eksperimentet som gjennomført, og i enkelte tilfeller registrere poeng.
  - Bruke telefonen som en del av eksperimentet, for eksempel ved å bruke sensorer
  - Få tilgang til enkelte eksperimenter og spill hjemmefra.
- Mulighet til å gjennomføre en quiz der spørsmålene er knyttet til eksperimenter eller temaer på Vitensenteret. Spørsmål vil variere slik at besøkende kan gjøre quizen flere ganger.
- Informasjonsside med informasjon om Vitensenteret.
- Topplister for quiz - daglig, ukentlig og månedlig.
- Bruke sensorene i telefonen for å legge til en ekstra dimensjon på enkelte av eksperimentene.

Vi skal også lage et webgrensesnitt som skal brukes for å administrere all informasjon og funksjonalitet i appen. Dette skal ha følgende funksjonalitet:

- Legge til en ny eksperimentstasjon
- Legge til og endre informasjon om eksperimentstasjonene
  - Informasjon om stasjonen
  - Hvordan stasjonen brukes og eventuelle hint
  - Plassering av stasjonen i senteret - eksperimentstasjonene flyttes av og til rundt på senteret.
  - Legge til, eller endre bilder knyttet til stasjonen.
- Se statistikk over bruk av eksperimentstasjoner, aktiviteter og besøk på senteret.
- Opprette og endre informasjon om de ulike områdene/rommene i senteret.
- Legge inn eller endre bilder knyttet til de forskjellige områdene i senteret.

3 Prosjektorganisering

3.1 Organisasjonskart



Figur 1: Organisasjonskart

3.2 Ansvarsforhold og roller

Scrum-teamet er organisert med vanlige roller og ansvarsforhold:

- Produkteier (Bodil Hansen)
  - Hun skal representere interessentene og synliggjøre Vitensenterets visjon under prosjektet.
  - Hun er vår hovedkontaktperson på Vitensenteret, og vil være tilstede på alle *sprint review*-møter.
  - Alle ansatte på Vitensenteret er likevel velkomne på disse møtene, og kan komme med innspill på lik linje med Bodil.
- Scrum master (Steffen Granberg)
  - Sørge for at produkteier forstår sin rolle.
  - Være bindeledd mellom utviklere og produkteier.
  - Sørge for at utviklingen går i riktig retning og tempo ved å iverksette tiltak hvis nødvendig.
  - Innkalle til nødvendige møter og lede disse, enten det er sprint planning, review, retrospective eller daily.

- Utviklere (Steinar Opphus og Ole André Slettum)
  - Utføre sine oppgaver i henhold til fastsatte rutiner og kodestandard.

3.3 Rutiner og regler i gruppen

Grupperegler

Gruppereglene i sin helhet legges ved i selve bacheloroppgaven. Noen viktige punkter:

- Hvert gruppemedlem skal ha som mål å jobbe minimum 30 timer per uke.
- Gruppemedlemmene skal minst ha et fysisk møte i løpet av en uke, men har som felles mål å sitte 4 dager sammen per uke.
- Timene skal loggføres og legges frem for gruppen på prosjektmøter.
- Møteplikt på prosjektmøter, statusmøter, møter med arbeidsgiver og møter med veileder.
- Det skrives referat etter alle prosjekt- og statusmøter. Steinar er satt som referent.

4 Planlegging, oppfølging og rapportering

4.1 Hovedinndeling av prosjektet

Prosjektet vårt består av 3 hoveddeler: mobilapplikasjon (iOS og Android), webgrensesnitt og et RESTful API. Utviklerne vil få hovedansvar for hver sin del.

Valg av SU-modell/prosessrammeverk

Ettersom Vitensenteret har gitt oss en flytende kravspesifikasjon paret med at kravene vil endres underveis så gir en smidig utviklingsmetodikk oss fleksibiliteten vi trenger. Vi skal samarbeide tett med styringsgruppen på Vitensenteret, jevnlige møter og oppdateringer vil kontinuerlig påvirke kravene til prosjektet. Vi ønsker samtidig en prosess med faste rutiner og struktur. Vi har derfor valgt å bruke SCRUM som utviklingsmetodikk. Vi har brukt SCRUM som metodikk i tidligere prosjekter og har lært mye av det. Disse erfaringene tar vi med oss inn i dette prosjektet.

I prosessen vil artefakter i SCRUM bli supplert med artefakter tradisjonelt brukt i RUP-metodikken[1]. Dette for å sikre god dokumentasjon av prosessen. Use case vil bli brukt til å identifisere og dokumentere funksjonalitet og system-sekvensdiagram vil bli brukt for å illustrere utvidede use case. I tillegg til use case skal det utarbeides risikoanalyser. Bruk av disse artefaktene sammen med SCRUM sine krav til jevnlige møter og rapportering gir oss en god plattform for å gjennomføre prosjektet på en smidig men samtidig strukturert måte.

Vi valgte å ha en sprintlengde på to uker, sprintstart er tirsdag og sprintslutt er mandag to uker etterpå. Lengre sprinter på et så kort prosjekt følte vi ble feil, siden det blir færre muligheter for produkteier å komme med innspill og styre oss inn på ønsket kurs, og generelt en mindre smidig arbeidsmetode. Kortere sprinter ville sannsynligvis ha ført med seg for mye overhead i form av tid som går bort på sprint *planning*, *reviews* og lignende.

Før hver sprint skal vi ha et sprint planning meeting hvor vi planlegger og identifiserer arbeidsoppgaver for den kommende sprinten. Etter hver sprint vil det gjennomføres møte med arbeidsgiver hvor vi presenterer det vi har jobbet med. Dette er også en mulighet for Vitensenteret til å uttrykke ønsker og gi oss tilbakemeldinger. Slik sørger vi for at alt vi utvikler er forankret hos arbeidsgiver. Når utviklerne møtes vil det bli gjennomført et daglig stand-up møte, der diskuterer vi hva vi jobber med og eventuelle problemer vi har som må løses for å opprettholde progresjon i arbeidet. Det vil variere hvor mange ukentlige stand-up møter vi gjennomfører da vi ikke sitter sammen og jobber hver dag.

4.2 Plan for statusmøter

Vi har fast møte med arbeidsgiver etter hver sprint annenhver mandag 14.30-15.30. Her skal det vises live demo av hvordan systemet har utviklet seg siden sist. Disse møtene kan avlyses av gruppen, eller av arbeidsgiver hvis møtet av ulike grunner ikke kan gjennomføres.

Hver onsdag kl. 09.00 har vi møte med veileder. Utover i prosjektet kan disse møtene være sjeldnere, møtehyppighet bestemmes kontinuerlig i samarbeid med veileder.

5 Organisering av kvalitetssikring

5.1 Dokumentasjon, lagring og kildekode

Vi skal ha høyt fokus på gode rutiner underveis i hele prosjektet, og bruke alt vi har lært om profesjonell systemutvikling. En stor del av dette er å følge standarder og konvensjoner for de programmeringsspråkene og utviklingsverktøyene som involveres i prosjektet (utskrift av alle kodestandarder legges ved selve bacheloroppgaven).

SonarQube

SonarQube er et verktøy for kontinuerlig analyse av kodekvalitet[2], og støtter de språkene vi har tenkt å benytte i prosjektet, inkludert JavaScript, C#, Java og Swift. Dette verktøyet skal brukes aktivt med mål om å unngå duplisering av kode, for kompleks kode, potensielle bugs, dårlig struktur og mangel på dokumentering og kode som ikke følger det gjeldende språkets kodestandard.

Vi skal sette opp en egen server for å kjøre SonarQube i stedet for å gjøre det lokalt, for å gjøre det enklere å samarbeide med retting av *issues*.

Testing

Kode skal skrives med tanke på at det skal være mulig å kjøre tester. Unit testing skal implementeres på kritiske komponenter, og enkelte trivielle hvis det er nok tid.

Vi skal også kjøre integrasjonstester for å kontrollere at RestAPI'et kommuniserer riktig med andre komponenter, og at dataene fremstilles riktig.

Vi planlegger å gjennomføre brukertester med de ansatte på Vitensenteret, samt tilfeldige besøkende på senteret underveis i prosjektet.

Språk	Rammeverk
Android	JUnit tests
C#	Integrert i .NET Core
iOS	Integrert i Swift
JavaScript	Jasmine

Internasjonalisering

Produkteier har lagt frem et ønske om at applikasjonene skal ha mulighet til å støtte flere språk.

Vi følger retningslinjene for internasjonaliseringsstøtte for hver enkelt plattform.

Dokumenter og lagring

- All prosjektkode ligger i sine respektive repositories.
- Rapporten skrives i $\LaTeX$ med online-verktøyet ShareLatex[3].
- $\LaTeX$-kildekode synkroniseres mellom ShareLatex og DropBox for ekstra sikkerhet.
- Utkast til dokumenter som møtereferater, kontrakter og lignende, lagres i delt katalog på Google Disk.

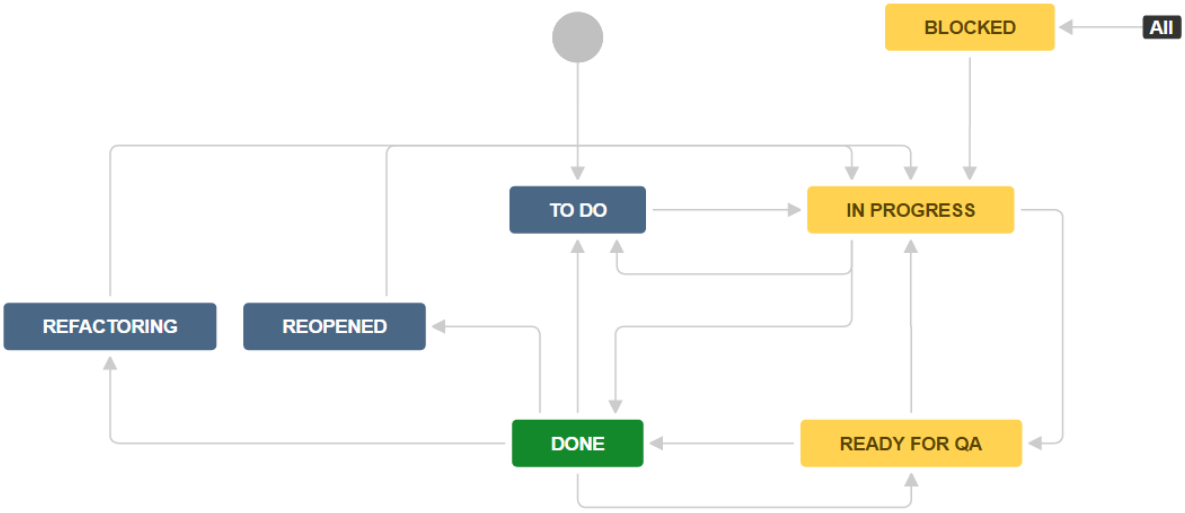
5.2 Faste utviklingsrutiner

- Hver enkelt arbeidsoppgave skal defineres og legges i *backlog* i Jira.
- *Story points* brukes for å estimere tidsbruk på hver oppgave.
- Vi vil bruke tre nivåer med *branching*; *master - developer - issue*. Hver *issue* i Jira jobbes på i egen *branch*, og merges til *developer* når jobben er gjort. I slutten av hver sprint, merges *developer* til *master*.
- Hver *commit* må inneholde *issue*-ID fra Jira, slik at det er mulig å se historikk på hver *issue*.
- Hver commit-beskjed skal i tillegg være informativ og lett forståelig.
- Kode skal *committes* jevnlig, spesielt når en ny funksjon eller klasse er implemenert.
- Hver klasse og funksjon skal kommenteres etter gjeldende språks retningslinjer.
- All ny kode skal kommenteres før den *committes*.
- Klasse-, funksjons- og variabelnavn skal være informative og følge retningslinjene for det gjeldende kodespråk.

Issues og commits

- Issues i sprintene branches ut i Jira slik at navnet på branchene inneholder issue-number som videre kan brukes til å linke commit i Bitbucket og issue i Jira
- Når en utvikler jobber med en issue og ser seg ferdig, så skal man evaluere kodekvaliteten ved å bruke SonarQube samt gjøre en grov test av funksjonaliteten man har utviklet. Critical-feil bør helst utarbeides, eventuelt så må man suppresse feilen og dokumentere hvorfor man suppresser. Videre merger man issuen inn i Developer-branchen.
- Etter hver sprint skal alle utviklerne samarbeide om å teste Developer-branchen før den merges inn i Master-branchen. Det vil da bli testet både funksjonalitet, robusthet og kodekvalitet ved bruk av SonarQube.

Arbeidsflyt i Jira



Figur 2: Arbeidsflyt i Jira

Status	Beskrivelse
To do	Ikke påbegynte oppgaver
In progress	Oppgaver under arbeid.
Ready for QA	Oppgave ferdig. Klar for kvalitetssikring.
Done	Oppgave ferdig.
Reopened	Ferdig oppgave som gjenåpnes for videre utvikling.
Refactoring	Ferdig oppgave som gjenåpnes for refaktorering.
Blocked	Oppgave blokkert. Kan ikke fortsette arbeid før en annen oppgave er ferdig

5.3 Verktøy

Navn	Type	Bruksområde
ShareLaTeX	Redigeringsverktøy og kompilator for LaTeX-dokumenter	Prosjektrapport
Jira	Digitalt Scrum board og issue tracking	Prosjektstyring
Bitbucket	Hosting-tjeneste for kildekode etter Git-standarden	Versjonskontroll
Android Studio	Offisiel IDE for Android-utvikling	Androidapplikasjon
draw.io	Nettverksdiagram	Flowcharts, UML og andre diagrammer
Xcode	IDE for iOS-utvikling	iOS-applikasjon
Visual Studio Code	IDE for utvikling av back-end (C#)	Back-end
Docker	Software containerization platform	Lage konteiner til backend-API
SonarQube	Plattform for forbedring av kodekvalitet	Kodeinspeksjon
Amazon EC2 Cloud	IaaS	Hosting av backend og SonarQube-server
TeamGantt	Online verktøy for Gantt-skjemaer	Gantt-skjemaer
Toggl	Online verktøy for timeregistrering	Timeregistrering
Bower	Package manager for web-applikasjoner	Frontend
Yeoman	Generer maler for web-applikasjoner	Frontend
Gulp	Automatisering av javascript-oppgaver	Frontend
Pointing Poker	Online planning poker	Sprint planning

5.4 Risikoanalyse

Under følger en tabell som analyserer ulike risikoer knyttet til prosjektet - sannsynlighet for at hendelsen inntreffer og eventuelle konsekvenser. Sannsynlighet er delt inn i usannsynlig, sannsynlig og svært sannsynlig. Konsekvensen av en hendelse er delt inn i uproblematisk, problematisk og kritisk. Under følger identifiserte risikohendelser:

Nummer	Risiko	Sannsynlighet	Konsekvens	Tiltak
1	Prosjektet er ikke ferdig til deadline. Forsinkelser i prosjektet kan ha mange årsaker - teknologiske problemer, tap av backup, feilvurdering av tidsbruk og mer.	Usannsynlig	Kritisk	Ja
2	Store endringer i kravspesifisering fra kunden underveis i prosjektperioden	Sannsynlig	Uproblematisk	Ja
3	Applikasjonen er ikke tilstrekkelig universelt utformet som kan lede til færre brukere	Sannsynlig	Kritisk	Ja
4	En eller flere av utviklerne blir syk over en lengre periode eller slutter under prosjektet.	Usannsynlig	Kritisk	Ja
5	Tap av rapport eller kildekode	Usannsynlig	Kritisk	Ja
6	Andre firmaer utvikler lignende løsninger	Svært sannsynlig	Uproblematisk	Nei
7	Vitensenteret skrinlegger prosjektet	Usannsynlig	Kritisk	Nei
8	Problemer med kommunikasjon mellom eksperimentstasjoner på senteret og mobilapplikasjon	Svært sannsynlig	Problematisk	Ja
9	Noen av ønskene til funksjonalitet kan ikke implementeres fordi det ikke er teknisk mulig	Sannsynlig	Problematisk	Ja
10	Eksperimentene fungerer ikke og applikasjonen kan derfor ikke samhandle med de	Svært sannsynlig	Problematisk	Ja

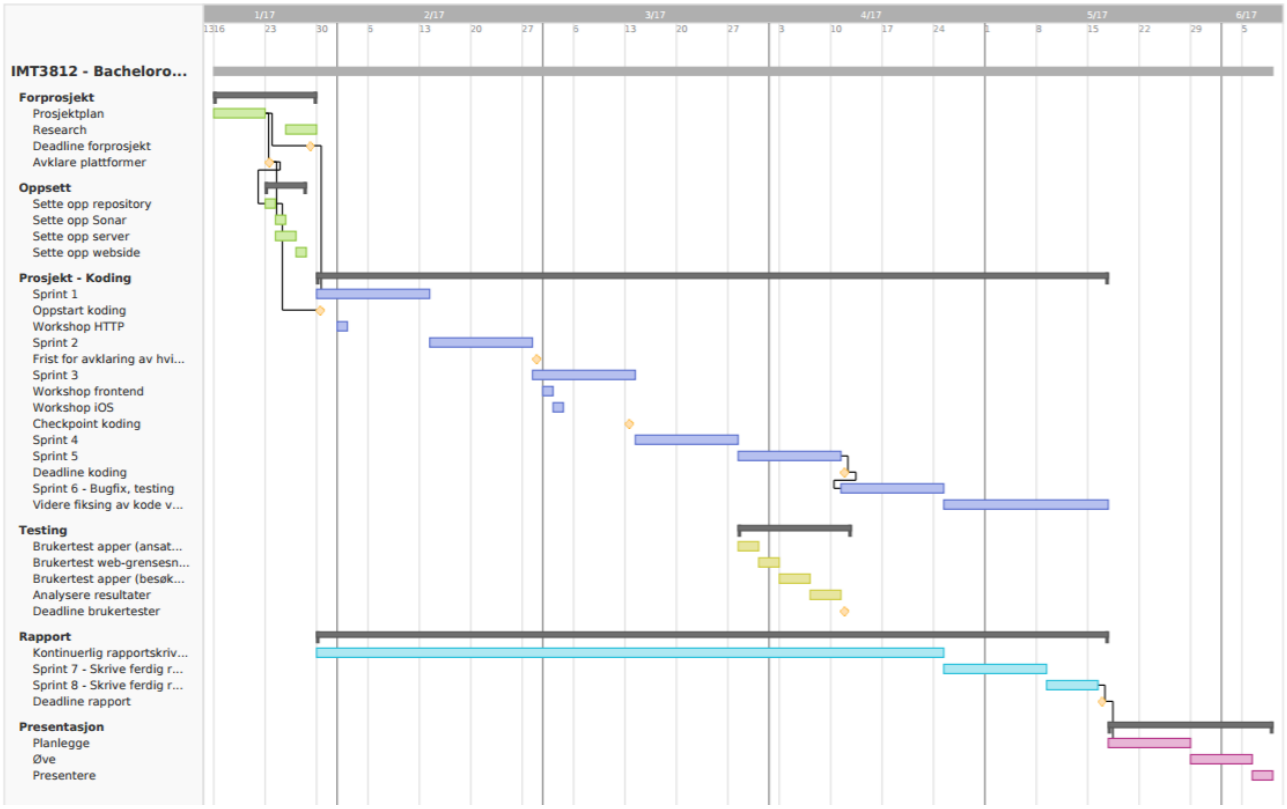
5.5 Plan for håndtering av risiko

Det er valgt åtte risikohendelser hvor det diskuteres hvilke tiltak man kan innføre for å redusere eventuelle negative konsekvenser av at hendelsene inntreffer. Dette er tiltak som reduserer sannsynlighet og/eller negative konsekvenser av slike hendelser.

Nr	Tiltak
1	Hvis vi ser at produktet vi skal levere blir forsinket, så må vi først og fremst informere produkteier. Man kan da i fellesskap bli enige om noe av den planlagte funksjonaliteten kan fjernes fra kravspesifiseringen. For å redusere sannsynligheten for forsinkelser, må vi underveis forsikre oss om at prosjektet ikke er forsinket. Hvis man ser at produktet er forsinket i henhold til den planlagte prosjektplanen, så må dette tas tak i tidlig. Det må utarbeides en god plan for hvordan man må jobbe for at produktet igjen skal være ajour i henhold til prosjektplanen.
2	Det er viktig med god forståelse mellom produkteier og utviklere om hvilke funksjonalitet som ønskes og hva produkteier forventer av det ferdige produktet. Vi kan diskutere ideer og drøfte hva som er mulig teknologisk. Gjennom prosjektet er det viktig at vi møter produkteier ofte for å vise hvordan prosjektet ligger ann. Hvis man møtes ofte er det lettere for Vitensenteret å uttrykke sine ønsker, og det vil være lettere for prosjektgruppen å gjøre kontinuerlige endringer.
3	Det er viktig å anvende forskning rundt universell utforming når man tar valg knyttet til grensesnittet. Vi skal følge standardene til Google og Apple i designet av Android og iOS-applikasjonene.
4	For å minimere konsekvensen av sykdom hos prosjektdeltakerne vil vi gjennom hele prosjektperioden ha egne dager hvor hver av oss presenterer det vi har gjort, og gir opplæring i syntaks og logikk i prosjektene vi jobber med. Dette vil minimere risikoen for tap av fremgang hvis en prosjektdeltaker blir syk eller må slutte i prosjektet.
5	For å unngå tap av rapport så lagres rapporten både på ShareLatex og Dropbox. Kildekode ligger i repository på bitbucket, lokalt hos alle utviklere og på en ekstern server i Amazon EC2 cloud.
8	Det er viktig tidlig i prosessen å identifisere hvordan teknologien rundt et eksperiment virker. Etter å ha gjort dette så kan vi drøfte om kommunikasjon mellom mobil og eksperiment er mulig, og gjøre et tidsestimat for hvor lang tid det eventuelt vil ta å utvikle interaktivitet mellom mobil og eksperiment. Hvis systemet for et eksperiment har en ekstern leverandør så er det hensiktsmessig å kontakte leverandøren tidlig, og undersøke mulighetene for å gjøre spesialtilpasninger.
9	Det er viktig at utviklerne tidlig i prosessen synliggjør hva som er teknologisk mulig å implementere. Forventninger fra kunden må håndteres på den måten at forventninger til det ferdige produktet er realistiske.
10	Hvis kommunikasjon mellom mobil og eksperiment feiler så bør brukeren informeres gjennom en feilmelding. Applikasjonen kan videre lagre feillogger slik at ansatte på Vitensenteret varsles om feil.

6 Plan for gjennomføring

6.1 Gantt-skjema



Figur 3: Gantt-skjema

Tabell 1: Gantt-skjema som tekst

i	Name/Title	Start Date	End Date	Predecessors
1	IMT3812 - Bacheloroppgave	2017-01-16	2017-06-07	
1.1	Forprosjekt	2017-01-16	2017-01-28	
1.1.1	Prosjektplan	2017-01-16	2017-01-20	
1.1.2	Research	2017-01-25	2017-01-27	
1.1.3	Deadline forprosjekt	2017-01-28	2017-01-28	1.1.1
1.1.4	Avklare plattformer	2017-01-23	2017-01-23	1.1.1
1.2	Oppsett	2017-01-23	2017-01-26	
1.2.1	Sette opp repository	2017-01-23	2017-01-23	1.1.4
1.2.2	Sette opp Sonar	2017-01-24	2017-01-24	1.1.4
1.2.3	Sette opp server	2017-01-24	2017-01-25	
1.2.4	Sette opp webside	2017-01-26	2017-01-26	
1.3	Prosjekt - Koding	2017-01-30	2017-05-16	
1.3.1	Sprint 1	2017-01-30	2017-02-13	1.1.3
1.3.2	Oppstart koding	2017-01-30	2017-01-30	1.2.1
1.3.3	Workshop HTTP	2017-02-01	2017-02-01	
1.3.4	Sprint 2	2017-02-14	2017-02-27	
1.3.5	Frist for avklaring av hvilke eksperimenter	2017-02-28	2017-02-28	
1.3.6	Sprint 3	2017-02-28	2017-03-13	
1.3.7	Workshop frontend	2017-03-01	2017-03-01	
1.3.8	Workshop iOS	2017-03-02	2017-03-02	
1.3.9	Checkpoint koding	2017-03-13	2017-03-13	
1.3.10	Sprint 4	2017-03-14	2017-03-27	
1.3.11	Sprint 5	2017-03-28	2017-04-10	
1.3.12	Deadline koding	2017-04-11	2017-04-11	1.3.11
1.3.13	Sprint 6 - Bugfix, testing	2017-04-11	2017-04-24	1.3.12
1.3.14	Videre fiksing av kode ved behov	2017-04-25	2017-05-16	
1.4	Testing	2017-03-28	2017-04-11	
1.4.1	Brukertest apper (ansatte)	2017-03-28	2017-03-29	
1.4.2	Brukertest web-grensesnitt (ansatte)	2017-03-30	2017-03-31	
1.4.3	Brukertest apper (besøkende)	2017-04-03	2017-04-05	
1.4.4	Analysere resultater	2017-04-06	2017-04-10	
1.4.5	Deadline brukertester	2017-04-11	2017-04-11	
1.5	Rapport	2017-01-30	2017-05-16	
1.5.1	Kontinuerlig rapportskriving	2017-01-30	2017-04-24	
1.5.2	Sprint 7 - Skrive ferdig rapport	2017-04-25	2017-05-08	
1.5.3	Sprint 8 - Skrive ferdig rapport	2017-05-09	2017-05-15	
1.5.4	Deadline rapport	2017-05-16	2017-05-16	1.5.3
1.6	Presentasjon	2017-05-17	2017-06-07	
1.6.1	Planlegge	2017-05-17	2017-05-26	1.5.4
1.6.2	Øve	2017-05-29	2017-06-05	
1.6.3	Presentere	2017-06-06	2017-06-07	

6.2 Milepæler og beslutningspunkter

Vi har fastsatt en rekke milepæler utover i prosjektet. Disse definerer når vi ønsker at ting skal være ferdige. Dette gjør at vi på jevnlige tidspunkter kan kvalitetsikre at vi ligger etter planen.

- **Milepæl 1, 23. januar:** Avklare plattformer
  - Bestemme oss for native eller cross-platform etter grundig research.
- **Milepæl 2, 28. januar:** Deadline forprosjekt
  - Ferdig med forprosjekt, oppsett av utviklingsmiljø og servere.
- **Milepæl 3, 30. januar:** Oppstart koding
  - Start av koding.
- **Milepæl 4, 28. februar:** Frist for avklaring av hvilke eksperimenter
  - Frist for å avgjøre i samarbeid med produkteier hvilke eksperimenter som skal inn i applikasjonene.
- **Milepæl 5, 13. mars:** Checkpoint koding
  - Grunnkomponentene på plass i iOS. Dette vil si kommunikasjon med backend, QR-skanning og eskperimentside.
  - Webgrensesnittet fungerende, og mangler bare finpuss. Det skal kunne legges til og endre eksperimenter, legge inn bilder og lage quizer.
  - Backend fungerer, og mangler bare finpuss. Alle komponenter støtter POST, PATCH og DELETE.
- **Milepæl 6, 28. mars:** Brukertester ferdig
  - Være ferdig med brukertester og påfølgende analyse.
  - Avklare hvilke endringer som skal gjøres som resultat av testene.
- **Milepæl 7, 11. april:** Koding ferdig og rapport godt underveis.
- **Milepæl 8, 16. mai:** Deadline rapport.

Referanser

[1] Rational Unified Process [wikipage]; 2016 [cited 26/10-2016]. Available from: https://no.wikipedia.org/wiki/Rational_Unified_Process.

[2] SonarQube: For Continuous Code Quality [webpage]; 2016 [cited 16/1-17]. Available from: https://www.sonarqube.org/.

[3] ShareLatex [webpage]; 2016 [cited 16/1-17]. Available from: https://www.sharelatex.com.

B Forprosjekt

B.1 Mappe 3

IMT3102 - Objektorientert systemutvikling

Prosjektrapport mappe 3

Steinar Opphus	140174
Ole Andre Slettum	141329
Ingvild Næss	248258
Steffen Granberg	141647

NTNU
Norges teknisk-naturvitenskapelige universitet

I samarbeid med:

vitensenteret INNLANDET

Høsten 2016

Contents

1 Goals and Boundaries

1.1 Background

Vitensenteret Innlandet is a science centre located in Gjøvik. It is one of 10 official science centres in Norway which has been given a task by the Ministry of Education and Research to inspire and teach children about the wonders of science.

The centre is open to the public on weekends and have several floors full of experiments designed to educate and inspire people of all ages to learn more about science - such as math, biology, astronomy and agriculture.

Vitensenteret Innlandet now wants to develop an mobile application that will guide and provide extra features when a person visits the science centre.

The goal of this application is to help the visitor - both young and adult - to learn and explore even more. It will also let the user continue with the exploration and learning experience at home.

1.2 Discipline / Problem area

To be able to fulfil their mandate of inspiring people to learn about science, the science centres need them to come visit, but they also need them to want to repeat their visit as often as possible. In a typical visit you will never be able to explore everything, but we see that many don't use the science centre nearly as often as we think they can.

By talking to some visitors and asking them what they are missing, Vitensenteret Innlandet often hear that the parents wonder how they can inspire and trigger their kids to explore more, but that they themselves lack the knowledge and experience of science and don't know what to do. If the parents had a way to get more information about the possibilities in the centre, what to ask the children during a visit, as well as ideas for experiments to do at home, some would feel it easier to engage and inspire curiosity.

Vitensenteret Innlandet has much to offer, but some feedback give the impression that it requires a bit of effort by the visitors themselves to really see the extent of the possibilities. Some experiments can be difficult to understand with the information on hand, even though the weekend staff helps where they can.

There is no guide apps for science centres that we know of, but we know that there is certain apps being developed for similar leisures like museums.

1.3 Project description

Our assignment is to deliver a mobile application that visitors at Vitensenteret Innlandet can use during a visit. The application will include the following functions:

- Allow the user to scan a experiment and get certain information and possibilities
  - Information about the station. Such as name, usage and hints
  - Mark the experiment as done, and in some cases register your score
  - Use the phone as a part of the experiment
  - Access to certain games and experiments at home
- User can make his own profile so he can store information about his activity at the centre (leader boards)

  - Be able to mark an station as completed and gain points

- Take a quiz involving questions around the science centre. The questions should not be the same and you should be able to show different questions each time.
- Info page with information about the science centre and location directions.
- Leader boards for quiz - daily, weekly, monthly, all-time
  - Use sensors in the phone to add an additional dimension/feature to the station

We will also develop a web application used for administrating application content. The administration system will have the following functions:

- Add new station
- Edit station information
- See different statistics
- Add question to question pool

2 Project Organization

2.1 Org chart

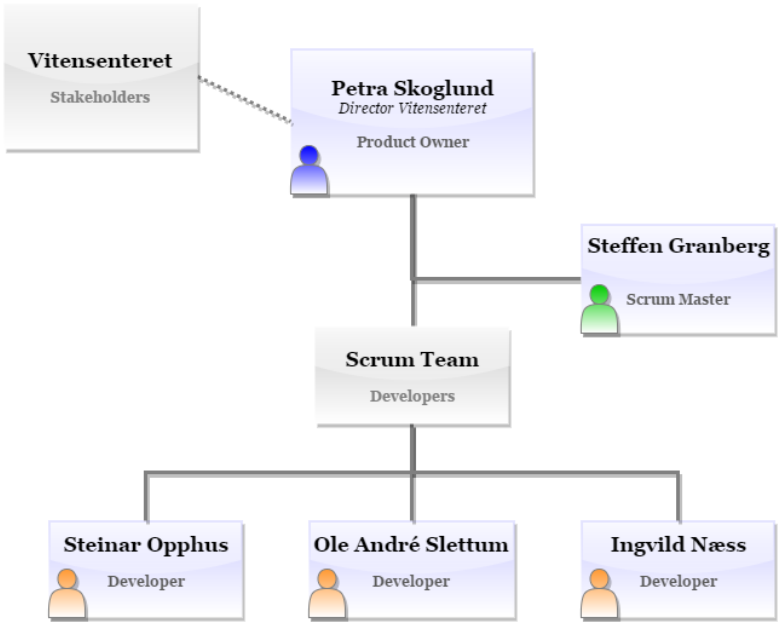


Figure 1: Organizational chart

2.2 Role Descriptions

The scrum-team is organized after normal scrum roles: The product owner, the scrum master and the development team.

Petra Skoglund, the director at Vitensenteret Innlandet, is chosen to be the product owner for this project. She has been the director there for the last nine years, and therefore has good knowledge of the centre. Her main role is to be the link between us and the stakeholders at Vitensenteret and to represent their vision during the project.

Steffen Granberg has been scrum master at a previous project, and was the most natural choice for us this time. He has qualities that make him well suited for this role. His role is to be the link between the development team and product owner. He will also make sure that the development team is working efficiently and gets all the tasks done during the sprints. Steffen will have a split role in this project. While he is scrum master, he will also be part of the development team.

The development team consists of Ole André Slettum, Steinar Opphus, Steffen Granberg and Ingvild Næss. The developers are responsible for delivering finished tasks when each sprint is over. The tasks are distributed among the developers at the beginning of a sprint. All members of the team help one another to ensure a successful sprint completion.

3 System Development Model

3.1 Project Characteristics

Our team consists of four developers. Our customer, Vitensenteret, have few rigid requirements for the final product. The core concept is a mobile app solution that will enhance user experience for people visiting the centre.

3.2 Discussion

The fact that feature specification isn't final makes this project ideal for use with an agile software development model. Our development team will be collaborating with a steering group of employees working at Vitensenteret. Constructing requirements and solutions by collaboration of cross-functional teams is one of the key principles in the use of agile development models.[1]

We will be using SCRUM as the foundational development model for our project. Sprint duration will be one week, Tuesday to Monday. Before each sprint we will have a sprint planning meeting where we'll discuss the next sprint and create the sprint backlog. When a sprint is finished we'll have a sprint review meeting. Daily scrum meetings will not be used very rigidly. This is due to the fact that this is one of several projects we're working on and it may not be given that all group members work with this project in the same day.

To ensure sufficient documentation we will seek inspiration from the artifacts used in the Rational Unified Process-model.[2] We will i.a write use cases to map and define functionality, system sequence diagrams to elaborate use cases, perform risk analysis and more.

4 Risk Analysis

4.1 Identify and analyse risks

We have chosen to set up a table where we look at each risk, looking at the probability and consequence of it. We have chosen to categorise the likelihood that a risk can occur with unlikely, likely and very likely. The consequences of that risk occurring is categorised unproblematic, problematic and critical. The following risks have been identified and analysed:

Number	Risks	Probability	Consequence	Actions
1	We can't deliver the product to deadline. This can be caused by many different reasons.	Likely	Critical	Yes
2	Major changes in the requirement from the customer during the project.	Likely	Unproblematic	Yes
3	The application is not user friendly enough for all users, which leads to people not using it.	Unlikely	Critical	No
4	One or more developers become ill over a longer period, or quits during the project.	Unlikely	Critical	No
5	Loss of source code.	Unlikely	Critical	No
6	Other companies come with the same idea and create a similar solution	Very likely	Unproblematic	No
7	Vitensenteret can stop the project.	Unlikely	Critical	No
8	Trouble linking the application with various experiments in the centre.	Very likely	Problematic	Yes
9	We bump into technological problems. Certain functionalities will not be implemented because it is too difficult technological or not possible to achieve.	Likely	Problematic	No
10	The experiments do not work and therefore the application can't be used there.	Very likely	Problematic	Yes

4.2 Plan for managing the main risks

We've selected four risks where we'll elaborate what counteractions can be made to reduce their potential of negative impact. These are counteractions that either reduces likelihood of the risk occurring, makes the consequences less if the risk occurs, or both.

Risk number	Action
1	If we get delayed at the end of the project, we can inform the product owner. We can together agree on what should be taken out of the delivery. To reduce the likelihood of it happening we can make sure not to be delayed along the way. We can make sure that all the tasks in the sprints are implemented. If we see that we are behind in the sprint, we can spend more time on a task or solve things in a different way that takes less time but not affect the final product to much.
2	It is important to have a good process plan early in the project so that Vitensenteret can express their requirements and desired results. We can discuss their ideas and tell them what is possible to do on a technological level. Throughout the project, it is important that we meet the customer often so we can show and demonstrate the application. Meeting often makes it easier for Vitensenteret to express their ideas, and for us to make continual changes.
8	We can early on in the process try to identify how the various experiments work. After doing this we can estimate the time we need to develop interactivity between the experiment and the app. If there is an external supplier of the experiment, we can contact them early and ask if we or them can change the solution so that we can use it.
10	If an experiment stops working, the app can pop up a notification with information of failure when the user tries to use the experiment.The application can notify the staff at the science centre through the administration web interface that a particular experiment does not work.

5 Quality Assurance

5.1 Documentation and use of Tools

Our experience from earlier projects, both related to work and school, combined with everything we've been taught in the different software development courses, has given us a clear picture of what defines a good development process. We've tried to make use of this knowledge as much as possible in this project.

Before Development

The first step in this project was a meeting between the development team, the product owner and stakeholders. The goal of this meeting was to define the basic idea of the application and what Vitensenteret would like to achieve with it. A brainstorming session gave lots of ideas which in turn was used as the basis for use cases and product backlog.

The first meeting between the developers was used to define roles, development model, which tools to use and fill the product backlog with tasks. We've already become quite proficient with Bitbucket and IntelliJ (and Android Studio, which is based on IntelliJ), and decided to continue with those tools. It was a natural decision to expand into other Atlassian products, to make use of the easy flow of information between tools from the same provider, so we decided to use Jira as a planning tool and scrum board, and Confluence for meeting notes and other information closely related to the project.

We considered using Confluence for this project report as well, but decided that LaTeX was a better choice to avoid being constrained by Confluence's Wiki-like style.

Jira is well suited for projects using Scrum as a development model, so it was decided early to use Scrum, at least as the base for our model.

During Development

To assure quality throughout the project, Jira must be used continuously, and good practises for programming followed, like use of frequent, informative commits and good commenting. We have adapted the project and coding guidelines of Ribot[3], which in turn is based on the regular Android coding and other guidelines, as well as regular Java guidelines.

- Every task must be defined and put in backlog (see figure 3).
- Story points will be used to estimate time used to complete each task
- Weekly sprints will be used, Tuesdays-to-Mondays, and tasks from the backlog will be assigned at the beginning of each sprint.
- Jira will be used as a digital SCRUM board (see figure 2. Each developer will move a task from "To Do" to "In Progress" when they start working on it (and back again, if they switch tasks for some reason). When a task is finished and merged back with master (for coding tasks), it must be moved to "Done".
- Burndown charts (figure 4), control charts and cumulative flow diagrams will be used to analyse progress, find bottle necks and check if the workload is reasonable. By looking at the burndown chart (figure 4) it is easy to see that we issued too many tasks on that particular sprint, since so many were unfinished by the end of it.

- Jira and Bitbucket is integrated, to make smart commits possible (to start and complete tasks directly from the IDE).
- When a developer starts a new coding task, a branch must be created (can be done directly from Jira), and all work done will be committed to this branch until the task is done. The branch will then be merged with the master branch.
- It's also very important to remember that when a branch is to be merged with master, **the master branch must be merged with the external branch first!**. This is to give the developer a chance to resolve merge conflicts and test out the new code before committing it to the main branch for all to use. If the merge results in critical bugs that's hard to fix, the branch can be rolled back to a working copy with no harm to master.
- Each commit message must contain the task ID from Jira, even if smart commits is not used (this will enable commit history on each task from within Jira).
- All commit messages must contain a informative description to make it easily understandable for everyone.
- All coding must be committed often, especially when a specific function is completed and working correctly.
- Every class and method must be commented following the JavaDoc standard. In-line comments can be used to clarify code within methods.
- Class and variable names must be informative and follow the standard of that specific coding language.

Furthermore, the scrum master will have the oversight and last say in delegating tasks to the other developers. He will also be the main link between the developers and product owner.

Every single meeting will have meeting notes with as detailed information as possible, to keep track of ideas and tasks generated in that setting.

The main structure of the code will be jointly developed, and single coding tasks will be completed by each developer on their own.

We will have a basic code review at the end of each sprint, so everyone will be familiar with everybody else's code and get a better picture of the structure as a whole.

Documentation

- All methods and classes in the code, will have comments following the JavaDoc standard, so documentation for later development and maintenance will be available in separate HTML-files for easy access in a structured and understandable format.
- This document will also be part of the documentation for the project, since it has valuable information on the overall structure and thought process.
- The web application used for administration of the content of the app, will have a separate user guide for the administrators at Vitensenteret.
- The users of the app itself will get basic instructions on posters on site, as well as instructional videos and in-app hints connected to specific experiments and stations.

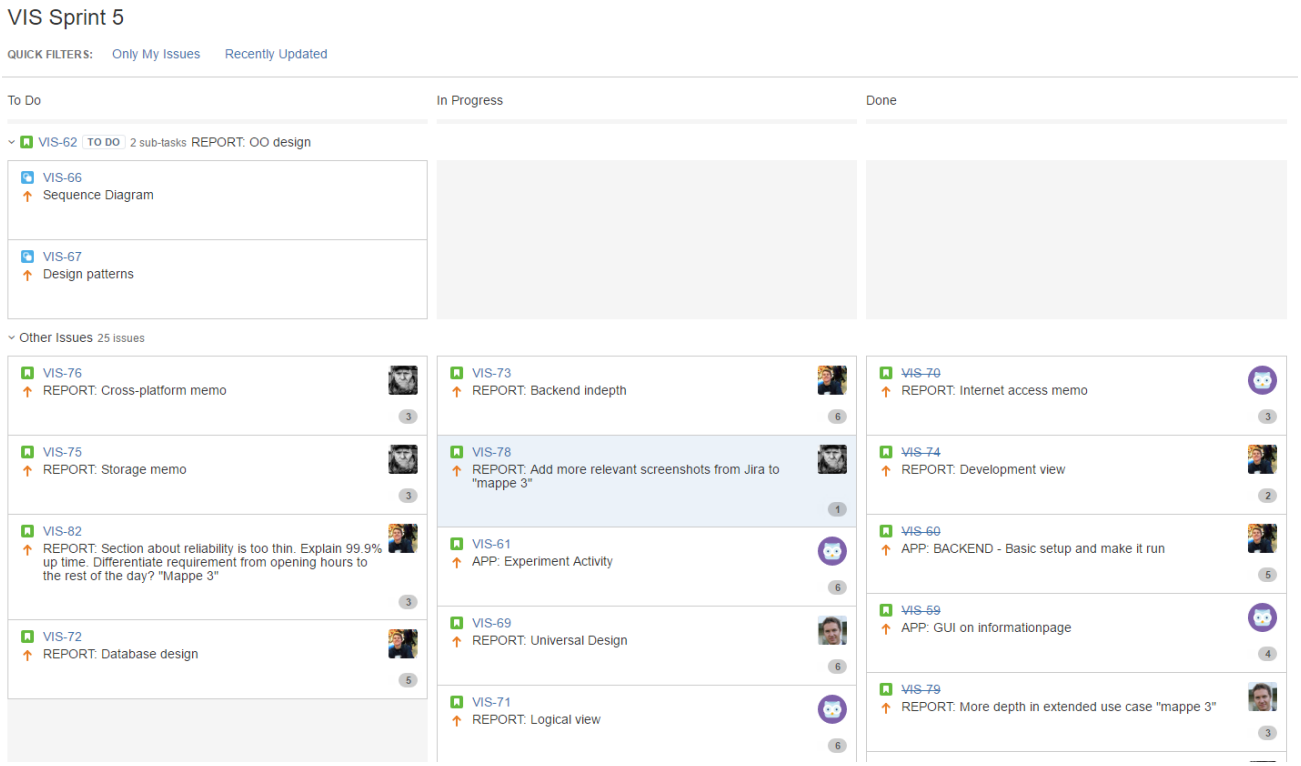


Figure 2: The SCRUM board from sprint 5

Tools

Name	Type	Used for
ShareLaTeX	LaTeX editor and compiler for online collaboration	Project report
Jira	Proprietary issue tracking product, developed by Atlassian	Project management
Bitbucket	Hosting service for projects with Git revision control system	Revision control
Android Studio	The official IDE for Android platform development	Android app
draw.io	Network diagram software	Flowcharts, UML and other diagrams
Xcode	IDE for developing software for iOS	iPhone app

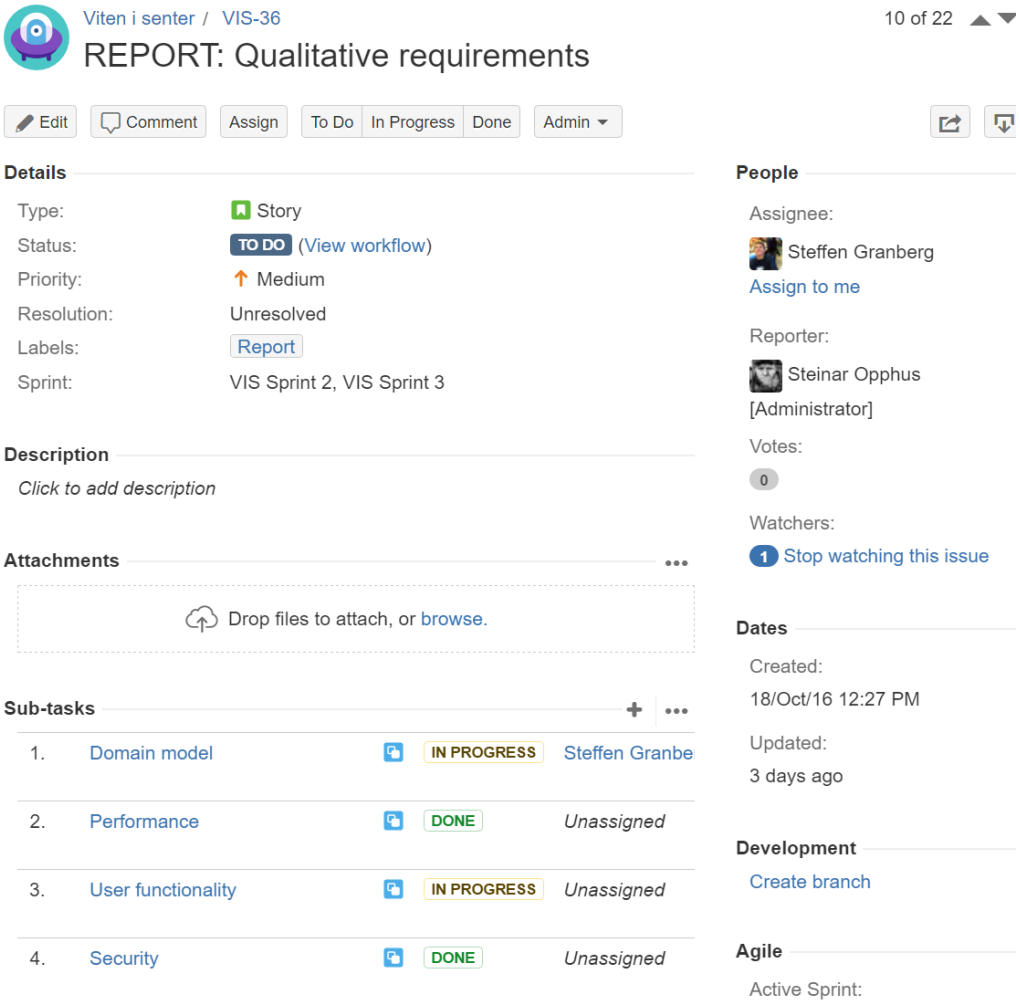


Figure 3: Example of a Jira issue/story, with assignees, sub tasks and progress

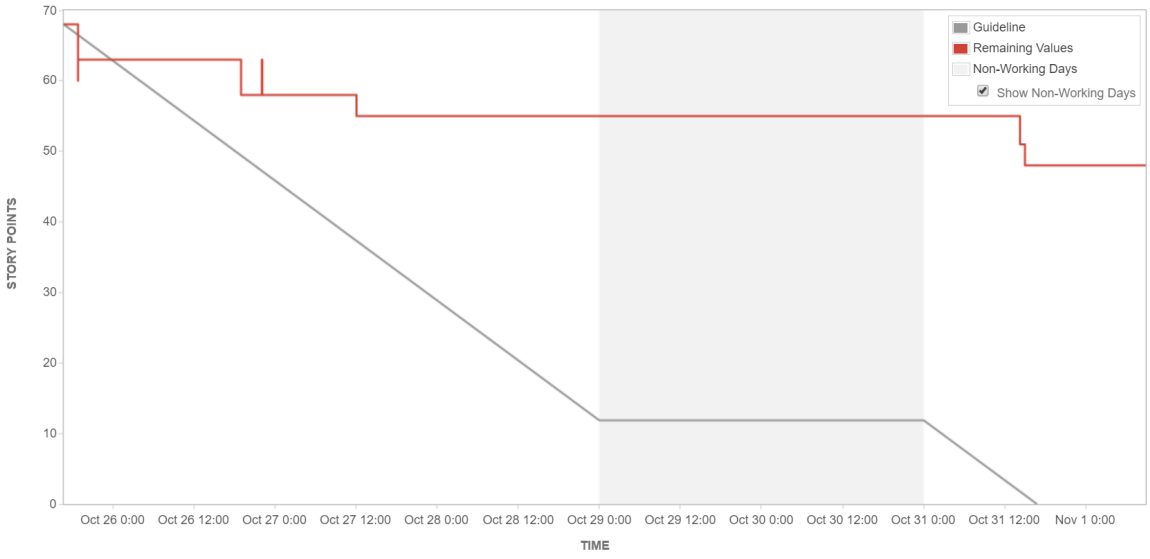


Figure 4: Burndown chart taken from Jira for sprint 2

6 User Groups

6.1 PACT analysis

We have chosen to include the PACT analysis in our project so we can pay attention to the ergonomics of the application. A ergonomic application is important to give the user a good user experience. The PACT analysis looks at the application in four categories: people, activities, contexts and technology. By doing this analysis we get a understanding of the most important things in the four categories, and can find a solution to combine the areas. We as developers of the application must understand user base, we must understand the activities those users will do and in what contexts the activities can take place. We also need to understand what technologies that can be used and how these technologies can be used to support the activities that will take place in the various contexts.

People

People of all ages are the potential users of the application. We must also take into consideration that the application will be used by adults with multiple children that don't have there own mobile phone, so the application must handle multiple users at the same time. The users can be those originally from the area and those who are visiting Gjøvik. The solution should be available and easy to use for people with different background. Consideration should be given to people with orientation difficulties. Orientation difficulty involves low vision, blindness or hearing loss. We must also take info consideration that not everyone understands Norwegian, so the application will also support English.

Activity

The purpose of the application is to enhance the experience at Vitensenteret. Users can get information about stations and the center, use the application as part of an experiment and participate in daily quizes. Users can create their own profile with a nickname, saving previously done experiments and gain points. This can i.a be used to create leaderboards for the user to see. Users can also access some experiments at home so that they can read more about them. Some of the games at the centre will be digitalized so the user can play them at home

Context

The main use of the application will be inside the center, but there are different environments inside the center that we need to take into consideration. Inside the center there can be areas with lots of light, little light and with lots of noise, the application must support all these environments. Users can be on the go when they use parts of the application, like the information page to get openings hours, prices and road description. This can give motoric problems, large surfaces on the app can minimize this problem.

Technology

The solution is a mobile application. First time use requires internet access to download content. For the application to work properly and with the latest updates and daily quiz it is recommended that the users update the application when arriving at Vitensenteret. Vitensenteret provides free wireless network to all guests. During use some of the content in the application don't need internet access to work, an example is the math game - when downloaded it's available offline. Since this application will be used by kids and young adults we need to think about how much amount of data that will be used when the device is

connected to mobile networks. An example of such considerations are not automatically starting a video or a sound clip. This are contraindicated due to high use of mobile data.

6.2 Personas

Creation of personas is a method used to identify the audience of a product, in our case a mobile application.[4] A persona is used to represent the user of a product. This can further be used to create use cases and analyze what each persona need to fully use the product. Old persons may need more color contrast or font size, young users may need more guidance etc. Creating diverse personas makes it easier for the designers and developers to see the product from a user perspective, ultimately resulting in a better product that satisfies users. We have created three personas with different background and age to help us identify measures for better user experience.

Personas 1: Julian Bergersen

Age: 10
Work: Student
Living: Skreia

Julian is an active 10 year old boy from Skreia, he lives home with his parents. He enjoys playing soccer and the piano - when he is not working on his homework. His favourite subject at school is mathematics and science. He reads a lot about science stuff, and he and his father often do experiments at home. He has just gotten a brand new smart phone and enjoys playing educational and challenging games. He also enjoys visiting Vitensenteret at Gjøvik, and has bought a pass for rest of the year. He especially enjoys the math area with all the challenging math games, he even have a couple of them at home. If he wants to check out what is happening at Vitensenteret he checks the Facebook page with his father, they often attend events.

Personas 2: Kristin Nilsen

Age: 35
Work: Project leader
Living: Hamar

Kristin is a positive and active woman. She is married to her husband Anders and together they have three kids Lillian (14), Marit(10) and Torgeir (7). She lives like any other mom a very busy life with lots of daily logistics to attend. All her kids is active in sports and they also try to do other family activities like going to Vitensenteret Innlandet in Gjøvik. Since her husband is a engineer he want the kids to explore the wonders of science and they visit Vitensenteret several times a year. The kids like to compete and they compete with everything from finding the most Pokemon's in Pokemon Go, to who can run the fastest. Kristin is not very up to date on technology but uses her iPhone 6 - that she got from work - daily, and Torgeir gets to borrow it from time to time since he is the only one in the family without a phone.

Personas 3: Dina Mekic

Age: 65
Work: Retired
Living: Bosnia
Dina is a conservative and curios woman. She is from Bosnia and are in Norway on a family vacation with her daughter, her son-in-law and her two grandchildren. She has an old phone, and is quite reluctant to try new technology. Her grandchildren have their own smart phones, but she don't see the use of them. She has been travelling a lot in her life, and her English is very good. On their vacation she does a lot of exciting activities with the kids, while the parents are free to do what they want to alone.

7 Use Case

7.1 Diagram

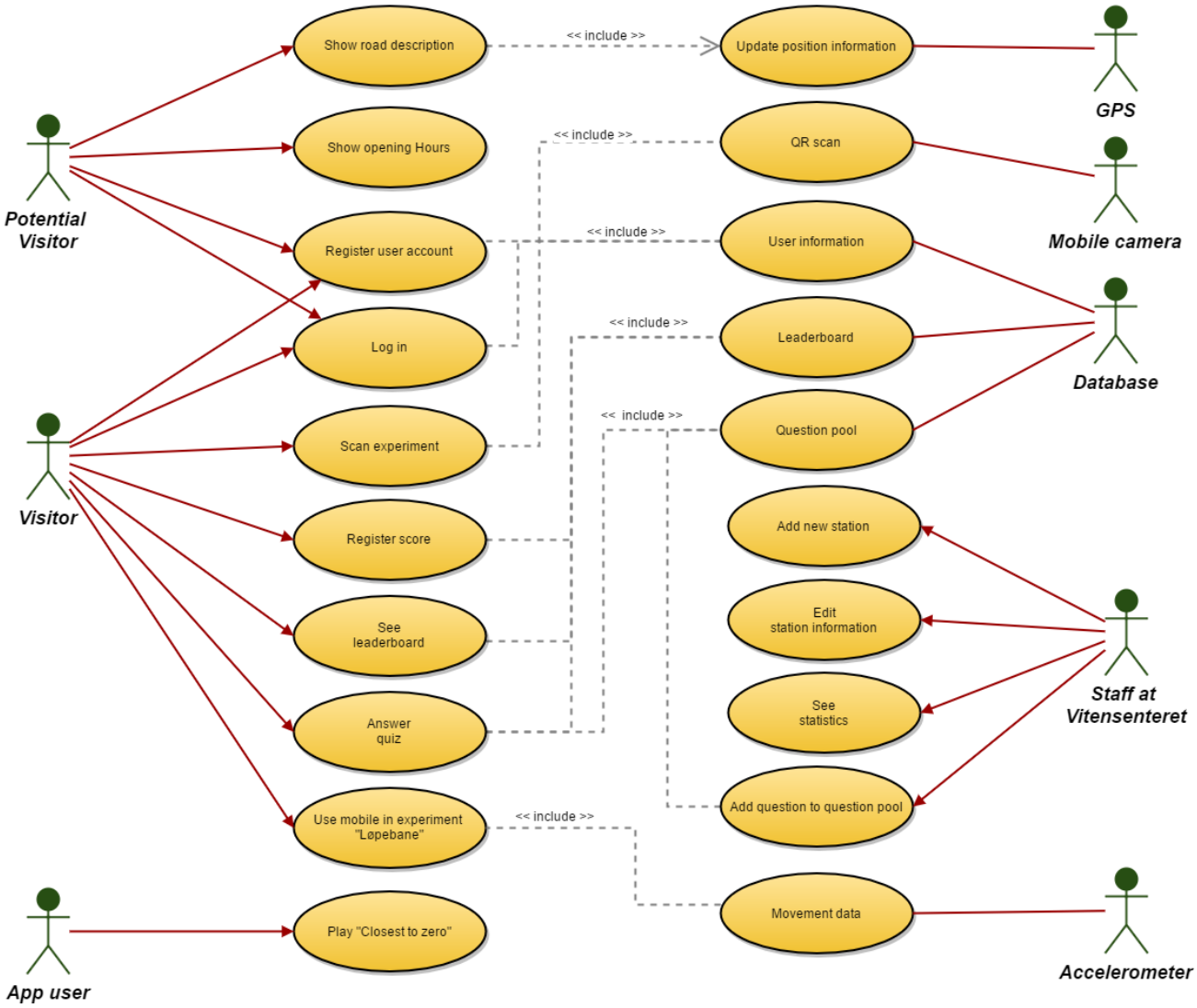


Figure 5: Use case diagram

7.2 Uses Cases (high level)

Use Case Name: Show road description
Actor(s): Potential visitor
Goal: See road description to Vitensenteret
Description: Inside the app Potential visitor gets a brief road description about Vitensenteret's location. He can click on a link which will send him to a map/GPS application of his choice, ultimately showing him detailed road descriptions.

Use Case Name: Show opening hours
Actor(s): Potential visitor
Goal: See Vitensenteret's opening hours
Description: Potential visitor uses application and visits the info page to see opening hours

Use Case Name: Scan experiment
Actor(s): Visitor
Goal: Scan and recognize experiment, get additional info about experiment
Description: Visitor uses his phone to scan experiments that are recognizable through QR-codes, a new activity inside the application shows him information about the station

Use Case Name: Register score
Actor(s): Visitor
Goal: Register score on an experiment
Description: User can register his score on certain experiments. This will be used on leaderboards and for himself to see

Use Case Name: See leaderboard
Actor(s): Visitor
Goal: See leader board on specific experiments
Description: User opens an experiment and can see leaderboards for competitions related to the experiment

Use Case Name: Answer quiz
Actor(s): Visitor
Goal: Start and complete a quiz
Description: Visitor starts quiz and answers question. If questions was answered within the given time frame, score and statistics are shown. If not, user is asked if he wants to try again. Depending on his answer or quiz restarts or user is returned back to Main activity

Use Case Name: Use mobile in experiment "Løpebane"
Actor(s): Visitor
Goal: Calculate acceleration when using experiment Løpebane
Description:Løpebane is an experiment where you can run 10 meters as fast as you can. User should be able to start a new run that will measure his acceleration when running

Use Case Name: Register user account
Actor(s): Potential visitor, visitor

Goal: Register new user account
Description: User can register an account with nickname, e-mail, password and age

Use Case Name: Log in
Actor(s): Potential visitor, visitor
Goal: Log in to user account
Description: User logs in to the application

Use Case Name: Play “closest to zero”
Actor(s): App user
Goal: Access and play game
Description: User can access and play a simple game on his phone - a game based on an experiment at Vitensenteret called “closest to zero”. Closest to zero is an experiment where one changes the position of numbers between 0 and 9, these are multiplied and added together in a specific way. The challenge is to find the combination that makes the result of the calculation zero.

Use Case Name: See statistics
Actor(s): Staff at Vitensenteret
Goal: See statistics related to the application
Description:Staff can see statistics about total number of users, total visitors per month(year), number of times an experiment is scanned and in what frequency people are revisiting the centre.

Use Case Name: Add new station
Actor(s): Staff at Vitensenteret
Goal: Add new station to the pool of station
Description: Add new stations that can be scanned and recognized with QR-codes, can add information like name, complementary information, videos, pictures and questions

Use Case Name: Edit station information
Actor(s): Staff at Vitensenteret
Goal: Edit information about existing station
Description: Staff can edit information about existing stations

Use Case Name: Add question to question pool
Actor(s): Staff at Vitensenteret
Goal: Add questions to question pool
Description: Staff can add questions to the random category

7.3 Use Cases (extended)

Use case name: Answer quiz
Scope: Vitensenteret application
Primary Actor: Visitor
Stakeholders and interests:

- Visitor: Want to answer daily quiz, save score and see leaderboards
- Vitensenteret: Want leaderboards updated and usage statistics saved
- Developer: Want performance statistics like information about application crashes

Purpose: Start and complete quiz
Description: Visitor starts quiz and answers questions. If questions was answered within the given time frame, score are saved in database and shown to user together with statistics, e.g daily leaderboard. If time ran out and visitor is still inside center, he's asked to try again. If time ran out and he's not inside the center anymore, he's returned back to MainActivity.
Type: Essential
Pre-condition: Logged in on the app and located inside the centre
Post-condition: User has answered the daily quiz and delivered his answers OR hasn't answered all questions within the given time frame
Main Success Scenario(Basic Flow):

1. User clicks "Start quiz"-button
2. Quiz activity starts and fetches X random questions from database
3. Question is given to user with four alternatives
4. User answers question
5. User is shown popup with either "right answer"/"wrong answer"
6. Visitor repeats step 3-5 until all questions are answered
7. Visitor score is sent to database
8. User is shown his score
9. Daily leaderboard is retrieved from database and shown to user
10. User can click buttons to get monthly or yearly leader boards
    (a) User clicks "See monthly leaderboards"
    (b) User is shown monthly leaderboard
        i. If user is in top ten, his placement is highlighted
        ii. If user is not in top ten, his placement is shown in addition to leaderboard

    OR

(c) User clicks ”See yearly leaderboards”

(d) User is shown yearly leaderboard

i. If user is in top ten, his placement is highlighted

ii. If user is not in top ten, his placement is shown in addition to leaderboard

OR

(e) User clicks back

(f) User is returned to MainActivity

Extensions:
User didn’t answer all questions within the given time and is still located inside the center

1. User gets the option to give the quiz another try or return to home screen of application

   (a) User clicks try again

   (b) QuizActivity is restarted

   (c) ”Main Success Scenario” is the next chain of events
   OR

   (d) User clicks ”Return home”

   (e) User is sent back to home screen of the application

User didn’t answer all questions within the given time frame and he’s not located inside center anymore

1. User is shown message in the likes of ”Oops. You went over the time limit for the quiz, come back to Vitensenteret to try quiz or other activities again”

2. User is sent back to home screen of the application

System failure, application suddenly stops. All answers to questions and state is saved for each user

1. User opens application after failure

2. User is asked if he wants to finish his previously started quiz

   (a) User answers that he want to finish previously started quiz

   (b) User is sent back to question number X + 1 where X is his last answered question
   OR

   (c) User answers that he don’t want to finish previously started quiz

   (d) User is sent back to MainActivity

7.4 System Sequence Diagram

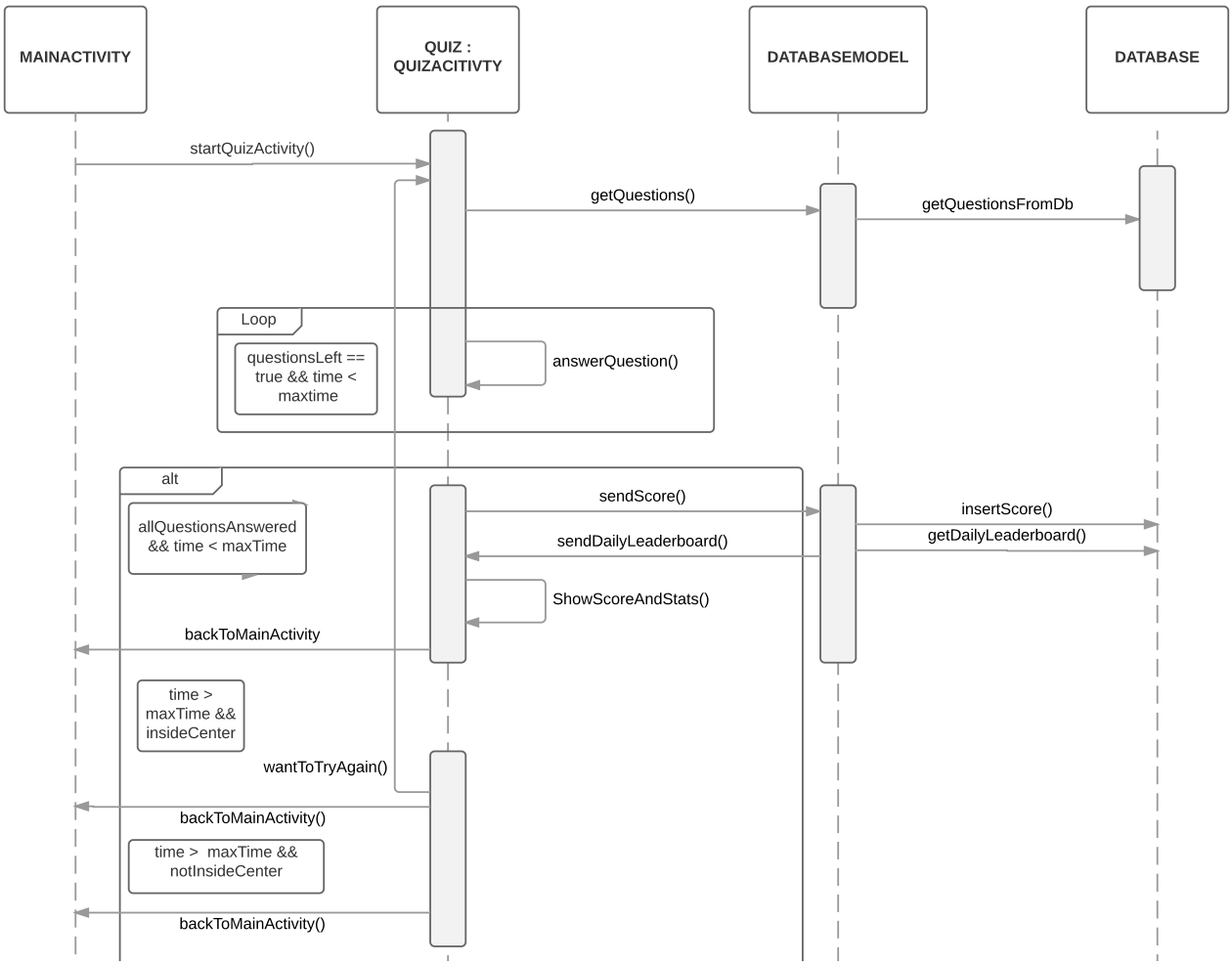
The following diagram shows activity flow and function calls in use case ”Answer quiz”. All alternative situations are not illustrated, but the diagram seeks to show the general flow in the use case.

QuizActivity is started from MainActivity when user clicks ”Answer quiz”, QuizActivity asks DatabaseModel for random questions. DatabaseHelper gets questions from the Database and returns them to the QuizActivity

User answers questions while there are questions left to answer, he must also answer within a given time frame. If all questions was answered in time, user score is sent to DatabaseModel which inserts score into database. DatabaseModel gets daily leaderboard after sending score. Leaderboard is sent back to QuizActivity which shows user the daily leaderboard. User can click buttons that will show him monthly and yearly leaderboards og send him back to MainActivity.

If all questions wasn’t answered in time and he’s still in the center, user gets the option to retake the quiz or return to MainActivity.

If user didn’t answer all questions within time frame and he’s not inside the center, user is shown a message and returned to MainActivity



8 Qualitative Requirements

8.1 Domain Model

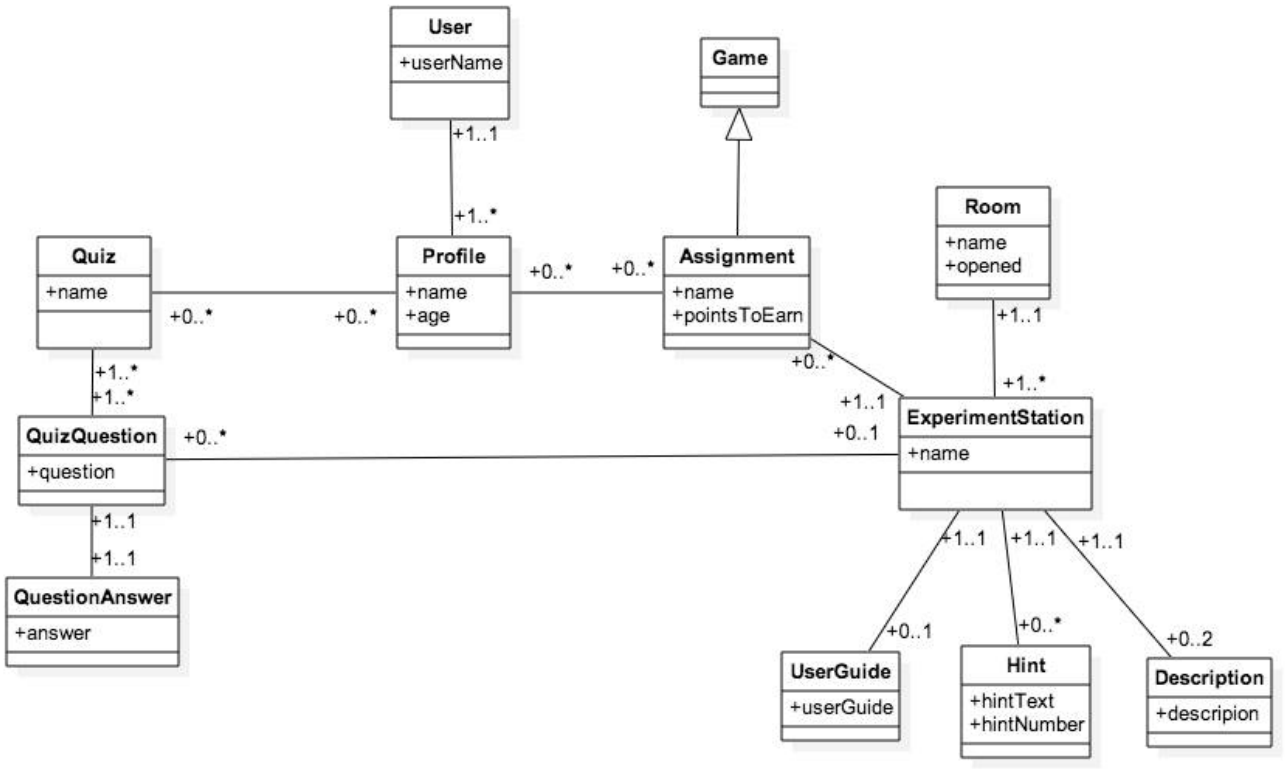


Figure 6: Domain Model

8.2 Technical possibilities

This app have a lot of functionality and we have looked into some possible technologies that could be used to implement some of these functions.

- To scan an experiment the mobile could present the user with a camera, allowing the user to scan a small QR code that is mounted on the experiment.
- The app could be developed using a cross-platform solution such as React Native[5], Cordova[6] or pure HTML5[7] - this depends on the amount of sensors requires and the possibilities of access to these.
- To recognise that a user is inside Vitensenteret we could place beacons around the centre[8]. With these we could use Beacons technology to locate the user.
- Use push technology to send informative messages and updates to the user.
- Use Google and Apples gamification systems[9]. That gives us access to achievement systems and other functionality.

- Sensors include accelerometer to measure acceleration on the running course as Vitensenteret. But we would also like to make use of gyroscope, gravity and temperature.

8.3 Ecosystems

It is important that the application work under different ecosystems That's why we have made a list of the different ecosystems and environments the application should work under:

- Android phones with Android 4.0 (Ice Cream Sandwich) or newer
- Apple phones with iOS 8 or newer.

The web interface should work in all newer browsers such as:

- Chrome
- Safari
- Firefox
- Opera

The web interface should also work in browsers on mobile phones such as:

- Chrome Mobile
- Safari Mobile
- Firefox for Mobile
- Opera Mobile

8.4 User Functionality

The app should be as intuitive and easy to use as possible. Vitensenteret has all kinds of visitors on a typical weekend. This include kids, young adults, parents, grandparents and foreign/Norwegian. It is important that we do not exclude any of these groups. Therefore the GUI should be self explanatory. You should get quick access to the most used functions as: Scan a new experiment and start a new Quiz.

The process of register a score on a station should be a fast process. We believe that the easier and faster this is, the more people will bother to do it. It is also important that the app is quick at starting up, this is important because a user will often open the app and want to scan a experiment as quickly as possible. More on this later.

Since Vitensenteret don't want all of its visitors going around with there head buried in their phones it the UI should give the opportunity to save descriptions and background information about experiments for later, so that the user can enjoy this at a later time.

8.5 Security

Security is important for us when developing this app. This because we know this app will be used by children and young adults. Privacy is in focus and we try to save as little information about the user as possible. And the information we do save, we try to make as anonymous as possible.

In Europe there are strict restrictions to make sure app developer follow the guidelines for privacy. To make sure we do this we have decided to follow *Datatilsynet*'s guide to integrate privacy into the application. Some of there points includes:

- Evaluate the consequences of poor privacy.
- Make privacy a default setting, and not something the user has to manually enable.
- Build privacy as a part of the system.
- Maintain the privacy from beginning to end - with use of encryption, rules for storing of data and safety of database server.
- Show openness of what kind of data and information you gather. And also what precautions you have taken.

To read the complete list see *Datatilsynet*'s website[10]. Some of the things we plan to do includes:

- The user can register an account by not giving up much personal information about them self.
- We do not save which user has used or scanned which experiment, only that a user has done this at this time. This is used by the Vitensenteret to determine if an experiment is used or not.
- The name in your profile can be changed at a later point, and you are not obligated to give your real name, because this user name will come up on leader boards.

Other security measures should include:

- Make sure only authorised persons at the Vitensenteret has access to statistics about usage and other information.
- Make sure you have to go through the server to update score and quiz information. So that the users can't cheat.

8.6 Reliability

It is important for Vitensenteret that the application is very reliable. This due to the fact that it is going to be used when only w1eekend personal is working in the centre. This means that there will be no technical support for the people using the app.

On average the center has 30-60 visitors on the weekends and the possibility of this being as high as 400-500 on some specials days. The server must handle this load, but according to the director at Vitensenteret it is acceptable to have some downtime during the weekdays, but as close to 100% up time on the weekends. This due to the fact that it is on the weekends the system main use will be. And if it goes down when the center is open, it should be some alert mechanisms in place to alert the system administrator so that it can be brought up as quickly as possible, from a remote location.

References

[1] Agile software development [wikipage]; 2016 [cited 23/10-16]. Available from: https://en.wikipedia.org/wiki/Agile_software_development.

[2] Rational Unified Process [wikipage]; 2016 [cited 26/10-2016]. Available from: https://no.wikipedia.org/wiki/Rational_Unified_Process.

[3] Project Guidlelines [github]; 2016 [cited 04/11-2016]. Available from: https://github.com/ribot/android-guidelines/blob/master/project_and_code_guidelines.md.

[4] Personas: bruk og praksis;. (Accessed on 11/23/2016). http://www.karde.no/personas.no/.

[5] React Native — A framework for building native apps using React [webpage]; 2016 [cited 4/11-16]. Available from: https://facebook.github.io/react-native/.

[6] Apache Cordova [webpage]; 2016 [cited 4/11-16]. Available from: https://cordova.apache.org/.

[7] HTML5 [webpage]; 2016 [cited 4/11-16]. Available from: https://www.w3.org/TR/html5/.

[8] Types of beacons - Wikipedia [webpage]; 2016 [cited 4/11-16]. Available from: https://en.wikipedia.org/wiki/Types_of_beacons.

[9] Achievements — Play Games Services — Google Developers [webpage]; 2016 [cited 4/11-16]. Available from: https://developers.google.com/games/services/common/concepts/achievements.

[10] Sjekkliste for innebygd personvern - Datatilsynet [webpage]; 2016 [cited 4/11-16]. Available from: https://www.datatilsynet.no/Teknologi/Innebygd-personvern/Sjekkliste-for-innebygd-personvern/.

B.2 Mappe 4

IMT3102 - Objektorientert systemutvikling

Prosjektrapport mappe 4

Steinar Opphus	140174
Ole Andre Slettum	141329
Ingvild Næss	248258
Steffen Granberg	141647

I samarbeid med:

Høsten 2016

Contents

1 Technical Memos

1.1 Native or Hybrid?

Factor	Should the application be developed as separate, native apps for each platform, or as a hybrid, cross platform?
Assessment	• We are already developing an android application natively for the prototype to be used in Mobile Development Project (IMT3672). • The finished application will need to use many of the phone sensors for some of the interactive experiments. Accessing sensors is easier for native apps compared to cross platform[1]. • Native apps are faster, but the difference is probably quite small in apps such as ours. • It is easier to control how often the app will connect to a remote resource, preventing the younger users from overusing the Internet bandwidth. • We, the developers of this application, has more experience with native development, diminishing the learning curve. • The only con of native development, as far as we see it, is the need for separate code bases, which in turn increases the amount of development and maintenance.
Solutions	• Based on the above assessment, we choose to build our app natively for android and for iOS. • Developing native apps for Windows phones or other OS's is not worth the effort, based on global market shares[2]. • The back end will be shared by both apps. Only the front end needs to be natively developed.

1.2 Minimize access of external storage

Factor	The application should access external storage as little as possible. The younger users might have limited access to Internet bandwidth. Since they are a big portion of the potential users, remote access should be limited.
Assessment	• Accessing remote databases dynamically might slow down the application in use, as well as drain battery faster[3].
Solutions	• To avoid downloading data when there are no updates, a version id should be checked before an eventual update. Every time new information is stored on the database, the version id is changed.

1.3 Remote database access

Factor	The application needs to access a remote database to get updated information about experiment stations and new quiz questions.
Assessment	• Direct connection from the application to a remote database is insecure, since an experienced programmer can reverse-engineer the app to get the user name and password to access the database[4]. • Handling all the database functionality locally slows the application down and drains battery faster. • By adding another layer to the architecture and splitting the system, we get more control over the structure of the database and increases backward compatibility for users who's slow to update the app[3].
Solutions	To prevent unauthorized access to the database, we will use the RESTful web services[5] to query the database from the application. REST will then return the results as JSON[6].

1.4 Internet access

Factor	The kids and young adults that are using the application might have limited access to Internet. The application should use the Internet as little as possible when in use.
Solutions	• We have chosen to have a thick client, so that most is stored on the device. This is to reduce the data usage when the application is being used. When a user scans a experiment the application gets the information from within the device, instead of accessing the data remotely. • We are giving each entry in the database a version id. We can check against each of these ids if there is any update. And if an entry is updated we can download just the one entry. This saves data because the user don‘t need to update the hole application every time. • Vitensenteret provides free wireless Internet to all guests. So the user can connect to this and update the application.

1.5 What technology to use for identifying experiment stations

Factor	
	The application will use QR code scanning to identify different experiment stations
Assessment	• Vitensenteret want the application to be able to provide different information to the user based on their location inside the centre. Since the application will be used indoors, use of GPS is excluded from the different technology alternatives. In the end we had to choose between using QR codes, NFC or Beacons. – Device support - Every smart phone can potentially scan QR codes, NFC was estimated to be compatible with 50% of all smart phones in 2015[7]. An example of a smart phone that is used a lot but doesn't have NFC support is the iPhone 5. Beacons also is supported by fewer devices than the use of QR code scanning. – Budget - QR codes and NFC tags are cheaper to use than Beacons. One beacon costs from 100-200 NOK[8] whereas you get up to 50 QR code stickers/NFC tags for approximately the same price.[9] – Beacons offers a very interactive experience because the application triggers the interaction, this is in contrast to QR codes and NFC tags where users need to actively scan a code. We wanted the user to have more control over the interaction, intentionally seeking tags to scan. This favoured the use of either QR codes or NFC tags. – Devices - more devices support QR code scanning than both Bluetooth Beacons and NFC tags. We don't want to discriminate user that have old phones. – Both QR code scanning, Beacons and NFC recognition can be done without an Internet connection. – Range - many of the experiment stations at Vitensenteret are very close to each other. This can lead to problems when recognizing stations through the use of Beacons, signals can interfere and there is a chance that the user will get information about another station than the one he's really at.
Solutions	Based on the above assessment we have chosen to use QR code scanning for recognizing experiment stations. The QR code simply holds an id that identifies the station he's at, this id is used to fetch information from the database. One of the advantages of only providing id in the QR code is that we don't need to print new QR codes when adding new information. Instead one change the database entry that is associated with that particular id.

2 In depth: Backend solution

In this section we are going to take a closer look at the backend part of the system. This is a vital part of the entire solution and it is important to take into consideration everything from up time, response time and backup.

This section will contain an overview of different approaches of setting up the backend system, with a more in-depth explanation and analysis of the approach we recommend.

2.1 Traditional vs Cloud

Global data centres and cloud-based IP traffic has had a tremendous growth the last years, replacing the traditional way of renting or owning a local server park, and that trend will probably continue[10].

Solutions as IaaS (infrastructure as a service) are getting more and more popular. Here you rent a virtual machine park, and since you don't have a physical server, you don't need employees to maintain it[11].

Another solution is PaaS (Platform as a Service) where you not only get a virtual machine park, but also a developer framework you can use to develop your applications. The supplier will do all the system administration for you, as well as the drifting of the physical servers[11].

All of these options, both a local server, PaaS and IaaS, is suitable for our project and for Vitensenteret to begin with, but since we don't want to bind the developers to any particular framework, PaaS is not our first choice.

A local server will give the developers full control, but drain more resources by increasing the need for drifting and system administration, physical hardware and the need for a backup solution.

By renting some servers in the cloud, all of these drawbacks is negated. Amazon EC2[12] seems to be a good choice. This has the benefit of being very scalable if set up properly, and easier to clone if other science centres would like to buy or adapt our solution.

2.2 RESTful backend

A solution like this needs to be both flexible and easy to use, to make it as easy as possible for the staff at Vitensenteret to update and maintain information about the different experiment stations and other parts of the application. To make it easier for the developers to accomplish this, we've decided to build a RESTful service as a backend[13]. This will be a great foundation to make the web interface quick and easy, and making changes and additions will be less complicated.

A RESTful service is an API that lets the clients or nodes update all text-based information through stateless operations. The HTTP protocol is most often used with the HTTP operation verbs: GET, POST, PUT, DELETE and so on. With this you can make a call to an URL using these protocols and it will manipulate or return the information for you. One key aspect of using a RESTful service is that the client always should get a response, it should always return something. A common way to do this is to also use the HTTP protocols error messages. For example if you want to retrieve experiment station with the id 2, but there is no station with this id. The API would just return status code 404 not found.

The information is encoded in JSON, this is easy to read and parse. The backend is mapped up into URLs and you specify what you want in the URL-string. Some examples:

- /api/experimentstation $\rightarrow$ will retrieve all experimentstations

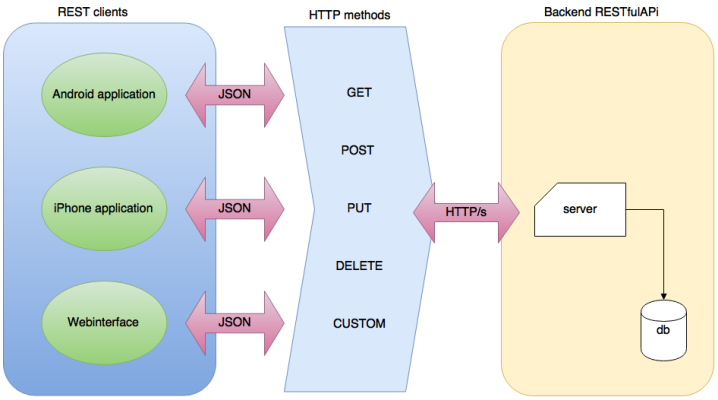


Figure 1: Communication with the restful service

- /api/experimentstation/4ecc184f-d8a7-4b0d-8046-14f5effefeb5 → Will give you information about the experiment with scan ID 4ecc184f-d8a7-4b0d-8046-14f5effefeb5.
- /api/quiz → gives you all quizzes

Another benefit from developing an separate API is that the development process can be divided. For example can one developer make the RESTful API, one the mobile applications and another the web interface for manipulation the API services. This way we decouple the system in a nice way. And the developing process can start at the same time, because if the developers agree on what the format of the JSON is going to be. The different parts can mock the API call until the backend is finished, and do not need to wait for the finished backend API.

2.3 RESTful developing

We have decided to develop our RESTful service in the brand new .NET Core 1.0 framework from Microsoft. This is a rewrite of the popular .NET framework Microsoft used for web before. .NET Core is open source, and can run on all platforms. What is especially nice is that it can deploy itself directly to a docker container, and be shipped to a VM. It is a modern framework when it comes to design and implementation.

The framework gives you a lot of free stuff from the beginning. It comes with its own middleware, that takes all incoming request and direct these to the right controller. Every time the client calls an endpoint (for example /api/quiz), the middleware will direct it to the right controller; in our case QuizController. In the QuizController .NET Core lets you specify functions as receivers of special sections of the endpoint, for example do we have a function that gets called when the user ask for a specific quiz, and another one when the user runs a HTTP POST command on the quiz endpoint. Lets see how a the GET function for a specific quiz looks.

```
1 // GET api/quiz/5
2 [HttpGet("{id}")]
3 public Quiz Get(int id)
4 {
5     return _context.Quizes
6             .Include(q => q.Questions)
7                 .ThenInclude(question => question.Answers)
8             .First(quiz => quiz.QuizID == id);
```

```
}
```

Listing 1: C# GET call example code.

Here the framework really does much work for us. The *[HttpGet("id")]* specifies that every time a HTTP GET request is received on the quiz endpoint, with a specific ID in the URL, this function will run, and everything this function returns will be sent to the user as JSON encoded objects. So a HTTP GET call to /api/quiz/1 will run this function, which looks in the context - the context is the database - finds the quiz that matches this ID, and returns it to the user.

Another interesting aspect; where do the context come from? This is where the software design pattern Dependency Injections gives us it's magic[14]. Every controller is dependent on the context, and instead of retrieving the context every time a call is made, we do this through dependency injection. We register our context as a service in the start-up file like this:

```
services.AddDbContext<BackendContext>(options =>
        {
            options.UseMySQL(Configuration.GetConnectionString("dConnection"));
        });
```

Listing 2: C# register context.

This makes sure that every time a controller is initiated, the context is injected in. So that you always have the context at hand, and can call the database directly. We can see how this looks in the UML in Figure 2.

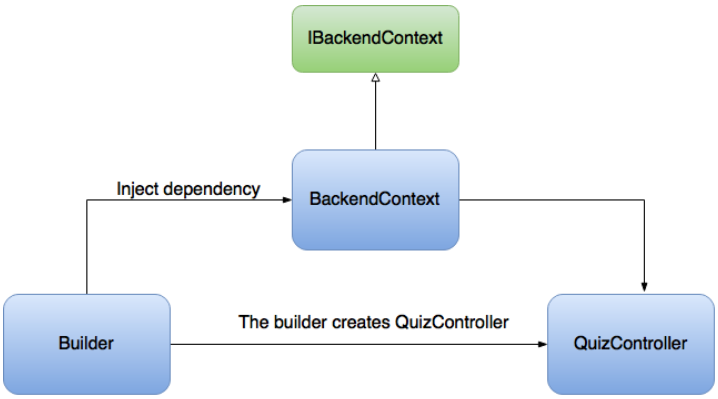


Figure 2: Communication with the restful service

The reason why this is important is that it makes a loosely coupled system between the classes and also make the code more readable, because of the reduction in lines of code required when the context is already there. It also makes testing easier, because when we want to test the controllers, we don't want to include the call to the database. Since we use dependency injection we can easily mock the context by providing the class with our own mocked context that return predefined data to the controller constructor. Making the system very maintainable.

2.4 Database design

Another key aspect of our solution is the importance of supporting several languages. This was something we had to deal with regarding the database design. Initially we had an database for the experiment station part of the design looking like figure 3.

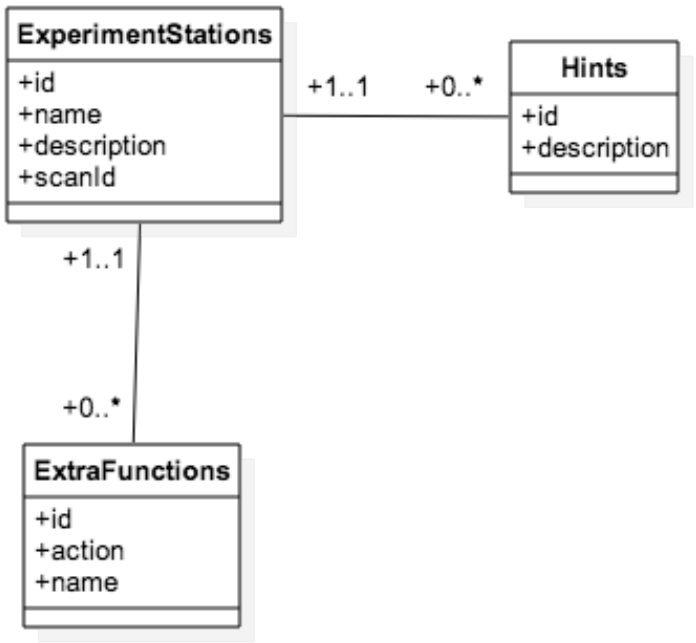


Figure 3: Communication with the restful service

But then when we was about to add another language we saw that it was no good way of doing this with our design. After some research we found several ways that a database could support multiple languages on.

- Column Approach. Add an extra column for each value, for example name_nb and name_en.
- Multi row Approach. Instead of duplication the column, we duplicate the entire row, and add a language code.
- Make an separate table with the name and language code. JOIN the tables on the id and language-code you want.

All of these three ways will give you a database that support several languages, but some of them have its drawbacks. The column approach is very hard to maintain, it is hard to add new languages because you need to change the schema of the table. And if the translation is not required you end up storing lots of empty fields if just some of the stations is translated. The multi row approach takes care of this, because you only have duplicates for the rows you have translated, and there is no JOINS required when querying for data. But it its hard to maintain and add new translations because you need to duplicate all other fields. And if one of the fields change, you need to update all other rows that is equal but only change the columns that requires translation. This again will produce duplication of fields.

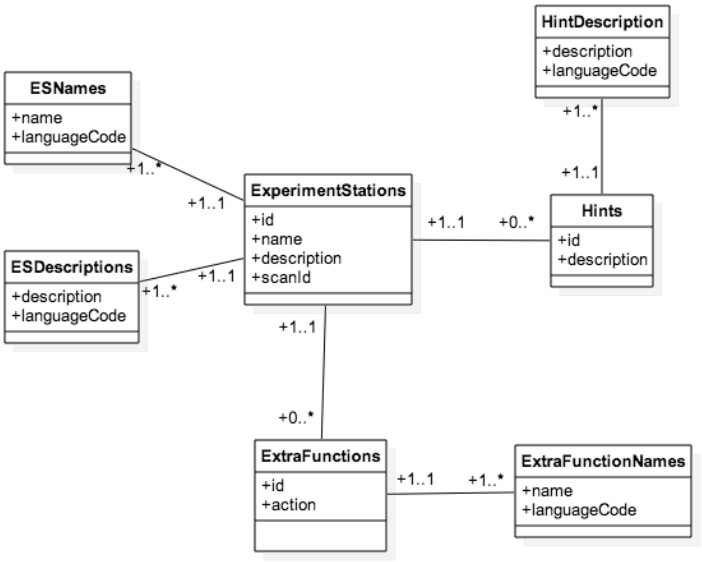


Figure 4: Communication with the restful service

So the best way is to go with the third option, to have separate tables with translations. And when you query for data, you JOIN on the primary key of the table and ask for the languages you want to have returned. After we implemented this, the database for the experimentstations looked like figure 4.

```
// GET api/experimentstation
[HttpGet]
public IEnumerable<ExperimentStation> Get([FromQuery]string language)
{
    // var language = Request.Params["Accept-Language"];
     if(String.IsNullOrEmpty(language)){
         language = "en_US";
     }
     var stations = _context.ExperimentStations
                     .Include(ex => ex.Hints)
                     .Include(ex => ex.ExtraFunctions)
                     .Include(ex => ex.ESNames)
                     .Include(ex => ex.ESDescriptions)
                     .OrderBy(experiment => experiment.ExperimentStationID)
                     .ToList();
     stations.ForEach(x => x.removeUnusedLanguage(language));
     return stations;
}
```

Listing 3: C# Multi-language GET call Example.

Now we can retrieve the experimentstation by including all the other tables, and by letting the user ask for a specific language through the query string, you only get the language you want. We see how this is coded in the example above. An example call to this endpoint would be:
/api/experimentstation?language=nb
returning all experimentstations with text in Norwegian.

3 RUP 4+1

3.1 Logical view

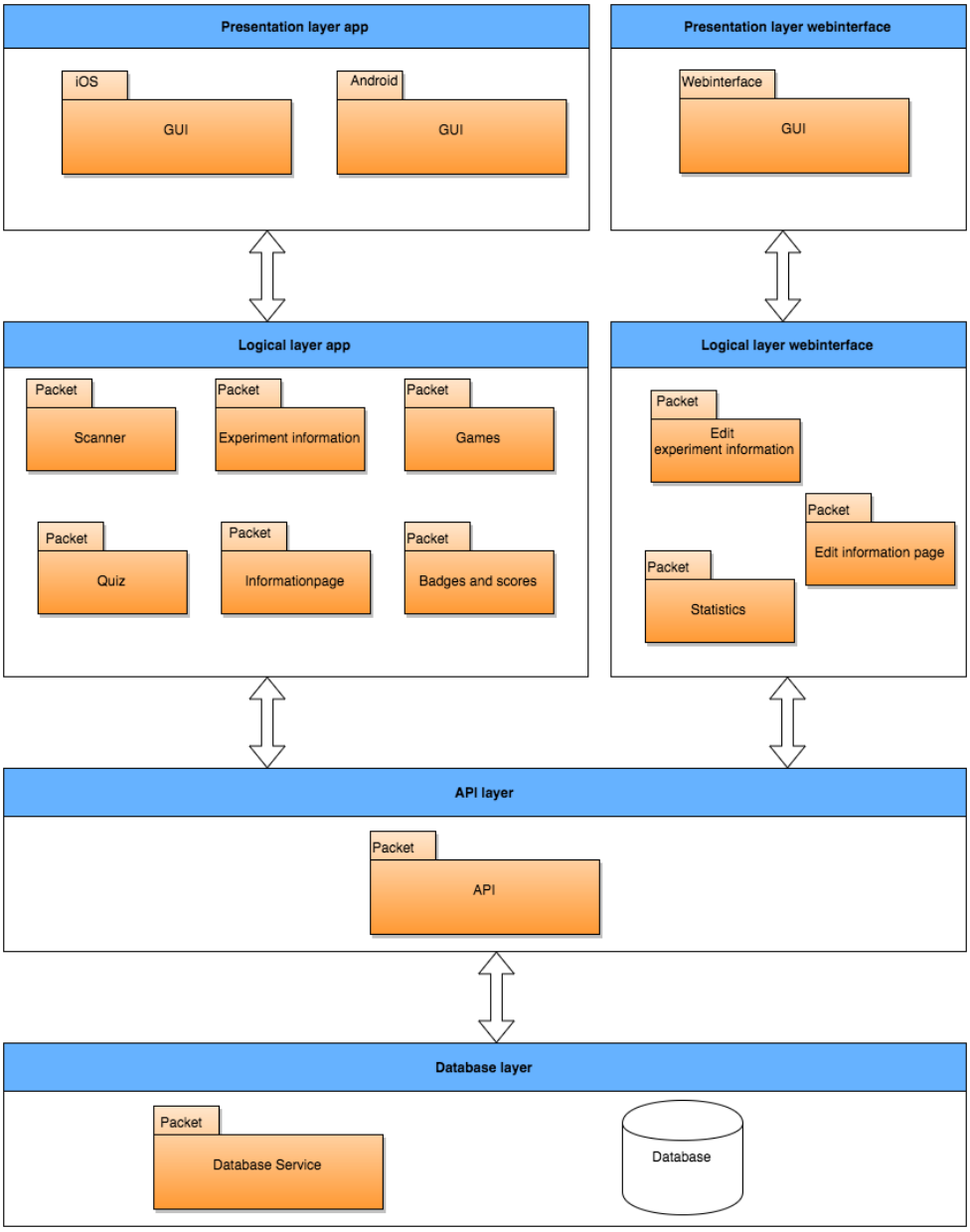


Figure 5: Process view diagram

Presentation layer

The presentation layer is presenting a graphical interface to the user. This layer consists of 3 packets, one for the iOS app, one for the android app and one for the web interface. All the packets are specially adapted to each view.These packets are based on packets from the logical layer.

Logical layer

The logical layer contains of packets thats holds all the functionality and business logic of the system and handles all requests to and from the presentation layer and the API layer.

API layer

The logical layer will use the API-packet to get data from the database and to keep the database updated. The API-packet will receive calls from the packets in the logical layer and provide the proper formatting for further processing.

Database layer

The database layer exposes the data to the API layer. All communication to and from the database is handled by the database service.

3.2 Process view

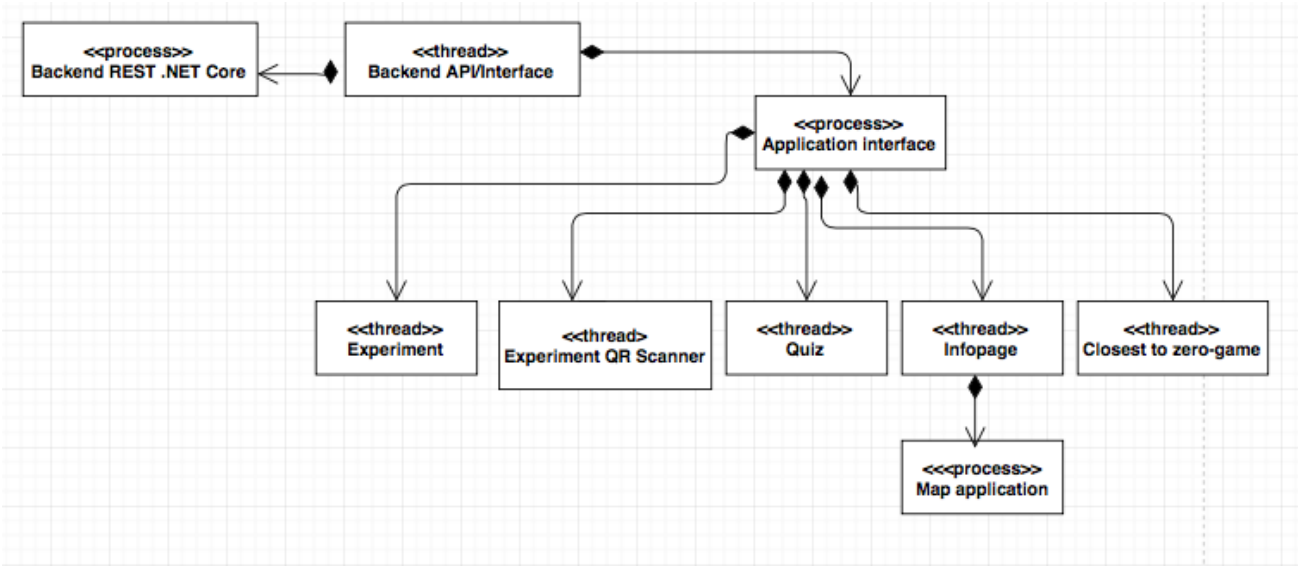


Figure 6: Process view diagram

Diagram shows the process view for the application. ”Application interface” represent the application, the application acts as an interface to the user for the different functionalities - Experiment, Experiment QR scanner, Quiz, infopage and closes to zero-game. Using the infopage, user can also get road description to the center which will open the phones default map application. Application has an interface which it can use to communicate with the backend server.

3.3 Deployment view

Since the backend system is made as a virtual machine park we tried to make use of all knowledge collected through subject IMT3441. To make this system tackle much incoming traffic, but also be small

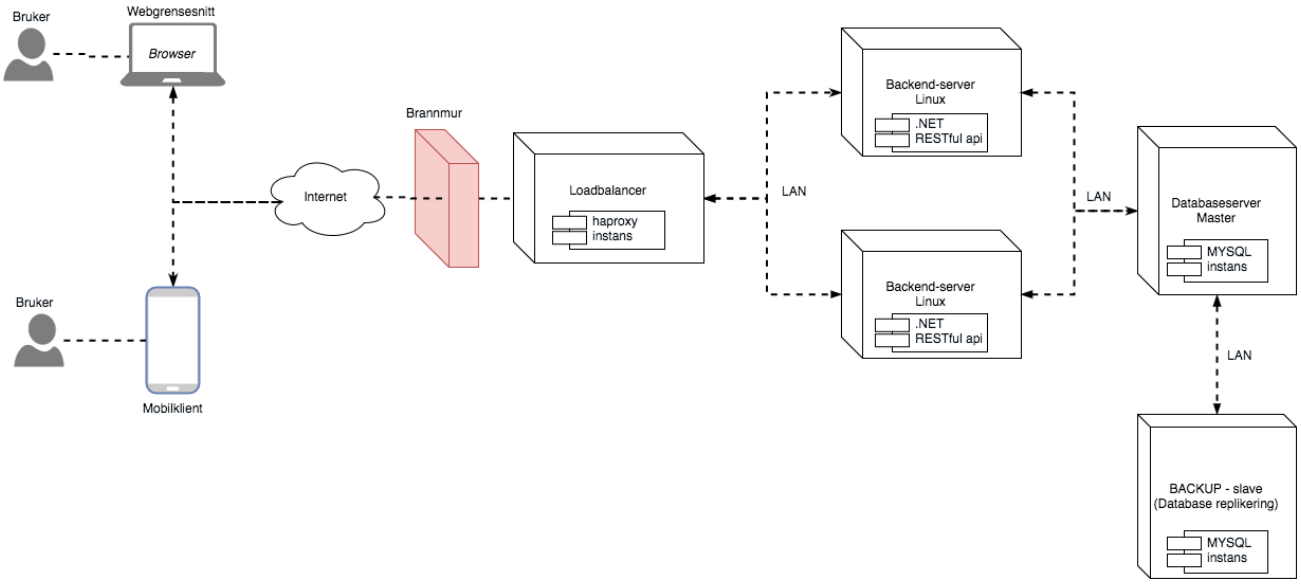


Figure 7: Deployment view diagram

and cheap as long as the traffic is small we decided to go with a loadbalancer in the front of the web-servers. This makes it easy to have one webserver in situations when the traffic i lower and increase the capacity when the load is expected to be higher. This is a really good option for Vitensenteret because on the weekdays - with the center closed - the traffic on the servers is expected to be much lower than in the weekends when the center is open for the public. We can with this knowledge write scripts that dynamically adds and removes webserver behind the loadbalancer when the traffic changes.

Behind the webserver we have a database server, that is replicated to a backup server. The way this is done is through database replication which means that all WRITE commands that changes the database is done on the main database server (the master) and the backup server (the slave) receives what the master server did, and replicated this. This also means that read commands can be done on both the master and the slave, and can therefore reduce traffic.

4 Universal design

It's estimated that 60% of an IT-solutions user base are dependent on "available services",[15] also known as universal design. That involves incorporation of universal design solutions from day one in the development process.

When designing the application we use "Løsningsforslag for Web" from Direktorat for forvaltning og IKT to help us make better design decisions. All new webpages and webapplications made in Norway are legally obliged to fulfill at least 35 out of the 61 success criterias in the WCAG 2.0-standard. In principle this also applies to new mobile applications, but many of the demands are not directly transferable from web to mobile.

WCAG 2.0 defines "Conformance", which are a leveling system for deciding if content meets the WCAG 2.0-standard. The conformance levels are defined as Level A, AA and AAA.[16]

When designing it's crucial to know your user base. An application that are designed for use by pilots don't have to be universally designed for other people than the pilots. This is in contrast to our application, whom has a large user base.

As described in the PACT-analysis in Mappe 3 the application need to be designed for people with color blindness, hearing difficulties, motoric challenges, foreigners and others.

Universal design are done so people with disabilites can fully use IT solutions. Universally designed solutions will also make the product better when there are problems with environmental surroundings.[17]

An example of an environmental problem at the Vitensenter is the variation of light strength between rooms and experiments, this can impact both the QR code scanner and the ability to see the mobile screen. Another example of an environmental problem is when using the application walking around the center, this will more or less give users more motoric problems, universal design will help these challenges.

4.1 Language

Potential users of the application are tourists that don't know Norwegian. The application will be internationalized so user can switch the language to English. We don't see it necessary at this time in the development life cycle to support more languages than Norwegian and English.

4.2 Vision problems

Designing usability for people with vision problems involve careful consideration of color, color contrast, font size and font type

We are going to use font type *Verdana*[18] in the application with a font size of 12 points. This is an approved combination in universal design.[19]

Colors used in the application will be chosen from the "conservative 7-color palette adapted for color blindness"[20] resulting from the works of Bang Wong in 2011 related to colors and color blindness. [21][22].

Final decision on combination of colors will be done after prototyping and discussions with product owner.

4.3 Motoric problems

Users with motoric problems may need customization of the app to get a good experience. Buttons and touch surfaces on the app will be made sufficiently large to meet such demands.

4.4 Platformspecific customization

When developing native applications for both iOS and Android, the developers need to take platform specific design considerations . [23][24] Unlike Android-phones the iPhone doesn't have an embedded back button for navigation. On the iOS app we will handle this by embedding a back button in the GUI in the top left corner, placing it in the top left corner is considered best practice.[23]

The tag bar in the applications will follow the design standards for both iOS and Android-phones, the menubar will be at the bottom of the screen for iPhones and at the top for Android-phones.

Both Android and iOS-developers can code applications that will hide the status bar, we will not do this as we don't see it necessary.

The applications will be coded responsively ensuring it looks good on different devices, pictures will be scaled up and down to fit all different screen sizes as good as possible.

References

[1] Hybrid vs Native Mobile App Development. Decide in 5 minutes! [Blog-post]; 2015 [cited 24/11-2016]. Available from: http://julyrapid.com/hybrid-vs-native-mobile-app-decide-5-minutes/.

[2] Mobile OS market share 2016 — Statista [Online]; 2016 [cited 14/11-2016]. Available from: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/.

[3] Why to use web services instead of direct access to a relational database for an android app? [Online]; 2012 [cited 24/11-2016]. Available from: http://softwareengineering.stackexchange.com/questions/170463/why-to-use-web-services-instead-of-direct-access-to-a-relational-database-for-an.

[4] Can an Android App connect directly to an online mysql database [Online]; 2013 [cited 14/11-2016]. Available from: http://stackoverflow.com/questions/19217835/can-an-android-app-connect-directly-to-an-online-mysql-database.

[5] Representational state transfer [Wikipage]; 2016 [cited 14/11-2016]. Available from: https://en.wikipedia.org/wiki/Representational_state_transfer.

[6] JSON [Wikipage]; 2016 [cited 14/11-2016]. Available from: https://en.wikipedia.org/wiki/JSON.

[7] What is NFC? [Online]; 2016 [cited 14/11-2016]. Available from: https://www.unitag.io/nfc/what-is-nfc#titleName2.

[8] iBeacons price [online]; 2016 [cited 14/11-2016]. Available from: http://www.ebay.com/sch/i.html?_from=R40&_trksid=p2050601.m570.l1313.TR0.TRC0.H0.Xble+beacon.TRS0&_nkw=ble+beacon&_sacat=0.

[9] Kode etiketter [online]; 2016 [cited 14/11-2016]. Available from: https://www.cottontrends.no/catalog/kode-etiketter-kode-generator-kode-klistremerker-p-190.html?language=no.

[10] Cisco Global Cloud Index: Forecast and Methodology, 2015–2020 [White paper]; 2015 [cited 29/11-2016]. Available from: http://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf.

[11] Cloud computing [Wikipage]; 2016 [cited 29/11-2016]. Available from: https://en.wikipedia.org/wiki/Cloud_computing.

[12] Elastic Compute Cloud (EC2) Cloud Server & Hosting – AWS [Online]; 2016 [cited 22/11-2016]. Available from: https://aws.amazon.com/ec2/?sc_channel=PS&sc_campaign=acquisition_ND&sc_publisher=google&sc_medium=ec2_b&sc_content=ec2_e&sc_detail=amazon.ec2&sc_category=ec2&sc_segment=154805225652&sc_matchtype=e&sc_country=ND&s_kwcid=AL!4422!3!154805225652!e!!g!!amazon.ec2&ef_id=WDQajwAABTM39pBe:20161122101439:s.

[13] Representational state transfer - Wikipedia [Online]; 2016 [cited 22/11-2016]. Available from: https://en.wikipedia.org/wiki/Representational_state_transfer.

[14] Dependency injection - Wikipedia [Online]; 2016 [cited 24/11-2016]. Available from: https://en.wikipedia.org/wiki/Dependency_injection.

[15] for forvaltning og IKT D. En introduksjon til universell utforming [Video]; 2016 [cited 17/11-2016]. Available from: https://player.vimeo.com/video/158629958?color=ffffff&title=0&byline=0&portrait=0#t=0415.

[16] Understanding Conformance [Online]; 2016 [cited 17/11-2016]. Available from: https://www.w3.org/TR/UNDERSTANDING-WCAG20/conformance.html.

[17] Nå må webutviklere forholde seg til helt nye regler - Tu.no [Online]; 2016 [cited 22/11-2016]. Available from: http://www.tu.no/artikler/na-ma-webutviklere-forholde-seg-til-helt-nye-regler/230187.

[18] Verdana.gif (864x576) [Online]; 2016 [cited 23/11-2016]. Available from: http://www.identifont.com/samples2/microsoft/Verdana.gif.

[19] Skrift på papir — Norges Blindeforbund [Online]; 2016 [cited 22/11-2016]. Available from: https://www.blindeforbundet.no/universell-utforming/skrift-pa-papir.

[20] colorblindness.palettes.trivial.png (3338x1017) [Online]; 2016 [cited 23/11-2016]. Available from: http://mkweb.bcgsc.ca/colorblind/img/colorblindness.palettes.trivial.png.

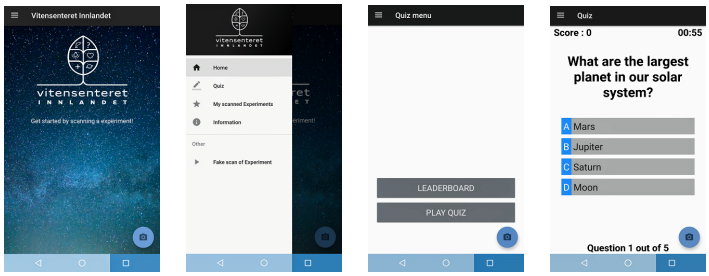
[21] Data Visualization, Design and Information Munging // Martin Krzywinski / Genome Sciences Center [Online]; 2016 [cited 23/11-2016]. Available from: http://mkweb.bcgsc.ca/colorblind/.

[22] Points of view: Color blindness : Nature Methods : Nature Research [Online]; 2016 [cited 23/11-2016]. Available from: http://www.nature.com/nmeth/journal/v8/n6/full/nmeth.1618.html.

[23] Back Button Considerations in Cross Platform Mobile App Design - New Signature [Online]; 2016 [cited 22/11-2016]. Available from: https://newsignature.com/articles/back-button-considerations-in-cross-platform-mobile-app-design/.

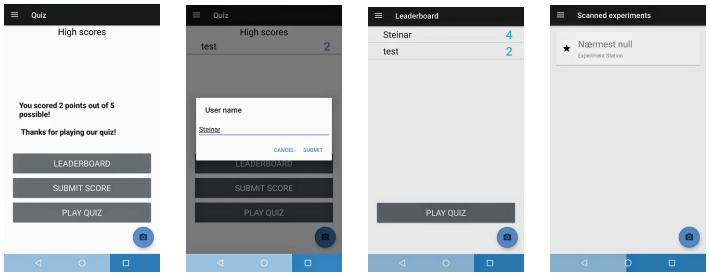
[24] iOS and Android Design Guidelines Cheat Sheet - Enterprise Mobile Backend as a Service — Kinvey [Online]; 2016 [cited 22/11-2016]. Available from: https://www.kinvey.com/ios-and-android-design-guidelines-cheat-sheet/.

B.3 Mobile Development Project



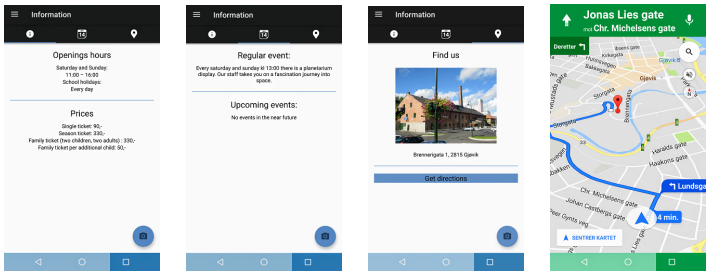
(a) Home (b) Drawer (c) Quiz menu (d) Quiz

Figur 1: Pilotapplikasjon hjemmeskjerm, meny og quiz



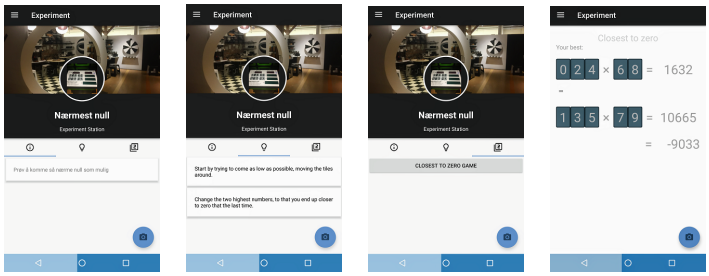
(a) Quiz result (b) Enter username (c) Leaderboard (d) Scanned experiments

Figur 2: Pilotapplikasjon resultat og ledertavle, skannede eksperimenter



(a) Information (b) Events (c) Address (d) Navigation

Figur 3: Pilotapplikasjon informasjonsside



(a) Experiment info (b) Experiment hints (c) Experiment extras (d) Closest to zero

Figur 4: Pilotapplikasjon eksperimentinfo, hints og ekstrafunksjoner, og spillet *Nærmest null*

C Grupperegler

1. Steffen Granberg fungerer som prosjektleder - ansvarlig for å følge opp møtetidspunkt, avtaler, at arbeidsoppgaver fordeles og utføres og kontakt med arbeidsgiver. Prosjektleder kan signere på vegne av gruppen.
2. Hvert gruppemedlem forplikter å jobbe minimum 30 timer per uke. Gruppemedlemmene skal minst ha et fysisk møte i løpet av en uke, men har som felles mål å sitte 4 dager sammen per uke.
3. Timene skal loggføres og legges frem for gruppen på prosjektmøter.
4. Det skal gis beskjed om en ikke har mulighet til å møte opp til avtalt tid.
5. Hvis et gruppemedlem ikke utfører arbeid i tråd med regler og/eller forventningen fra de andre vil rutine ved “alvorlige brudd på reglene” følges.
6. Alle kostnader deles likt på alle tre medlemmer av gruppen.

Arbeidsregler

1. Ved felles arbeidsøkter skal det konsentreres om arbeid. Bruk av tiden til personlige interesser ikke relatert til arbeidet er ikke tillatt.
2. Møteplikt på prosjektmøter, statusmøter, møter med arbeidsgiver og møter med veileder.
3. Gruppen plikter tidlig å informere dersom de ikke er fornøyd med innsatsen til enkelt gruppemedlemmer.
4. Det skrives referat etter alle prosjekt og statusmøter.
5. Det kan gis dispensasjon fra møteplikten ved god grunn og varsel i forkant.
6. Regler for kodestandard og verktøybruk skal følges av alle medlemmer.

Rutine ved “alvorlige brudd på reglene”

1. Samtale(r) mellom alle gruppemedlemmene om problemet
2. Skriftlig advarsel fra andre medlemmer der:
    a. Regelbrudd blir påpekt
    b. Hva som kreves for å oppveie for tapt arbeid
    c. Redegjøre for konsekvensene ved videre brudd på reglene
3. Samtale mellom hele gruppen og veileder.
4. Skriftlig ekskludering fra gruppen signert av alle andre.

D Prosjektavtale

NTNU

Norges teknisk-naturvitenskapelige universitet

PROSJEKTAVTALE

mellom NTNU v/Avd. Informatikk og Medieteknikk (NTNU/AIMT) (utdanningsinstitusjon), og

Vitensenteret Innlandet (oppdragsgiver), og

Ole Andre Slettum, Steinar Opphus og Steffen Granberg (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 16/01-17 til 16/05-17 .

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der AIMT 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 AIMT 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 Gjøvik/AIMT. 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. AIMT 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 faglærer/veileder og sensor (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 åpen 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 dekan 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 AIMT 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/AIMT og/eller studenter har interesser.

6. 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.

7. 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.

8. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i Fronter. I tillegg leveres et eksemplar til oppdragsgiver.

9. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av AIMT er det dekan/prodekan som godkjenner avtalen.

10. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og AIMT 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 AIMT som partner.

11. Når NTNU/AIMT også opptrer som oppdragsgiver, trer NTNU/AIMT inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

12. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. 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.

13. Deltakende personer ved prosjektgjennomføringen:

NTNU/AIMTs veileder (navn): Tom Røise

Oppdragsgivers kontaktperson (navn): Petra Skoglund

Student(er) (signatur): _____ dato 18/1-17

_____ dato 18/1-17

_____ dato 18/1-17

_____ dato _____

SØ 93

Norges teknisk-naturvitenskapelige universitet

Oppdragsgiver (signatur): _____ dato 13/1-17

Signert avtale leveres digitalt i Fronter(IMT3912)
Godkjennes digitalt av AIMTs dekan

Om papirversjon med signatur er ønskelig, må papirversjon leveres til AIMT i tillegg.
Plass for evt sign:
AIMT Dekan/prodekan (signatur): _____ dato _____

E Statusrapport

Statusrapport - bachelorprosjekt

Viten i Senter

24/04-17
Medlemmer:

1. Steinar Opphus
2. Ole Andre Slettum
3. Steffen Granberg

1. **Status for:**
    a. **Planlegging**
        i. Planleggingsfasen ble avsluttet 30.januar. Her ble det det laget et forprosjekt, med milepæler som skal overholdes underveis i prosjektet.
    b. **Organisering av gruppens arbeid og ansvarsområder.**
        i. Henviser til prosjektplanen, dette ble beskrevet der. Gruppens arbeid og ansvarsområder er delt inn og planlagt.
    c. **Klargjøring av problemstilling (eks. systemering).**
        i. Avgrensning av oppgaven ble gjort i samarbeid med Vitensenteret i slutten av forprosjektet.
    d. **Løsningsmetode (eks. koding).**
        i. Koding, testing og rapportskriving
    e. **Rapportskriving**
        i. Rapportskriving er påbegynt, og godt underveis.
2. **Muligheter trusler/problemer?**
    a. Vi er fornøyd med fremgangen, og ser per dags dato ingen trusler.
3. **Hva er avsluttet? Hvilke oppgaver er ferdige?**
    a. Koding av nye funksjoner er avsluttet
    b. De resterende ukene skal gå med til refactorering, testing og rapportskriving
4. **Tidsfrister**
    a. Alle tidsfristene er overholdt, og vi har ikke overskredet noe i prosjektplanen enda.
5. **Motivasjon, samarbeid**
    a. Gruppas motivasjon er veldig god. Gruppesamarbeidet har fungert på en utmerket måte.
6. **Veilederkontakt**
    a. Veilederkontakt oppleves som upåklagelig. Vi er veldig fornøyd med samarbeidet.

F High-level use case-beskrivelser

Use case navn: Vis veibeskrivelse
Aktører: Potensielle besøkende
Mål: Se veibeskrivelse til Vitensenteret
Beskrivelse: Potensiell besøkende trykker på knapp som sender han til telefonens innebygde kartapplikasjon. Senterets koordinater blir sendt med som destinasjon.

Use case navn: Vise åpningstider
Aktører: Potensielle besøkende
Mål: Se Vitensenterets åpningstider
Beskrivelse: Potensielle besøkende navigerer til informasjonssiden hvor han ser senterets åpningstider

Use Case navn: Se kontaktinformasjon
Aktører: Potensielle besøkende
Mål: Se telefonnummer og link til senterets nettside
Beskrivelse: Potensiell besøkende navigerer til informasjonssiden hvor han ser kontaktinformasjon - telefonnummer og link til nettside. Informasjonen er klikkbar - trykker man på telefonnummer får man spørsmål om å ringe dette nummeret, trykker man på linken blir man sendt til nettsiden.

Use case navn: Skanne eksperiment
Aktører: Besøkende
Mål: Skanne og få tilgang til eksperimentstasjonens side
Beskrivelse: Besøkende bruker telefonen sin til å skanne QR-koder som han finner på eksperimentet. Applikasjonen sender brukeren videre til en aktivitet som viser mer informasjon om eksperimentet - info, hint og eventuell ekstrafunksjonalitet

Use Case navn: Motta varsler fra beacons
Aktører: Besøkende
Mål: Brukere er inne på senteret og mottar informasjon som pushes fra beacons-enheter
Beskrivelse: Brukeren har bluetooth aktivert og mottar informasjon fra beacons - for eksempel 'Velkommen til senteret' og 'Nå er det planetarievisning'

Use Case navn: Registrere quizpoeng
Aktører: Besøkende og bruker av mobilapplikasjon
Mål: Brukeren ønsker å registrere sine quizpoeng til topplisten etter at han har gjort en quiz
Beskrivelse: Brukeren trykker på "Last opp poengsum"-knapp og blir spørt om navn han vil laste opp poengsummen med valgfritt navn. Hvis noen har brukt navnet før vil han få beskjed om det og igjen spørres om et annet navn.

Use Case navn: Se toppliste på quiz
Aktører: Besøkende og bruker av mobilapplikasjon
Mål: Brukeren kan se toppliste på quizzen han har valgt eller nettopp har gjennomørt
Beskrivelse: Brukeren velger kategori og quiz og blir sendt til siden hvor han kan starte quizzen, her ser han også quizzens toppliste. Når brukeren er ferdig med en quiz er det en knapp hvor han kan trykke seg videre til topplisten

Use Case navn: Svare på quiz
Aktører: Besøkende og bruker av mobilapplikasjon
Mål: Brukeren kan se toppliste på quizzen han har valgt eller nettopp har gjennomørt
Beskrivelse: Brukeren velger kategori og quiz og blir sendt til siden hvor han kan starte quizzen, her ser han også quizzens toppliste. Når brukeren er ferdig med en quiz er det en knapp hvor han kan trykke seg videre til topplisten

Use Case navn: Se skannede eksperimenter
Aktører: Bruker av mobilapplikasjon
Mål: Brukeren ønsker å se sine tidligere skannede eksperimenter
Beskrivelse: Brukeren trykker på Skannetfanen og får oversikt over sine tidligere skannede eksperimenter, her kan han trykke seg videre til hvert individuelle eksperiment

Use case navn: Spille “closest to zero”
Aktører: Bruker av mobilapplikasjon
Mål: Aksessere og spille spill
Beskrivelse:Etter å ha skannet dette eksperimentet på senteret, har brukeren tilgang til spillet “Closest to zero”. Closest to zero er et eksperiment hvor man legger en rekkefølge av brikker med tall fra 0 til 9, disse er ganget sammen og lagt til hverandre på en spesifikk måte. Utfordringen er å finne en kombinasjon som gjør at resultatet av regnestykket blir 0.

Use Case navn: Spille ”hjernespill”
Aktører: Bruker av mobilapplikasjon
Mål: Aksessere og spille spill
Beskrivelse: Brukeren får tilgang til hjernespill ved å skanne det på senteret. ”Hjernespill” har fire knapper med ulike farger, disse knappene lyser opp i en bestemt sekvens som brukeren må gjenta. Hver gang brukeren gjentar sekvensen riktig startes en ny sekvens hvor antall deler i sekvensen øker med en hver gang.

Use Case navn: Skifte språk
Aktører: Bruker av mobilapplikasjon
Mål: Skifte språk på innhold i applikasjonen
Beskrivelse: Ved hjelp av innstilling inne i applikasjonen, skal en bruker kunne endre språket på innholdet som lastes ned fra backend (språk i brukergrensesnittet er basert på global telefoninnstilling).

Use Case navn: Legge til og endre quizspørsmål
Aktører: Ansatte på vitensenteret
Mål: Legge til spørsmål i spørsmålsbanken
Beskrivelse: Ansatte kan legge til spørsmål relatert til spesifikke eksperimenter eller tema

Use Case navn: Legge til og endre informasjon om eksperimentstasjon
Aktører: Ansatt på Vitensenteret
Mål: Legge til ny stasjon i appen
Beskrivelse: Legge til ny stasjon som kan bli skannet ved å legge til en QR kode på eksperimentet. Brukeren kan legge til informasjon som navn, komplementær informasjon, video, bilder og spørsmål

Use Case navn: Slette oppføring i toppliste
Aktører: Ansatt på Vitensenteret
Mål: Fjerne oppføring i topplisten for quiz
Beskrivelse: Ved hjelp av web-administrasjonsverktøyet kan en ansatt fjerne en oppføring i topplisten, f.eks. hvis banneord er brukt i brukernavnet

Use Case navn: Legge til og endre informasjon om rom
Aktører: Ansatt på Vitensenteret
Mål: Legge til endre endre informasjon om rom eksperimentstasjoner kan knyttes til
Beskrivelse: Ved hjelp av web-administrasjonsverktøyet kan en ansatt legge til eller endre informasjon om rom/områder på Vitensenteret som inneholder ett eller flere eksperimentstasjoner.

Use Case navn: Legge til og slette språk
Aktører: Ansatt på Vitensenteret
Mål: Legge til eller slette språk som appbrukerne kan velge mellom
Beskrivelse: Ved hjelp av web-administrasjonsverktøyet kan en ansatt legge til eller slette språk som kan brukes i applikasjonen.

Use Case navn: Legge til quizkategori
Aktører: Ansatt på Vitensenteret
Mål: Legge til en kategori som quizzer kan sorteres under
Beskrivelse: Ved hjelp av web-administrasjonsverktøyet kan en ansatt legge til en ny quizkategori.

Use Case navn: Legge til ny quiz
Aktører: Ansatt på Vitensenteret
Mål: Lage ny quiz og velge hvilke spørsmål den skal inneholde
Beskrivelse: Ved hjelp av web-administrasjonsverktøyet kan en ansatt lage en ny quiz, gi den et navn, velge kategori og hvilke spørsmål den skal inneholde.

Use Case navn: Endre passord
Aktører: Ansatt på Vitensenteret
Mål: Endre passord
Beskrivelse: Ved hjelp av web-administrasjonsverktøyet skal en ansatt kunne endre passord på sin bruker.

Use Case navn: Legge til og endre bruker
Aktører: Administrator
Mål: Legge til ny bruker og gi/fjerne adminrettigheter
Beskrivelse: En administrator kan legge til ny bruker og eventuelt gi den adminrettigheter, eller fjerne rettighetene til en annen adminbruker.

G Suppress av Lint- og Sonaradvarsler

G.1 Android

Filnavn	Suppress	Merknad
AndroidManifest.xml	tools:ignore=GoogleAppIndexingWarning"	Appen trenger ikke å være søkbar på google foreløpig.
strings.xml	tools:ignore=PluralsCandidate"	Det er ikke behov for annet en flertallsbeskrivelse etter tallet i strengen
HomeScreenActivity.java	@SuppressWarnings({"UnusedParameters",squid:S1172"})	View må være med som parameter på enkelte funksjoner, selv om den ikke brukes i metoden.
QRCodeScanner.java	@SuppressWarnings(deprecation")	Bruker foreldet API (android.hardware.Camera) for å sikre bakoverkompatibilitet.
CameraSource.java	@SuppressWarnings(squid:S00112")	Ignorert siden dette er i et nedlastet bibliotek og de generelle er beskrevet godt.
CameraSource.java	@SuppressWarnings(deprecation")	Brukerforeldet API (android.hardware.Camera) for å sikre bakoverkompatibilitet.
CameraSource.java	@SuppressWarnings(EmptyMethod")	Dette er et interface.
ApiClient.java	@SuppressWarnings(SameParameterValue"	Parameter er en konstant nå, men dette kan endres hvis applikasjonen skaleres opp.
Constants.java	@SuppressWarnings(squid:S1313")	Vi syntes det var unødvendig å flytte IP-adressen ut på fil.
QuizScoreApiUsage	@SuppressWarnings(deprecation")	HTTP.UTF_8 er foreldet fra API 19, men vi trenger å støtte API 18 også.
Getable, Postable, Saveable	@SuppressWarnings("unused")	De er interfaces og blir brukt.
Answer, ExtraFunction, Hint, Question, Quiz, QuizScore	@SuppressWarnings("unused")	Variablene blir brukt av GSON, men det fanger ikke lint opp.
Question.java	@SuppressWarnings({FieldCanBeLocal")	Variabel kan bli brukt i senere versjoner.

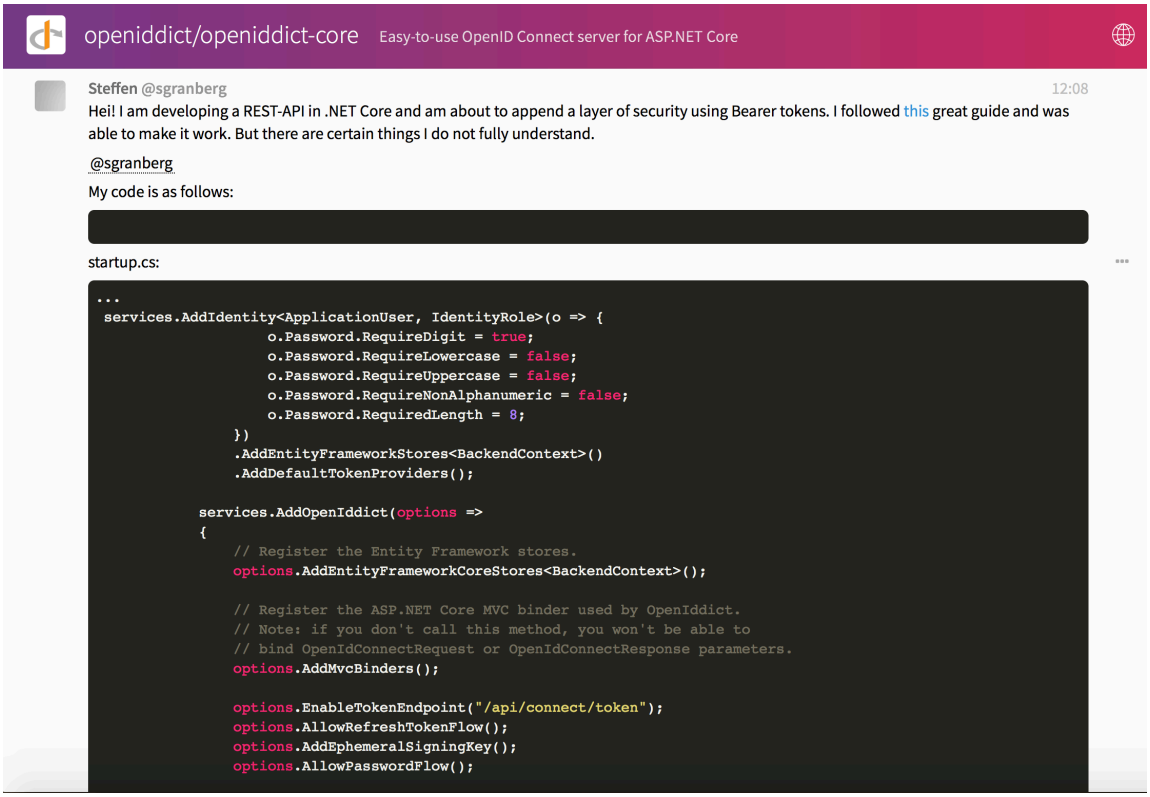
Tabell 1: Suppress av Lint- og Sonaradvarsler i Android

G.2 iOS

Filnavn	Suppress	Merknad
Alle	line_length	Denne ble disablet fordi den kastet en error som gjorde at vi ikke fikk kompilert. Denne warningen er fikset på de aller fleste stedene, men noen steder tok vi en avgjørelse på at det ble mindre lesbart hvis endret det.
Alle	force_cast	Dette er også en warning som gjør at programmet nekter å kompilere. Vi har etter beste evne fjernet alle forcecastene som vi kan, men noens steder har man ikke noe valg. I tillegg er det brukt på en rekke steder i andre bibliotek.
Alle	identifier_name	Denne klaget på lengden på noen klasse- og variabelnavn. Siden for eksempel QuizCategoryViewController er mer lesbart enn noen andre versjoner, ignorerte vi disse advarslene med vilje.

Tabell 2: Supress av lintadvarsler i Swiftlinter på iOS

H Samtale Openiddict



openiddict/openiddict-core Easy-to-use OpenID Connect server for ASP.NET Core

Steffen @sgranberg 12:08

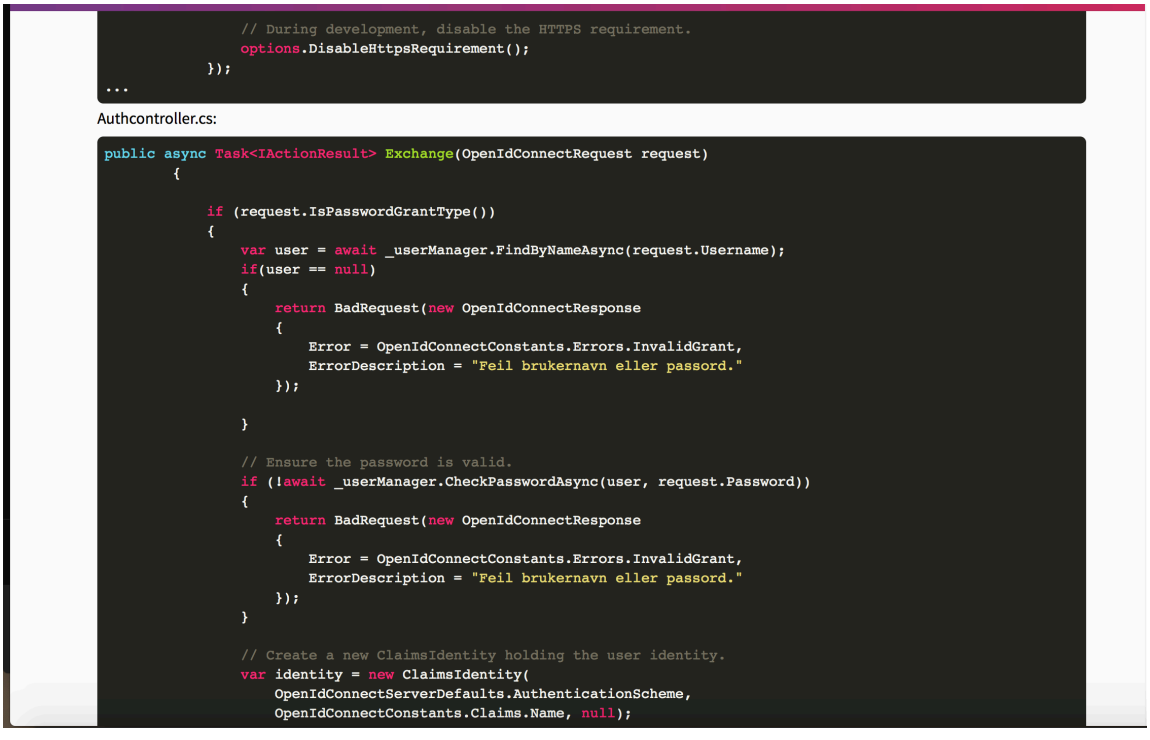
Hei! I am developing a REST-API in .NET Core and am about to append a layer of security using Bearer tokens. I followed this great guide and was able to make it work. But there are certain things I do not fully understand.

@sgranberg

My code is as follows:

startup.cs:

```
...
 services.AddIdentity<ApplicationUser, IdentityRole>(o => {
                    o.Password.RequireDigit = true;
                    o.Password.RequireLowercase = false;
                    o.Password.RequireUppercase = false;
                    o.Password.RequireNonAlphanumeric = false;
                    o.Password.RequiredLength = 8;
                })
                .AddEntityFrameworkStores<BackendContext>()
                .AddDefaultTokenProviders();

            services.AddOpenIddict(options =>
            {
                // Register the Entity Framework stores.
                options.AddEntityFrameworkCoreStores<BackendContext>();

                // Register the ASP.NET Core MVC binder used by OpenIddict.
                // Note: if you don't call this method, you won't be able to
                // bind OpenIdConnectRequest or OpenIdConnectResponse parameters.
                options.AddMvcBinders();

                options.EnableTokenEndpoint("/api/connect/token");
                options.AllowRefreshTokenFlow();
                options.AddEphemeralSigningKey();
                options.AllowPasswordFlow();
```



```
                // During development, disable the HTTPS requirement.
                options.DisableHttpsRequirement();
            });
...
```

Authcontroller.cs:

```
public async Task<IActionResult> Exchange(OpenIdConnectRequest request)
        {

            if (request.IsPasswordGrantType())
            {
                var user = await _userManager.FindByNameAsync(request.Username);
                if(user == null)
                {
                    return BadRequest(new OpenIdConnectResponse
                    {
                        Error = OpenIdConnectConstants.Errors.InvalidGrant,
                        ErrorDescription = "Feil brukernavn eller passord."
                    });

                }

                // Ensure the password is valid.
                if (!await _userManager.CheckPasswordAsync(user, request.Password))
                {
                    return BadRequest(new OpenIdConnectResponse
                    {
                        Error = OpenIdConnectConstants.Errors.InvalidGrant,
                        ErrorDescription = "Feil brukernavn eller passord."
                    });
                }

                // Create a new ClaimsIdentity holding the user identity.
                var identity = new ClaimsIdentity(
                    OpenIdConnectServerDefaults.AuthenticationScheme,
                    OpenIdConnectConstants.Claims.Name, null);
```

```
                }

                // Create a new ClaimsIdentity holding the user identity.
                var identity = new ClaimsIdentity(
                    OpenIdConnectServerDefaults.AuthenticationScheme,
                    OpenIdConnectConstants.Claims.Name, null);
                    // Add a "sub" claim containing the user identifier, and attach
                    // the "access_token" destination to allow OpenIddict to store it
                    // in the access token, so it can be retrieved from your controllers.
                    identity.AddClaim(OpenIdConnectConstants.Claims.Subject,
                    "71346D62-9BA5-4B6D-9ECA-755574D628D8",
                    OpenIdConnectConstants.Destinations.AccessToken);

                    identity.AddClaim(OpenIdConnectConstants.Claims.Name, request.Username,
                                      OpenIdConnectConstants.Destinations.AccessToken
                                     );

                var principal = new ClaimsPrincipal(identity);

                // Generate a new token and return an OAuth2 token response.
                return SignIn(principal, OpenIdConnectServerDefaults.AuthenticationScheme);
            }
            throw new InvalidOperationException("The specified grant type is not supported.");
        }
```

My questions is as follows:

1. Is this setup something I can be sure is secure?
2. This does not use JWT but encrypted token as i Understand, does this mean that the tokens is stored on the server?
3. The code ```

```

Steffen @sgranberg 12:14

```
identity.AddClaim(OpenIdConnectConstants.Claims.Subject,
                  "71346D62-9BA5-4B6D-9ECA-755574D628D8",
                  OpenIdConnectConstants.Destinations.AccessToken);
```

What is "71346D62-9BA5-4B6D-9ECA-755574D628D8", ? and cant this be replaced by something else?

Sorry for the many post, new to gitter and this markdown. Hopefully you can point me in the right direction or help :)

Ruben de la Torre @rubit0 12:21

hello @sgranberg

this is how I have done it: https://github.com/rubit0/YoApp/blob/master/src/YoApp.Backend/Controllers/AuthorizationController.cs

Steffen @sgranberg 12:30

@rubit0 Fairly similar i see. But if you se jwt why do you need a signinmanager, isn't this just for cookie based?

Ruben de la Torre @rubit0 12:34

My code is based on an older tutorial, since it worked flawless I honestly didn't bothered with it. But you are actually correct If I think now about it.

@PinpointTownes can you please look at this?

Kévin Chalet @PinpointTownes 12:35

`71346D62-9BA5-4B6D-9ECA-755574D628D8` is the user identifier (here's it's static because it's a sample :P)

BTW, it's clearly indicated in the code comment :P

"Add a "sub" claim containing the user identifier"

Steffen @sgranberg 12:37

@PinpointTownes Yea, that what was I thought it was, but other tutorials kept not having the sub so i was just wondering if I completely missed something important... Great tutorial btw :)

Kévin Chalet @PinpointTownes 12:38

Older tutorials used `ClaimTypes.NameIdentifier`, which is an older WS-Fed claim type. It's still supported, but I'm considering removing complete support in a future version.

To encourage people to directly use the JWT claim types.

(that live in `OpenIdConnectConstants.Claims`)

@rubit0 if you don't need to ensure the user is allowed to sign in (e.g if he confirmed his email address) or don't need lockout support, then that's fine.

Steffen @sgranberg 12:41
```

To encourage people to directly use the JWT claim types.

(that live in `OpenIdConnectConstants.Claims`)

@rubit0 if you don't need to ensure the user is allowed to sign in (e.g if he confirmed his email address) or don't need lockout support, then that's fine.

Steffen @sgranberg 12:41
@PinpointTownes I want to support refresh tokens, do you have a tutorial about how I can add that to my current setup?

Kévin Chalet @PinpointTownes 12:42
Take a look at the refresh token flow sample in the samples repo.

Steffen @sgranberg 12:43
Thanks!

Kévin Chalet @PinpointTownes 12:43
A blog post about that is planned, but I'm really busy these days :P

Steffen @sgranberg 12:43
I feel yah! :)

Steffen @sgranberg 12:48
@PinpointTownes Last question: my question 2 in the previous post. This setup uses encryptet JWT right? but does that mean that the tokens are stores in the server somewhere? Or is it like jwt, the server sign them and forget about them?

Kévin Chalet @PinpointTownes 12:49
No, it doesn't use JWT.

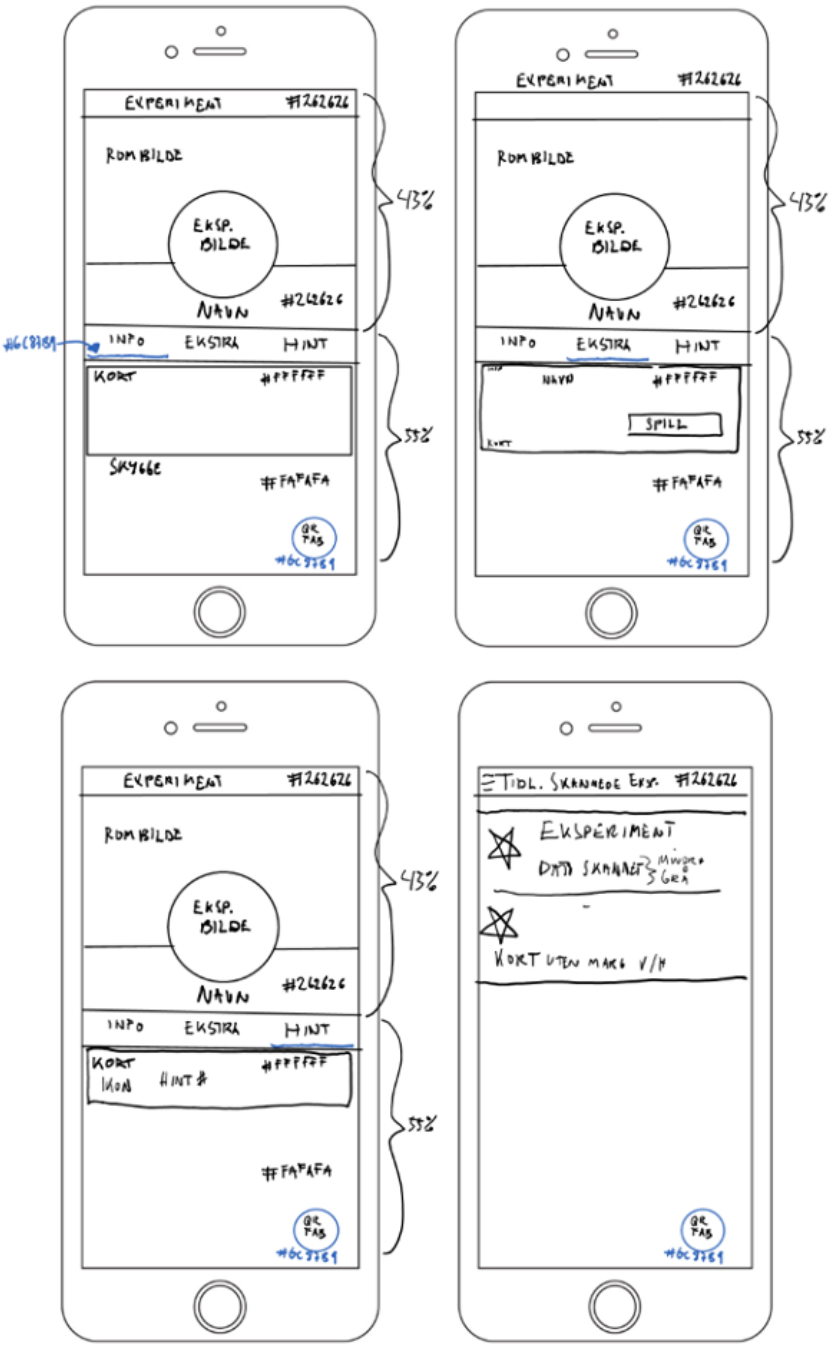
It's used the same encrypted format as ASP.NET Core for its authentication cookies.

And yes, the access tokens are not stored.

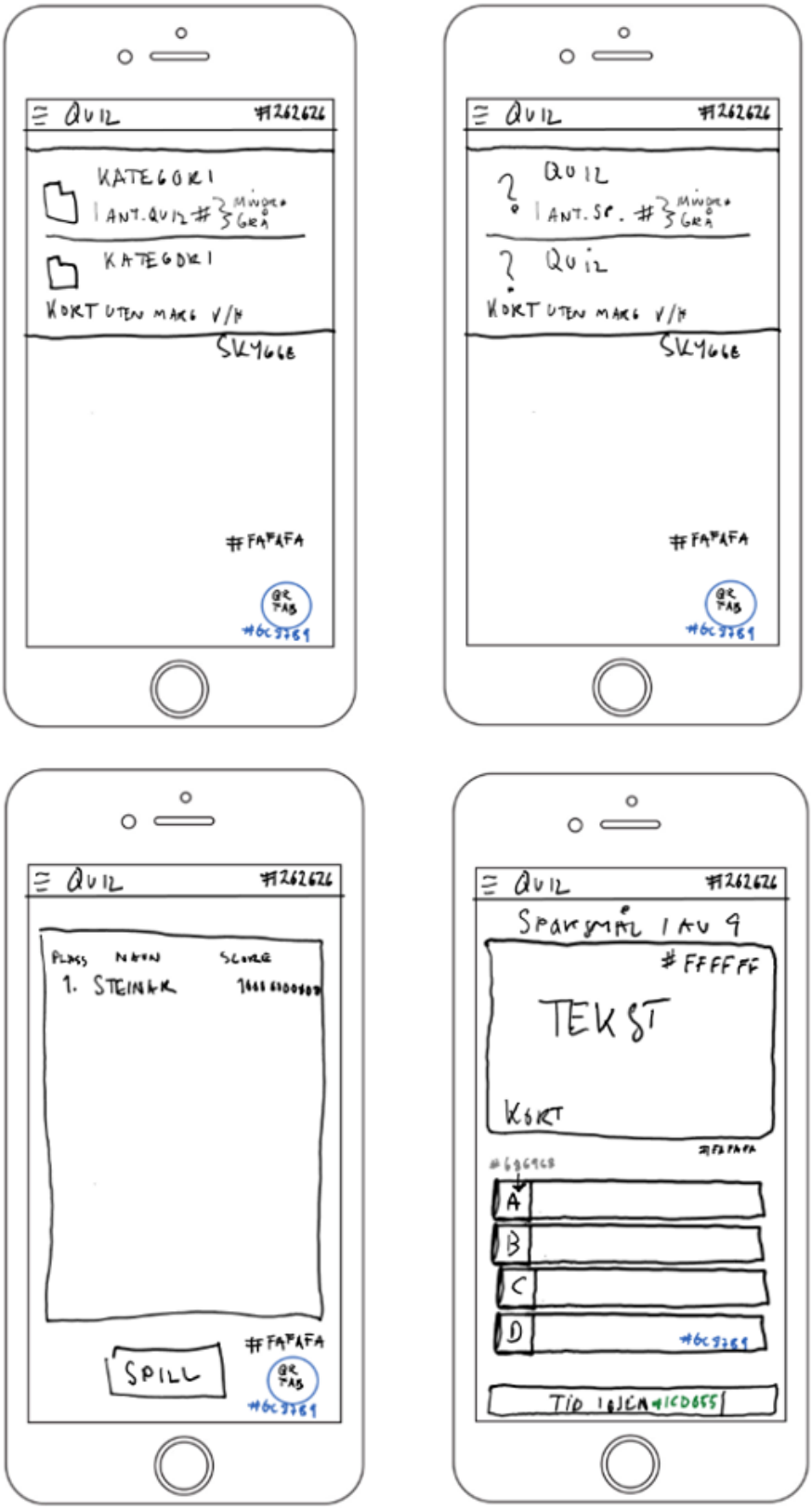
Steffen @sgranberg 12:51
I see, thanks for your time!

Kévin Chalet @PinpointTownes 12:53
;)

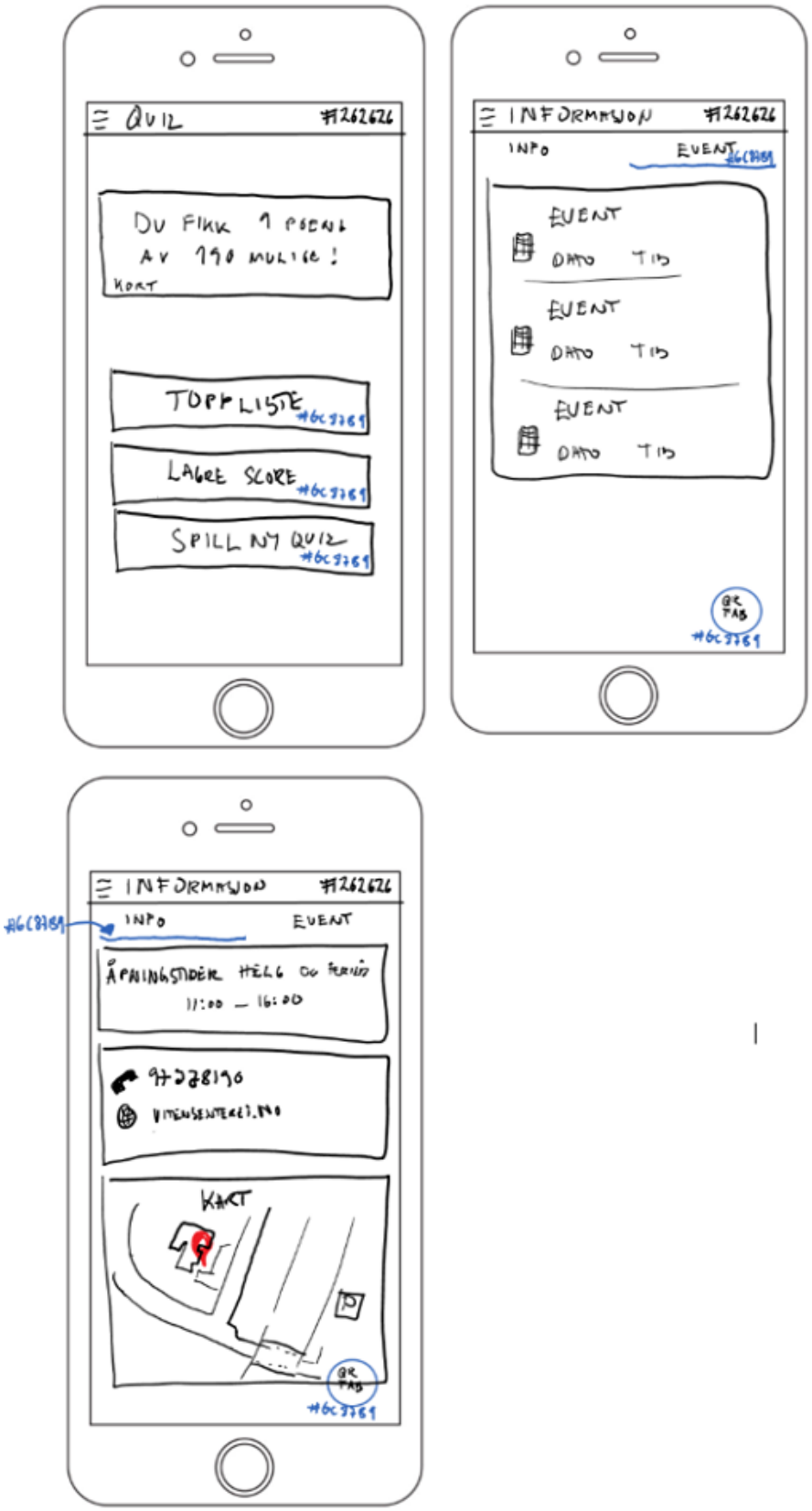
I GUI workshop



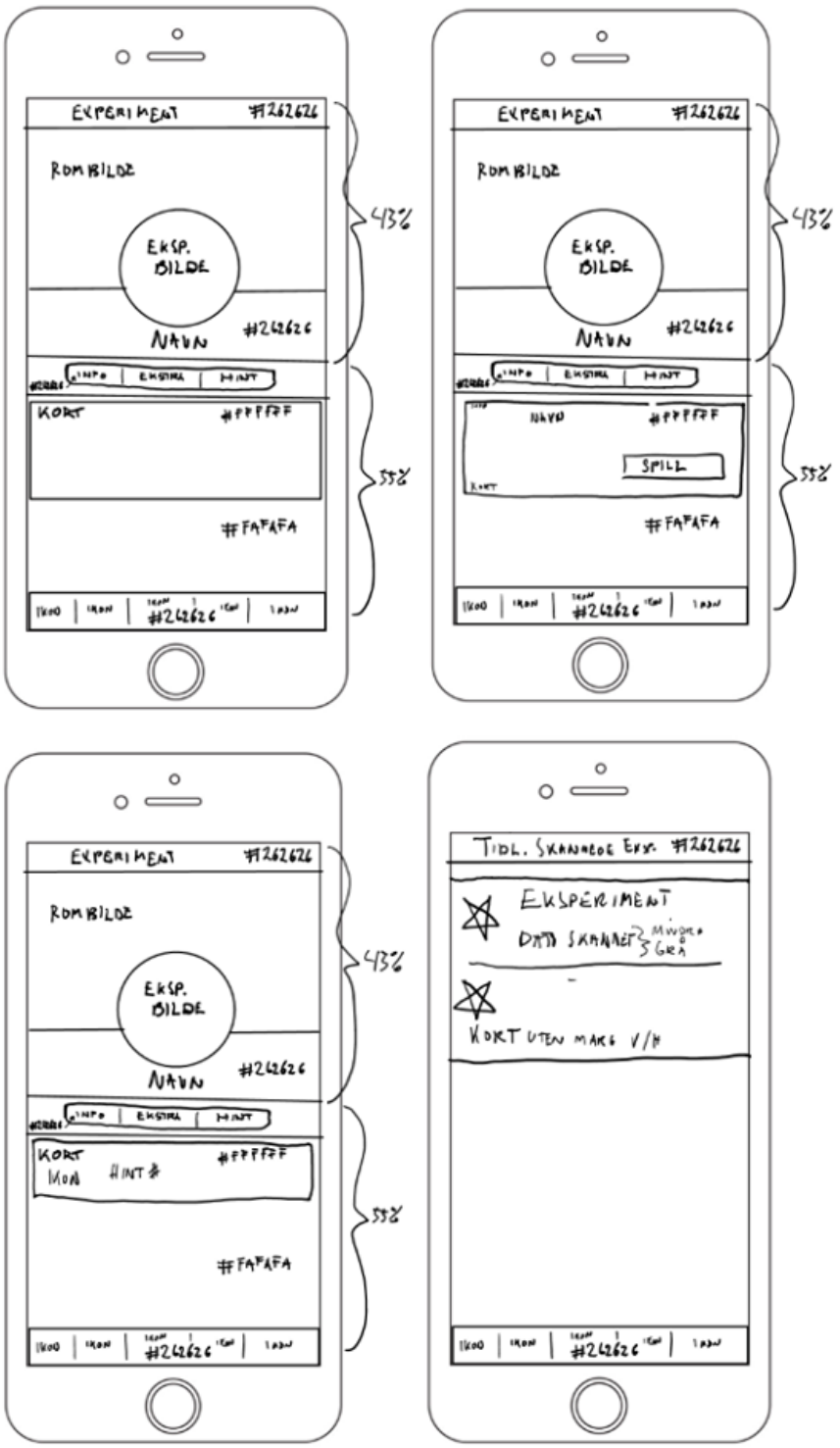
Figur 1: Mockups av eksperimentsidene på Android



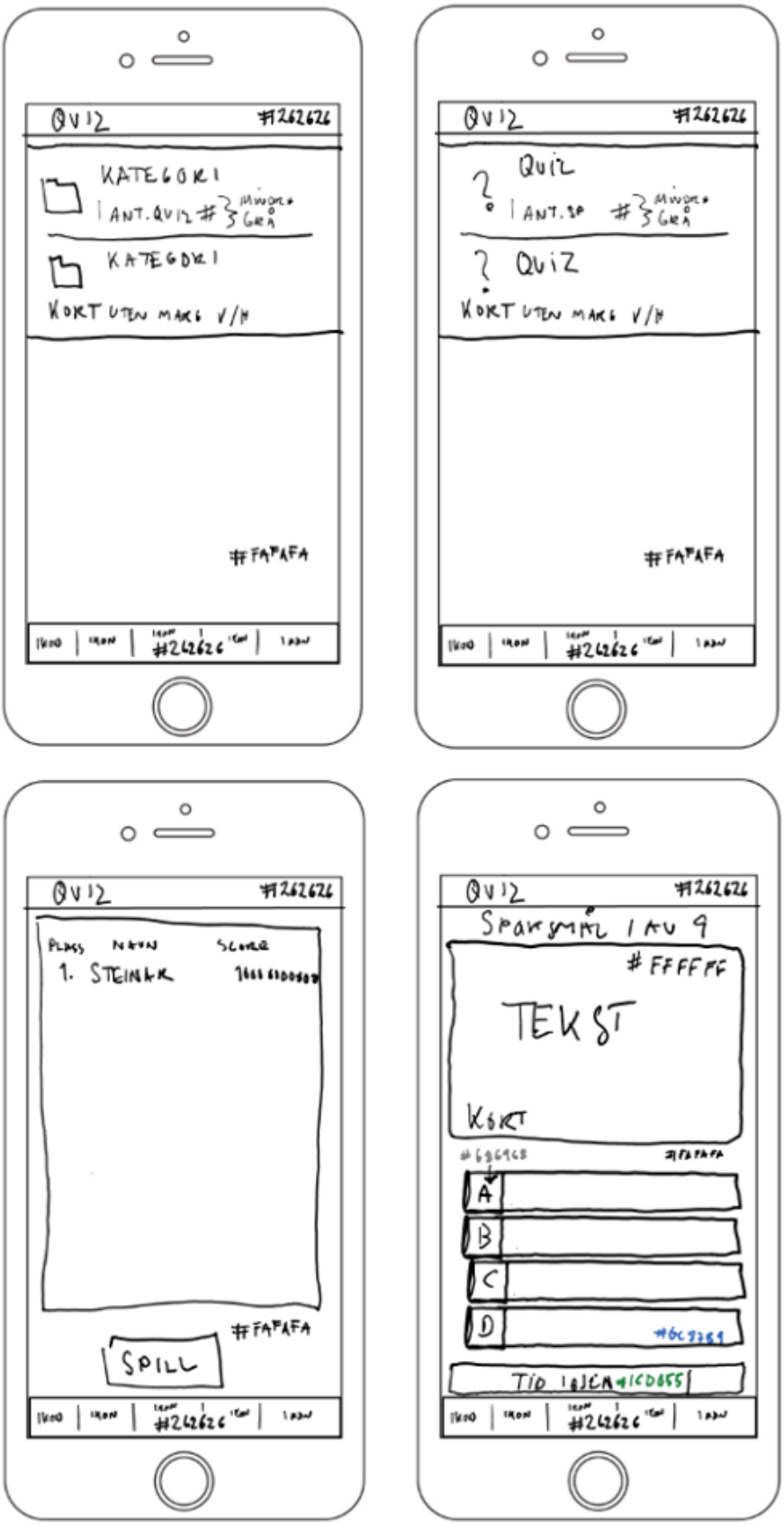
Figur 2: Mockups av quizsidene på Android



Figur 3: Mockups av informasjonssiden og quizresultat på Android



Figur 4: Mockups av eksperimentsidene på iOS



Figur 5: Mockups av quizsidene på iOS

Figur 6: Mockups av informasjonssiden og quizresultat på iOS

J Møtereferater

11.01.2017 - Lynkurs i prosjektstyring

Deltakere

- Tom Røise (foreleser)
- Alle gruppemedlemmer
- Andre bachelor-studenter

Mål

Få noen holdepunkter for å komme i gang med bacheloroppgaven på en strukturert måte.

Agenda

1. Særpreg ved Bacheloroppgavearbeidet
2. Kommentarer til noen punkter i Emnebeskrivelsen og Retningslinjer Ntnu Gjøvik
3. Prosjektplanen – gjennomgang av mal
4. Valg av systemutviklingsmodell / prosessrammeverk
5. Avsluttende råd og vink til prosjektstyring i Bacheloroppgaven

Oppgaver

Bli ferdig med prosjektplan til fristen, 28. januar.

12.01.2017 - Oppstartsmøte med Vitensenteret

Deltakere

- Fra Vitensenteret:
    - Petra Skoglund
    - Bodil Hansen
    - Hanne Røstøen
- Alle gruppemedlemmene

Agenda

- Prosjektleder oppdaterer oppdragsgiver om status og veien videre
- Avklare hvem som blir kontaktperson på Vitensenteret
- Kontrakt mellom oppdragsgiver, gruppemedlemmer og NTNU
- Avtale faste møtetidspunkter
- Klarere arbeidssted
- Fastsette oppgaven

Kontaktperson

Bodil Hansen blir Vitensenterets kontaktperson mot Scrum-teamet og fungerer som *product owner*.

Kontrakt

Arbeidsperioden er kontraktsfestet til tidsrommet 16. januar til 16. mai. Vitensenteret får eierskap til alt vi lager, mens NTNU får tilgang til å bruke både produktet og oppgaven i studiesammenheng.

Faste møter

- Det blir avholdt *sprint review* i slutten av hver to ukers-sprint, hvor *product owner* og andre fra Vitensenteret som ønsker det er til stede.
- Hver sprint går fra tirsdager til og med mandagen to uker senere.
- Utgangspunktet for de faste *Sprint review*-møtene blir annenhver mandag fra 14:30 til 15:30.
- Første *sprint review* blir 30. januar, 14:30.
- Andre *sprint review* blir 13. februar, 12:00.

Arbeidssted

- Møterommet i andre etasje på Vitensenteret er stort sett ledig, og står til vår disposisjon når ingen andre har booket det.
- Steffen har tilgang til bookingkalenderen, og sjekker den fortløpende.
- Å låne grupperom på biblioteket ble nevnt som et alternativ hvis det skulle bli kræsj.

12.01.17 - Internt planleggingsmøte

- Kryss-plattform eller *native*?
  - Fordeler og ulemper ved begge alternativene er grundig diskutert fra tidligere.
  - Vi fortsetter å utvikle *native*, siden vi har ønske om å benytte oss mer aktivt av telefonens sensorer i fremtidig funksjonalitet. Kryss-plattform kompliserer bruk av interne sensorer.
- Prioriterte oppgaver:
  - Sette opp fungerende infrastruktur
  - Tilpasse back-end til web-grensesnitt
  - Finne ut rammeverk som skal brukes til web-grensesnitt
  - Få iOs-app opp til samme standard som Android-app
- Ansvarsområder:
  - **iOs:** Ole André
  - **Android:** Steinar
  - **Back-end:** Steffen
  - **Web:** Steinar
- Rutiner:
  - Sørge for at all ny kode er godt dokumentert før *commit*
  - Bruke konsekvent *brancher* under utvikling:
    - Det skal lages en egen *branch* for hver Jira-*issue*.
    - **Developer:** All ny, FUNGERENDE kode inne i en sprint *merges* hit.
    - **Master:** Fungerende kode (fra *developer*) *merges* hit kun i slutten av hver sprint.

18.01.17 - Veiledning med Tom Røise

Deltakere

- Tom Røise (veileder)
- Alle gruppemedlemmene

Prosjektplan

- Få til flere prosjektspesifikke punkter i gantt-skjemaet. For eksempel frister for når enkelte vurderinger skal være avklart, slik som hvilke eksperimenter som skal inn.
- Avklare om vi burde ha *code reviews*, og eventuelt hvor detaljerte de skal være. All kode som skal merges til master bør kanskje være reviewet grundig?

Selve oppgaven

- Ta en grundig vurdering på brukertester og hvor detaljerte de skal være. Godt organiserte tester kan gi analyse verdifull både til applikasjonene og bacheloroppgaven.
- Vise bevisshet rundt valg tatt om backend og server. Vitensenteret skal kunne ta over prosjektet. Hvorfor Amazon? Hvorfor IaaS?
- Ha en diskusjon rundt problemer med forskjellige typer kontoer og aldersgrenser. Google-kontoer har til eksempel en grense på 13 år, mens mange av Vitensenterets besøkende er under denne grensen.
- Vi må sette opp en fornuftig sperre for banneord i brukernavn, som lett kan driftes av Vitensenteret.
- Vi må ta en grundigere diskusjon rundt valget mellom native og cross-platform. Gjøre research. Er det like vanskelig å bruke sensorer i cross-platform i dag som det var tidligere? Finne referanser.
- Gjøre et bevisst valg rundt bakoverkompatibilitet. Diskutere hvor gamle telefoner vi skal støtte. Bør unngå at enkelte brukere, for eksempel noen elever i en skoleklasse, blir ekskludert fordi de har for gamle telefoner. Begrunne valg av minimum API-level 17 på Android.
- Finne statistikk på hvilke mobil-OS som er i bruk i dag, samt hvilke versjoner av hvert enkelt OS.
- Finne ut hvilke rutiner det er for unit testing i iOS.
- Være bevisst på å vise utviklingen i prosjektet i bacheloroppgaven. Fortelle om vanskelige valg. Vise før og etter-screenshots av GUI. Fortelle om idéer om blir forkastet.
- Være bevisst på rekkefølgen i ting må utføres. Er det noe som er avhengig av noe annet? Backend kontra frontend? Få dette inn i gantt-skjemaet?
- Hvis vi må mocke statisk server for at arbeided på frontend kan begynne, må dette begrunnes i oppgaven.
- Ta stilling til informasjonssikkerhet så tidlig som mulig.
- Sjekke om vi kan hente inspirasjon og idéer fra andre prosjekter eller museum, som for eksempel Teknisk Museum.
- Redegjøre for hva vi tar med fra forprosjektet, og hva vi eventuelt forkaster.

Oppgaver

Frist	Ansvar	Oppgave
Til neste veiledning	Gruppen	Forberede en presentasjon av status. Hva er ferdig? Hva tar vi med oss fra forprosjektet? Hva forkastes?
Til neste veiledning	Tom Røise	Lese igjennom prosjektplanen og komme med tips til forbedringer.
Til neste veiledning	Tom Røise	Sjekke med Simon om developer-konto til iOS.
Til neste veiledning	Steinar Opphus	Sende epost til Hilde angående oppsett av webområde.

20.01.17 - Daily scrum

- Vi er litt foran tidsskjema og kan begynne å sette opp miljøer.
- Alle setter opp repo for sitt eget ansvarsområde:
    - Steffen:
        - viten-i-senter-backend
    - Ole André:
        - viten-i-senter-ios

    - Steinar:
        - viten-i-senter-frontend
        - viten-i-senter-android

- Alle gir hverandre admin-rettigheter til hvert repo.
- Ingen committer kode før vi har vært i "professional programming", i tilfelle det er en spesiell måte vi bør gjøre det på.
- Steffen begynner å sette opp Sonar-server.
- Begynne å gjøre research.
- Steffen og Steinar må diskutere hvordan kommunikasjonen mellom front og backend skal fungere.
- Bør ha begynt å kode i midten av neste uke.
- Første sprint planning blir førstkommende mandag. Da skal vi bestemme hva slags koding som skal på plass i løpet av sprinten.
- Tirsdag går bort i karrieredag.

23.01.17 - Sprint planning

Mål

- Definere oppgaver som skal være ferdig i løpet av denne sprinten.
- Definere arbeidsrutiner.
- Definere hvilke større oppgaver som må prioriteres på lang sikt.

Agenda

- Autentisering:
    - Tidkrevende, spesielt på backend.
    - Bruke JSON web tokens.
    - Det er viktig å få lagd en fungerende backend ganske fort, for å gjøre det enklere å lage de delene av front-end som skal kommunisere med den.
    - For å kunne få en fungerende backend, venter vi med autentisering til senere.
    - Frontend må kunne lagre token i 24 timer. Slettes etter 24 timers inaktivitet.
- Estimering:
    - Skal vi bruke story points eller estimere timer?
    - Timeestimering blir for rigid. Story points er mer smidig[124]. https://agilefaq.wordpress.com/2007/11/13/w is-a-story-point/
    - Vi har blitt enige om å bruke toerpotenser som basis for estimering, altså; 1, 2, 4, 8, 16 osv.
    - 2 tar **omtrent** dobbelt så lang tid som 1, 4 dobbelt så lang tid som 2 osv.
        - 1 - Triviell. Noen minutter.
        - 2 - Veldig liten. 15-30 minutter.
        - 4 - Liten. 1-2 timer.
        - 8 - Medium. 3-4 timer.
        - 16 - Stor. Hel dag.
        - 32 - Veldig stor. Over flere dager
    - Vi bruker planning poker for å estimere story points på hver oppgave. Verktøyet heter Pointing Poker[28].
    - Vi ble enige om å prøve å ha mer highlevel-oppgaver som issues, som vi kommer frem til i fellesskap, for deretter å legge til detaljerte oppgaver som sub-tasks.
- Backlog:
    - Vi bruker dagens møte for å definere mest mulig oppgaver til backlog, også langsiktige (se

tabell 1.

25.01.17 - Veiledning med Tom Røise

Deltakere

- Tom Røise (veileder)
- Alle gruppemedlemmene

Agenda

- Generelt:
  - Visning av demo ble utsatt til neste onsdag. Den gangen tar vi møtet på selve Vitensenteret, så Tom får et inntrykk av selve senteret og lettere kan forstå rollen appen vår kan ha.
  - Ole har vært i kontakt med Simon og Mariusz angående developerkonto til iOS.
  - Rapporten kan gjerne være på norsk. Det er ikke noe minus.
- Prosjektplan:
  - Effektmål:
    - Litt for mye teori i forhold til prosjektspesifikke mål.
    - Lite kvantifiserbare mål.
    - Mer om hva Vitensenteret ønsker.
  - Resultatmål:
    - Mer målbart.
    - Mer om drifting/videregutvikling etter prosjektets slutt.
  - Læringsmål:
    - Mer fordypning.
    - “Forstå viktigheten av testing” er for vagt.
  - Rammer:
    - Ting VI velger å gjøre, er ikke en ramme.
    - Vitensenteret definerer rammene.
  - Omfang:
    - Skille ut teknologi til et eget punkt.
    - Mer om verdenen vi skal inn i, om vitenskap.
    - Fortelle mer om eksperimentene. Hvordan er et typisk eksperiment.
    - Gjøre det klart at vi skal fokusere på å lage et rammeverk/grensesnitt for å vise informasjon om eksperimenter, ikke så mye på selve eksperimentene (dette gjør Vitensenteret i hovedsak selv etterpå).
  - Prosjektorganisering:
    - Utdyp mer produkteiers rolle i prosjektet. Skal kun hun våre med på sprint review meetings, eller skal flere være med?
  - Risiko:
    - Diskutere om hvordan vi skal unngå at våre spesialistroller blir et problem ved sykdom.
  - Gjennomføring:
    - Mer detlajer på sprint 1 i ganttskjemaet.
    - Avklare eksperimenter. Bør formuleres som en FRIST.
    - Ha brukertesting litt senere.

        - Glidende overgang mellom rapport og koding.
        - Flere avklaringsfrister. Avklare personvernproblematikk. Avklare iOS-opplæring.
        - Flere styringspunkter.
        - Mer innhold om brukertester. Hva tenker vi?
    - Verktøy:
        - Fortell mer om Jira.
        - Hvilke statuser.
        - Workflow.
    - Arbeidsrutiner:
        - Hvor ofte sprintmøter.
        - Hvilke typer møter.
        - Hvorfor? Mer argumentasjon.

30.01.17 - Sprint review

Deltakere

- Bodil Hansen (produkteier)
- Alle gruppemedlemmene

Agenda

- Gjennomgikk status etter sprint.
    - Prosjektplan er ferdig.
    - Scrum-master tok en rask gjennomgang av innholdet.
    - Produkteier har tidligere lest igjennom planen og vært behjelpelig med korrektur.
- Plan for videre utviklingløp (henviste til milepæler i gantt-skjema)
    - **Milepæl 1, 23. januar:** Avklare plattformer
        - Vi la frem argumentasjon for hvorfor vi mente det er best med native.
    - **Milepæl 2, 28. januar:** Deadline forprosjekt
        - Ferdig med forprosjekt, oppsett av utviklingsmiljø og servere.
    - **Milepæl 3, 30. januar:** Oppstart koding
        - Startskuddet for seriøs koding er nå.
    - **Milepæl 4, 28. februar:** Frist for avklaring av hvilke eksperimenter
        - Produkteier skulle snakke med resten av interessentene om eventuelle ønsker for eksperimenter de gjerne vil ha ekstra funksjonalitet på.
    - **Milepæl 5, 13. mars:** Checkpoint koding
        - Grunnkomponentene på plass i iOS. Dette vil si kommunikasjon med backend, QR-skanning og eskperimentside.
        - Webapplikasjonen fungerende, mangler bare finpuss. Det skal kunne legges til og endre eksperimenter, legge inn bilder og lage quizer.
        - Backend fungerer, og mangler bare finpuss. Alle komponenter støtter POST, PATCH og DELETE.
    - **Milepæl 6, 28. mars:** Brukertester ferdig
        - Være ferdig med brukertester og påfølgende analyse.
        - Produkteier sa at det er gode muligheter for å organisere brukerteter, både ved klassebe-

søk i ukedagene, og med publikum i helgene.

- **Milepæl 7, 11. april:** Koding ferdig og rapport godt underveis.
- **Milepæl 8, 16. mai:** Deadline rapport.

- Beacons kontra QR
  - Spørsmålet om beacons kom opp og vi diskuterte litt fordeler og ulemper.
  - Vi konkluderte at det er en ny og spennende løsning, men at det er for mange hindre som må overkommes for et så kort prosjekt.
    - Det første problemet vi ser er de områdene på Vitensenteret der eksperimentstasjonene er plassert veldig tett og i stort antall (Origo). Vi tror det vil skape problemer for treffsikkerheten.
    - Bodil hadde også det veldig gode poenget at statistikken for hvor mange besøkende pr. eksperimentstasjon ville bli mindre verdt. Det er jo mange steder man kan gå forbi en stasjon mange ganger i løpet av et besøk, uten egentlig å være interessert i selve stasjonen.

01.02.17 - Veiledning med Tom Røise på Vitensenteret

Deltakere

- Tom Røise (veileder)
- Alle gruppemedlemmene

Agenda

- Viste frem Android-prototypen.
  - Viktig med insentiv for å besøke vitensenteret flere ganger.
  - Achievements ble nevnt. “Samle” eksperimenter. Prøve å finne alle.
- Status:
  - God fremdrift.
  - Estimering. Viktig å ikke overestimere. Ikke legge inn oppgaver verdt mer storypoints enn det man klarte forrige sprint. Skal være realistiske mål.
- Problemstilling:
  - Vi luftet idéen om å fokusere på selve utviklingsprosessen og ha det som en problemstilling i rapporten.
  - Vi bør ikke fokusere for mye på det, men diskutere mye rundt det.
- Tidligere prosjektoppgaver:
  - Viktig å strekke oss lengre. Verden har gått videre.
- X-factor:
  - Kan være byggeverktøy.
- Omvisning på Vitensenteret til slutt.

14.02.17 - Sprint planning

- Ekstra funksjonalitet på eksperimenter legges i backlog. Dette er en funksjon de vanlige brukerne på Vitensenteret ikke trenger. Det er naturlig å vente til etter vi har lagd innloggingssystemet og kan opprette en superbruker.
- Vi gjør om skalaen på story points til den som er mest vanlig, basert på Fibonacci.
- Ole tar for seg tidligere skannede eksperimenter og quiz denne sprinten.

- Steinar og Steffen jobber også med quiz, i hver sin ende.
- Gamification ble nevnt. Achievementsystem ble skrinlagt allerede i forprosjektet, siden tanken var å basere det på Google Play, og de har aldersgrense - men det bør være en enkel sak å lage statistikk over hvor mange stasjoner en bruker har besøkt.
- Bannefilter må innføres i forbindelse med leaderboard.
- Vi bestemte oss for å separere språkene i quizene, slik at det er mulig å lage en quiz i kun ett språk. Man slipper derfor problemet med å alltid måtte lage hvert spørsmål på alle tilgjengelige språk for å unngå blanke spørsmål.
- Vi innfører kategorier på quiz, så det er mulig å velge tema og vanskelighetsgrad.
- Vi diskuterte mulige problemer med å sende med alle quiz-spørsmål inkludert riktig svar med samme JSON-objekt (det kan exploites), men fant ut at det inntil videre er mer jobb enn det er verdt å endre på.

15.02.17 - Veiledning med Tom Røise

Deltakere

- Tom Røise (veileder)
- Alle gruppemedlemmene

Agenda

- Det står 2016 flere steder i rapporten. Rette.
- Første inntrykk er at det er et uklart fokus for prosjektet. Før, under eller etter besøket?
- Bestemme seg og være konsekvent på om det skal skrives i fortid eller nåtid.
- Brukergruppen må presiseres. Ikke bare appbrukere, webbrukere.
- Få frem klarere hva som skal gjøres nå, hva som er utfordringen i forhold til forprosjektet.
- Vi har fornuftig referansebruk. Fortsette med det.
- “Systemet vil kunne”, “vi har ganske stor frihet”. Være mer presis. SKAL! VIL!
- Mer klarhet i hva som er gjort i forprosjektet og hva som skal gjøres nå.
- VIL egentlig all kode fra forprosjektet bli med videre? Refaktorering.
- Mer argumentasjon rundt valg av løsninger. Hvorfor .NET, Polymer....
- Hvorfor er det lurt å dele i frontend og backend? Ikke så mye om hvilkke miljøer vi skal bruke, siden det blir beskrevet i detalj senere.
- Lite om arbeidsform foreløpig.
- Veldig spinkel argumentasjon for scrum.
- Storypoints. Er det vi som har funnet opp standarden, eller har vi valgt å følge det som er en etablert praksis?
- Generelt mer bilder, illustrasjoner, figurer.
- Oversette usecase diagrammet.
- Hvorfor har vi valgt use case?
- Databasen skal ikke være ekstern aktør i use case-diagrammmet.
- Språkproblematikk. Skrive om i rapport...?
- Techincal memo rundt https. Opp til Vitensenteret om de skal implementere.

K Jira

K.1 Sprint forprosjekt

K.1.1 Backlog etter første sprint planning

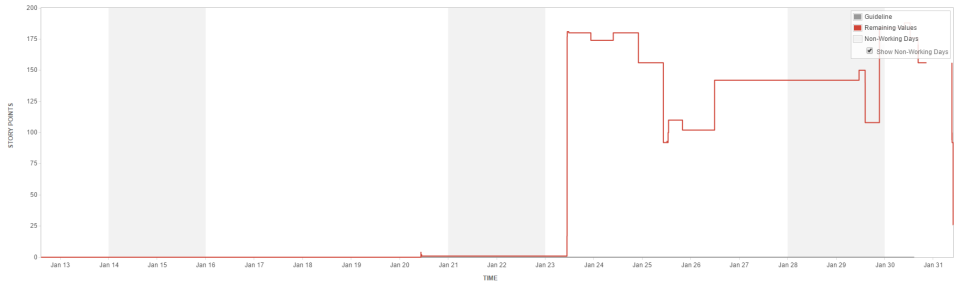
Tabell 1: Issues i backlog etter første sprint planning

Issue key	Summary	Assignee	Status
VISB-4	Rapport: Planlegging, oppfølging og rapportering	steffen.granberg	Done
VISB-5	Rapport: Organisering av kvalitetssikring	admin	Done
VISB-6	Rapport: Plan for gjennomføring	admin	Done
VISB-7	Rapport: Brukergrupper	steffen.granberg	Done
VISB-9	Rapport: Kvalitetskrav		Done
VISB-11	Android: Fork repository	admin	Done
VISB-12	Backend: Create fresh repository	steffen.granberg	Done
VISB-13	iOS: Create repository	ole.andre1	Done
VISB-14	Frontend: Create repository	admin	Done
VISB-15	iOS: Research	ole.andre1	In Progress
VISB-16	Polymer, Backend: Research Authentication	steffen.granberg	To Do
VISB-17	Backend: Research	steffen.granberg	In Progress
VISB-18	Rapport: Skrive om hvordan vi har definert storypoints.		To Do
VISB-19	Frontend: Define and set up the structure.	admin	In Progress
VISB-20	iOS: Set up class structure	ole.andre1	To Do
VISB-21	iOS: Implement QR-scanner	ole.andre1	To Do
VISB-22	iOS: Implement experiment page	ole.andre1	To Do
VISB-23	iOS: Setup GUI	ole.andre1	To Do
VISB-24	iOS: Implement Quiz	ole.andre1	To Do
VISB-25	iOS: Implement Math Game	ole.andre1	To Do
VISB-26	iOS: Show question		To Do
VISB-27	iOS: Answer Question		To Do
VISB-28	iOS: Get question from backend		To Do
VISB-29	iOS: Implement countdown timer		To Do
VISB-30	iOS: Show Score		To Do
VISB-31	iOS: Submit score		To Do
VISB-32	iOS: Show leaderboard		To Do
VISB-33	iOS: Get information from backend		To Do
VISB-34	iOS: Show name and description		To Do
VISB-35	iOS: Show hints		To Do
VISB-36	iOS: When scan go to experiment page		To Do
VISB-37	iOS: Implement information page	ole.andre1	To Do
VISB-38	iOS: Show opening hours		To Do
VISB-39	iOS: Show driving directions		To Do
VISB-40	iOS: Show upcomming events		To Do
VISB-41	iOS: Implement page with previous scanned experiments	ole.andre1	To Do
VISB-42	iOS: Leaderboard for Quiz	ole.andre1	To Do
VISB-51	Frontend: Login	admin	To Do
VISB-52	Frontend: Edit informationpage	admin	To Do
VISB-53	Frontend: User administration	admin	To Do
VISB-54	Delete user		To Do
VISB-55	Add user		To Do
VISB-56	Edit user		To Do
VISB-57	Frontend: Experiment administration	admin	To Do

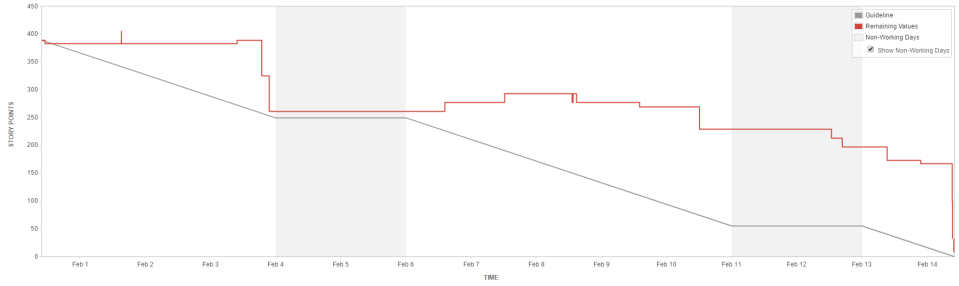
VISB-58	Frontend: Delete experiment		To Do
VISB-59	Frontend: Add new experiment		To Do
VISB-60	Frontend: Relocate exeperiment		To Do
VISB-61	Frontend: Edit name, descriptions		To Do
VISB-62	Frontend: Add new hint		To Do
VISB-63	iOS: Implement extra functions	ole.andre1	To Do
VISB-64	Frontend: Add extra function		To Do
VISB-65	Frontend: See statistics	admin	To Do
VISB-66	Frontend: Quiz administration	admin	To Do
VISB-67	Frontend: Make new quiz		To Do
VISB-68	Frontend: Edit quiz questions		To Do
VISB-69	Frontend: Add/remove question		To Do
VISB-70	Frontend: See leaderboard		To Do
VISB-71	Frontend: Delete entry in leaderboard		To Do
VISB-72	Frontend: Add information about center		To Do
VISB-73	Frontend: Edit openinghours		To Do
VISB-74	Frontend: Event administration	admin	To Do
VISB-75	Frontend: Make new event		To Do
VISB-76	Frontend: Delete event		To Do
VISB-77	Frontend: Edit event		To Do
VISB-78	Frontend: Delete hint		To Do
VISB-79	Frontend: Delete extra functions		To Do
VISB-80	Frontend: Add image		To Do
VISB-81	Frontend: Delete image		To Do
VISB-82	Frontend: Room administration	admin	To Do
VISB-83	Frontend: Add new room		To Do
VISB-84	Frontend: Delete room		To Do
VISB-85	Frontend: Edit room		To Do
VISB-86	Frontend: Upload image		To Do
VISB-87	Frontend: Delete image		To Do
VISB-88	Backend: Experiment endpoint	steffen.granberg	To Do
VISB-89	Backend: Authentication endpoint	steffen.granberg	To Do
VISB-90	Backend: Room endpoint	steffen.granberg	To Do
VISB-91	Backend: Event endpoint	steffen.granberg	To Do
VISB-92	Backend: Quiz enpoint	steffen.granberg	To Do
VISB-93	Backend: User endpoint	steffen.granberg	To Do
VISB-94	Backend: Name reservation	steffen.granberg	To Do
VISB-95	Backend: DELETE experiment		To Do
VISB-96	Backend: POST experiment		To Do
VISB-97	Backend: PATCH Experiment		To Do
VISB-98	Backend: Login and recieve token		To Do
VISB-99	Backend: Verify token		To Do
VISB-100	Backend: DELETE room		To Do
VISB-101	Backend: POST room		To Do
VISB-102	Backend: PATCH room		To Do
VISB-103	Backend: POST event		To Do
VISB-104	Backend: DELTE event		To Do
VISB-105	Backend: PATCH event		To Do
VISB-106	Backend: POST quiz		To Do
VISB-107	Backend: DELETE Quiz		To Do
VISB-108	Backend: PATCH quiz		To Do
VISB-109	Backend: GET quiz		To Do
VISB-110	Backend: GET Event		To Do
VISB-111	Get room		To Do
VISB-112	Backend: GET experiment		To Do
VISB-113	Backend: DELETE user		To Do
VISB-114	Backend: POST user		To Do
VISB-115	Backend: GET user		To Do

VISB-116	Backend: PATCH user		To Do
VISB-117	Backend: Setup automapper	steffen.granberg	To Do
VISB-118	Backend: Setup class structure, with interfaces.	steffen.granberg	In Progress
VISB-119	Android: Update communication with backend	admin	To Do
VISB-120	Rapport: Introduksjon		To Do
VISB-121	Rapport: Bakgrunn	steffen.granberg	To Do
VISB-122	Rapport: Prosjektbeskrivelse	steffen.granberg	To Do
VISB-123	Rapport: Rammer	ole.andre1	Ready for QA
VISB-124	Rapport: Brukergruppe	admin	Ready for QA
VISB-125	Rapport: Hvorfor valgte vi oppgaven	admin	To Do
VISB-126	Rapport: Forprosjekt	admin	To Do
VISB-127	Rapport: Gruppemedlemmer	ole.andre1	Ready for QA
VISB-128	Rapport: Roller	ole.andre1	Ready for QA
VISB-129	Rapport: Utviklingsprosess		To Do
VISB-130	Rapport: Utviklingsmetodikk		To Do
VISB-131	Rapport: Valg av metodikk		To Do
VISB-132	Rapport: Gjennomføring		To Do
VISB-133	Rapport: Besøkende		To Do
VISB-134	Rapport: Kravspesifikasjon		To Do
VISB-135	Rapport: Teknologier		To Do
VISB-136	Rapport: Polymer		To Do
VISB-137	Rapport: .NET Core		To Do
VISB-138	Rapport: Entity Framework		To Do
VISB-139	Rapport: Gradle		To Do
VISB-140	Rapport: Design		To Do
VISB-141	Rapport: Overordnet struktur		To Do
VISB-142	Rapport: Frontend web		To Do
VISB-143	Rapport: Backend API		To Do
VISB-144	Rapport: Android applikasjon		To Do
VISB-145	Rapport: iOS applikasjon		To Do
VISB-146	Rapport: Produksjonsetting		To Do
VISB-147	Rapport: Kvalitetssikring		To Do
VISB-148	Rapport: Testing		To Do
VISB-149	Rapport: Brukertester		To Do
VISB-150	Enhetstester		To Do
VISB-151	Rapport: Diskusjon		To Do
VISB-152	Rapport: Konklusjon		To Do
VISB-153	Rapport: Videre arbeid		To Do
VISB-154	General: Setup Sonar server		To Do
VISB-155	General: Stup backend server		To Do

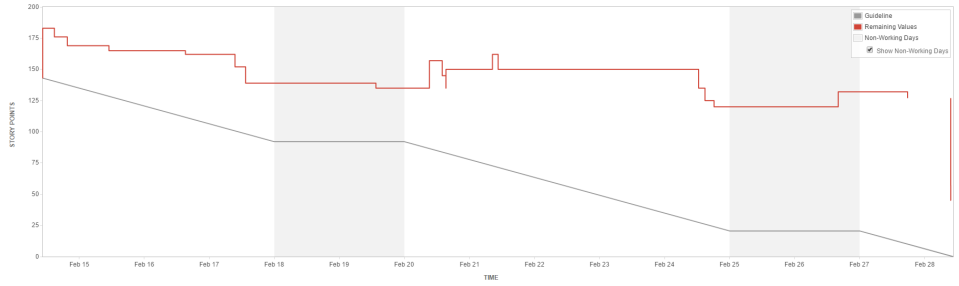
K.2 Grafer for hele prosjektet



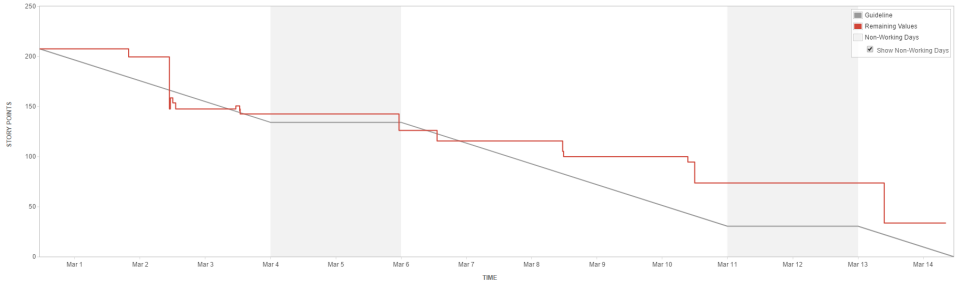
(a) Forprosjekt



(b) Sprint 1

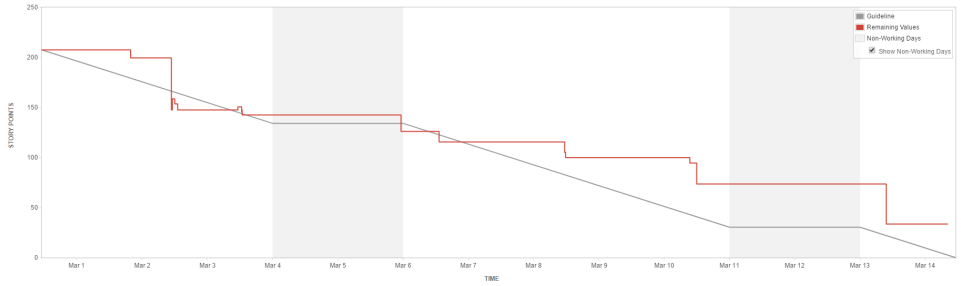


(c) Sprint 2

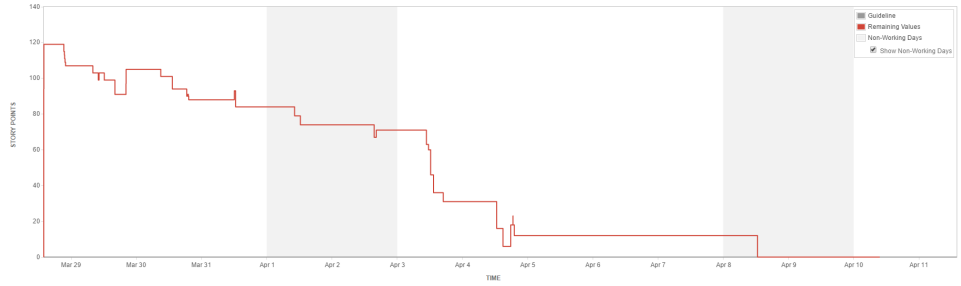


(d) Sprint 3

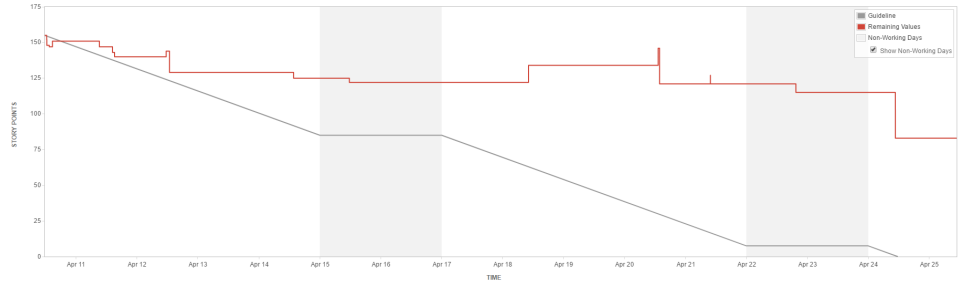
Figur 7: Burndown chart for forprosjekt og sprint 1 - 3



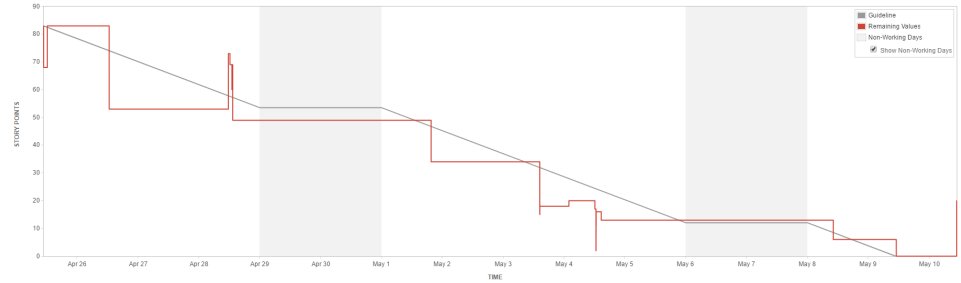
(a) Sprint 4



(b) Sprint 5

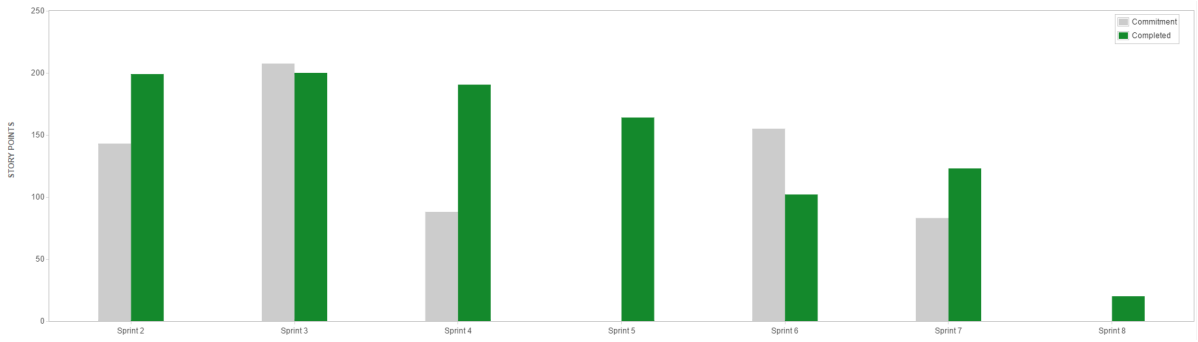


(c) Sprint 6

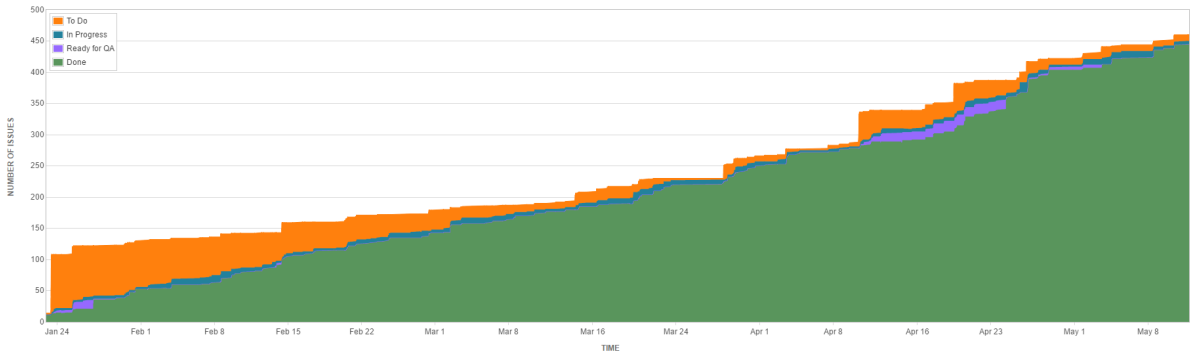


(d) Sprint 7

Figur 8: Burndown chart for sprint 4 - 7



Figur 9: Velocity chart



Figur 10: Cumulative Flow Diagram

L Timelogg

L.1 Ukeslogger

Tabell 2: Timelogg uke 2

User	2017-01-09	2017-01-10	2017-01-11	2017-01-12	2017-01-13	2017-01-14	2017-01-15	Total
Ole Slettum							01:39:38	01:39:38
Steinar Opphus			04:52:38	01:39:37	02:51:49			09:24:04
Steffen Granberg			02:00:00	01:30:00				03:30:00
			06:52:38	03:09:37	02:51:49		01:39:38	14:33:42

Tabell 3: Timelogg uke 3

User	2017-01-16	2017-01-17	2017-01-18	2017-01-19	2017-01-20	2017-01-21	2017-01-22	Total
Steinar Opphus	02:27:03	04:23:59	05:12:16		04:49:01	02:06:57		18:59:16
Steffen Granberg	02:55:00	04:15:56	03:43:53		06:07:00			17:01:49
Ole Slettum		05:20:59	05:42:59	02:17:09	05:39:06			19:00:13
	05:22:03	14:00:54	14:39:08	02:17:09	16:35:07	02:06:57		55:01:18

Tabell 4: Timelogg uke 4

User	2017-01-23	2017-01-24	2017-01-25	2017-01-26	2017-01-27	2017-01-28	2017-01-29	Total
Steinar Opphus	08:37:21	00:49:14	04:52:10	05:45:14	06:29:44		03:42:47	30:16:30
Ole Slettum	06:54:47	01:17:54	04:45:54	05:24:31	04:54:42		05:02:02	28:19:50
Steffen Granberg	09:12:56	02:08:00	09:15:22	07:16:55	07:08:04			35:01:17
	24:45:04	04:15:08	18:53:26	18:26:40	18:32:30		08:44:49	93:37:37

Tabell 5: Timelogg uke 5

User	2017-01-30	2017-01-31	2017-02-01	2017-02-02	2017-02-03	2017-02-04	2017-02-05	Total
Ole Slettum	09:13:51	07:13:01	05:51:25		06:19:49	01:26:48	01:43:14	31:48:08
Steinar Opphus	07:48:30	06:30:52	05:37:45	00:55:59	08:09:17	05:07:12		34:09:35
Steffen Granberg	08:53:11	03:31:55	05:10:50	03:00:00	11:21:55			31:57:51
	25:55:32	17:15:48	16:40:00	03:55:59	25:51:01	06:34:00	01:43:14	97:55:34

Tabell 6: Timelogg uke 6

User	2017-02-06	2017-02-07	2017-02-08	2017-02-09	2017-02-10	2017-02-11	2017-02-12	Total
Ole Slettum		08:05:27	09:01:46	07:04:55	04:31:53	00:58:07	01:53:35	31:35:43
Steinar Opphus	08:59:28	07:45:37	01:43:23	05:06:00	04:13:36	07:34:07	01:03:15	36:25:26
Steffen Granberg	10:46:01	06:49:13	06:18:00				01:22:00	25:15:14
	19:45:29	22:40:17	17:03:09	12:10:55	08:45:29	08:32:14	04:18:50	93:16:23

Tabell 7: Timelogg uke 7

User	2017-02-13	2017-02-14	2017-02-15	2017-02-16	2017-02-17	2017-02-18	2017-02-19	Total
Ole Slettum	05:09:32	06:20:20	07:38:06	05:25:41	09:13:11	04:59:46	01:30:01	40:16:37
Steffen Granberg	07:39:38	09:18:18	05:32:29		05:42:41		04:47:42	33:00:48
Steinar Opphus	08:36:56	10:54:23	03:57:28	08:02:28	04:49:34			36:20:49
	21:26:06	26:33:01	17:08:03	13:28:09	19:45:26	04:59:46	06:17:43	109:38:14

Tabell 8: Timelogg uke 8

User	2017-02-20	2017-02-21	2017-02-22	2017-02-23	2017-02-24	2017-02-25	2017-02-26	Total
Ole Slettum	05:31:06	07:48:26	04:34:26		06:13:53			24:07:51
Steffen Granberg	10:03:32	06:32:46	02:55:54		06:41:09	00:30:01	03:16:51	30:00:13
Steinar Opphus	02:25:29	03:07:02	00:52:10	06:39:42	08:37:54			21:42:17
	18:00:07	17:28:14	08:22:30	06:39:42	21:32:56	00:30:01	03:16:51	75:50:21

Tabell 9: Timelogg uke 9

User	2017-02-27	2017-02-28	2017-03-01	2017-03-02	2017-03-03	2017-03-04	2017-03-05	Total
Ole Slettum		09:23:53	07:29:27	03:00:00	05:49:51		02:46:26	28:29:37
Steinar Opphus	02:14:11	06:07:09	03:06:24	06:30:43	01:59:39	04:35:34	11:19:47	35:53:27
Steffen Granberg	Sykdom	09:03:56	08:55:47	03:37:33	06:28:42		01:57:29	30:03:27
	02:14:11	24:34:58	19:31:38	13:08:16	14:18:12	04:35:34	16:03:42	94:26:31

Tabell 10: Timelogg uke 10

User	2017-03-06	2017-03-07	2017-03-08	2017-03-09	2017-03-10	2017-03-11	2017-03-12	Total
Ole Slettum	07:36:11	04:06:01	04:03:53	01:26:38	06:46:43	03:05:32	03:54:11	30:59:09
Steinar Opphus	06:32:00	06:48:34	05:59:16	02:15:05	05:30:13	04:26:43	02:21:26	33:53:17
Steffen Granberg	04:47:31	05:36:09	07:22:22		06:21:45		06:57:41	31:05:28
	18:55:42	16:30:44	17:25:31	03:41:43	18:38:41	07:32:15	13:13:18	95:57:54

Tabell 11: Timelogg uke 11

User	2017-03-13	2017-03-14	2017-03-15	2017-03-16	2017-03-17	2017-03-18	2017-03-19	Total
Ole Slettum	08:27:51	08:29:09	07:20:36		05:47:27	01:20:30		31:25:33
Steffen Granberg	07:00:40	07:26:07	10:21:47	03:49:16	09:22:55	00:30:14	04:10:36	42:41:35
Steinar Opphus	05:57:08	07:37:08	05:19:54	05:37:50			04:16:47	28:48:47
	21:25:39	23:32:24	23:02:17	09:27:06	15:10:22	01:50:44	08:27:23	102:55:55

Tabell 12: Timelogg uke 12

User	2017-03-20	2017-03-21	2017-03-22	2017-03-23	2017-03-24	2017-03-25	2017-03-26	Total
Ole Slettum	04:15:06	06:06:47	06:14:48	04:45:12	07:41:37	00:58:04	02:42:55	32:44:29
Steinar Opphus	07:23:45	07:20:56	07:32:22	02:27:16	07:08:12	00:54:29		32:47:00
Steffen Granberg	07:41:23	06:53:12	06:57:25	02:23:33	07:24:11	03:00:41		34:20:25
	19:20:14	20:20:55	20:44:35	09:36:01	22:14:00	04:53:14	02:42:55	99:51:54

Tabell 13: Timelogg uke 13

User	2017-03-27	2017-03-28	2017-03-29	2017-03-30	2017-03-31	2017-04-01	2017-04-02	Total
Ole Slettum	05:06:53	03:18:42	06:05:42	02:38:54				17:10:11
Steinar Opphus		04:42:39	03:29:34	02:27:19	04:39:35	03:58:44	03:18:36	22:36:27
Steffen Granberg	01:05:48	07:30:09	06:22:03	04:30:53	06:21:10		02:33:35	28:23:38
	06:12:41	15:31:30	15:57:19	09:37:06	11:00:45	03:58:44	05:52:11	68:10:16

Tabell 14: Timelogg uke 14

User	2017-04-03	2017-04-04	2017-04-05	2017-04-06	2017-04-07	2017-04-08	2017-04-09	Total
Ole Slettum	01:02:14	Sykdom	Sykdom	Sykdom	03:26:54	Sykdom	01:49:00	06:18:08
Steinar Opphus	05:54:45	09:25:40	02:19:32	04:01:25	05:51:10	02:57:27	01:37:59	32:07:58
Steffen Granberg	07:15:03	07:04:06	05:32:39		04:45:43	03:18:00	03:20:45	31:16:16
	14:12:02	16:29:46	07:52:11	04:01:25	14:03:47	06:15:27	06:47:44	69:42:22

Tabell 15: Timelogg uke 15

User	2017-04-10	2017-04-11	2017-04-12	2017-04-13	2017-04-14	2017-04-15	2017-04-16	Total
Ole Slettum	05:59:13	04:59:45	05:18:28		00:52:09		00:23:25	17:33:00
Steinar Opphus	04:55:25	04:12:00	04:48:20	02:45:25	03:38:38	01:38:03	03:36:24	25:34:15
Steffen Granberg	10:04:05	07:02:49	06:55:04				04:11:55	28:13:53
	20:58:43	16:14:34	17:01:52	02:45:25	04:30:47	01:38:03	08:11:44	71:21:08

Tabell 16: Timelogg uke 16

User	2017-04-17	2017-04-18	2017-04-19	2017-04-20	2017-04-21	2017-04-22	2017-04-23	Total
Ole Slettum		Eksamen	Eksamen	07:44:30	05:53:59	04:40:31	01:55:51	20:14:51
Steinar Opphus	04:56:06	05:30:57	03:08:55	06:18:17	04:57:12		05:11:56	30:03:23
Steffen Granberg	03:22:20	06:29:42	06:45:59	08:09:24	05:53:17			30:40:42
	08:18:26	12:00:39	09:54:54	22:12:11	16:44:28	04:40:31	07:07:47	80:58:56

Tabell 17: Timelogg uke 17

User	2017-04-24	2017-04-25	2017-04-26	2017-04-27	2017-04-28	2017-04-29	2017-04-30	Total
Ole Slettum	03:45:54	05:23:40	05:52:38	03:29:48	05:51:27	01:40:58		26:04:25
Steffen Granberg	03:24:04	06:01:18	04:53:37	02:59:53	06:15:37			23:34:29
Steinar Opphus	05:02:07	04:24:40	04:40:05	01:37:00	05:27:28	04:17:22		25:28:42
	12:12:05	15:49:38	15:26:20	08:06:41	17:34:32	05:58:20		75:07:36

Tabell 18: Timelogg uke 18

User	2017-05-01	2017-05-02	2017-05-03	2017-05-04	2017-05-05	2017-05-06	2017-05-07	Total
Ole Slettum	00:56:06	06:05:22	04:48:27	05:00:41	03:15:26			20:06:02
Steinar Opphus	01:32:23	03:23:08	02:38:07	03:44:33	05:54:00		02:44:28	19:56:39
Steffen Granberg	01:44:47	06:01:18	05:00:26	05:18:22	05:21:13		01:00:00	24:26:06
	04:13:16	15:29:48	12:27:00	14:03:36	14:30:39		03:44:28	64:28:47

Tabell 19: Timelogg uke 19

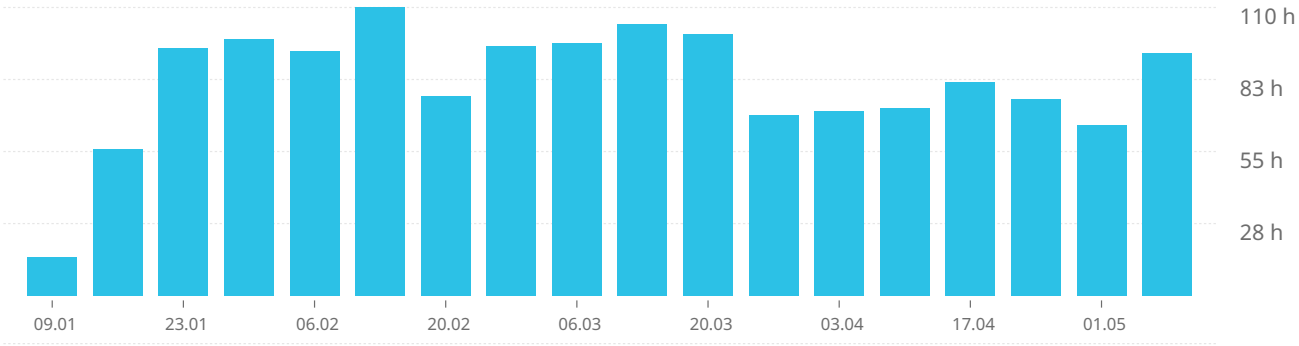
User	2017-05-08	2017-05-09	2017-05-10	2017-05-11	2017-05-12	2017-05-13	2017-05-14	Total
Ole Slettum	06:01:40	05:56:00	05:35:00	05:04:00	07:52:49			30:29:29
Steffen Granberg	03:51:58	07:04:13	06:05:00	04:40:00	08:43:07			30:24:18
Steinar Opphus	06:28:00	06:06:00	04:31:51	07:46:00	06:02:00			30:53:51
	16:21:38	19:06:13	16:11:51	17:30:00	22:37:56			91:47:38

L.2 Sammendrag av timeoppføringer i Toggl

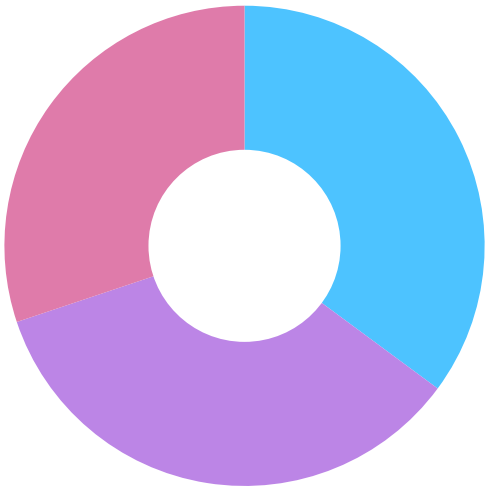
Summary report

2017-01-11 - 2017-05-12
Total 1454 h 42 min

Ole Slettum, Steffen Granberg, Steinar Opphus selected as users
IMT3912 - Bacheloroppgave selected as projects

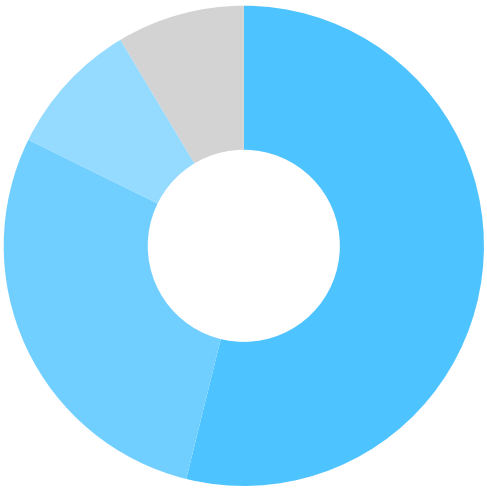


Users



- Steffen Granberg 510:57:29
- Steinar Opphus 505:21:43
- Ole Slettum 438:22:54

Time entries



- Koding 783:24:23
- Rapport 413:28:52
- Møte 132:56:03
- Other 124:52:48

Users / Time entries	Duration
Ole Slettum	**438:22:54**
Koding	240:30:10
Møte	41:57:44
Professional programming	18:21:10
Rapport	103:55:59
Refleksjonsnotat	0:43:13
Research	32:54:38
Steffen Granberg	**510:57:29**
Koding	267:17:05
Møte	45:57:01
Professional programming	31:47:50
Rapport	146:25:01
Research	19:30:32
Steinar Opphus	**505:21:43**
Koding	275:37:08
Møte	45:01:18
Professional programming	15:51:09
Rapport	163:07:52
Research	5:44:16