UNIVERSITETET I AGDER

Kandidater: Christian Moen, Erlend Thorvik, Erlend Wiklem, Sindre Grønstøl Haugeland, Martin Nenseth, Mathias Hartveit

Emnekode	IS-304
Emnenavn	Bachelorprosjekt i informasjonssystemer
Institutt	Institutt for Informasjonssystemer
Emneansvarlig	Hallgeir Nilsen
Veileder	Stig Nordheim
Oppdragsgiver	EVRY
Innleveringsfrist	16 Mai 2018
Antall ark inkl. Forside og vedlegg	122
Tittel på bacheloroppgave	ApiBot

Vi bekrefter at vi ikke siterer eller på annen måte bruker andres arbeider uten at dette er oppgitt, og at alle referanser er oppgitt i litteraturlisten.	Ja **X**	Nei

Kan besvarelsen brukes til undervisningsformål	Ja **X**	Nei

Vi bekrefter at alle i gruppa har bidratt til besvarelsen	Ja **X**	Nei

Bitfarmers

ApiBot

Vår 2018

IS-304 Bacheloroppgave i informasjonssystemer

Bachelorprosjekt - EVRY, Universitetet i Agder

Christian Moen, Erlend Thorvik, Erlend Wiklem, Sindre Grønstøl Haugeland, Martin Nenseth, Mathias Hartveit

Forord

Takk til Merethe Sjøberg(produkteier ved EVRY) for innspill, råd og evalueringer av prosjektet samt prosjektets behov underveis i utviklingen av systemet.

Takk til Espen Limi ved EVRY for kritiske og nødvendige innspill til tekniske og organisatoriske valg, samt generell veiledning underveis i prosjektet.

Takk til veileder Stig Nordheim for kontinuerlige tilbakemeldinger og råd til samtlige aspekter ved prosjektet underveis i prosjektet.

Takk til Hallgeir Nilsen og deltakere i IS-304 våren 2018 for diskusjon og refleksjon av vårt og andres prosjekt.

Takk til Daniel Hellang ved EVRY for verdifull input under utviklingen av systemet.

Målgruppen for rapporten er personer med bakgrunn innenfor IT og informasjonssystemer, og det er derfor anbefalt å ha noe forkunnskaper på dette området, da rapporten ikke vil utdype kjente begreper og teorier fra dette fagområdet.

Sammendrag

Formålet med dette prosjektet har vært å utforske og utvikle et system som forbedrer kundekontakt til servicetorg innen norske kommuner. Vi har fokusert på maskinlæring som vårt hovedverktøy for denne forbedringen, ettersom det er en fremtidsrettet og populær teknologi som vil se mye bruk fremover. Vi har også brukt prosjektet som en mulighet til å utvide vår egen kunnskap om prosjektgjennomføring innenfor IT og informasjonssystemer.

Vår valgte metode for dette prosjektet har vært en agil tilnærming, spesifikt Scrum. Prosjektleder og produkteier var godt kjent med rammeverket og hvordan dette skulle brukes, i tillegg til at vi følte det passet godt for prosjektet og gruppen. Verktøy som Visual Studio Team Services har gitt gode styringsmuligheter gjennom hele prosjektet. Vi etablerte tidlig interne regler og rutiner for å sikre høy kvalitet av produktet, noe som gjennom hele prosjektgjennomføringen har bidratt til å sikre at kvaliteten på prosessen og produktet var på det nivået vi og oppdragsgiver ønsket. Dette skjedde blant annet gjennom en forhåndsdefinert testplan og nøye bruk av kode kontroll.

Under gjennomføringen av prosjektet har det vært vesentlig å opprettholde en god og kontinuerlig kontakt med produkteier for å sikre at produktet som utvikles reflekterer kundens ønske. Derfor har det blitt holdt jevnlige møter med produkteier hvor prosjektets fremgang og status, samt fremvisning av implementert funksjonalitet, har vært typiske temaer.

I form av læringsutbytte har samtlige i gruppen fått en bredere forståelse for hvordan det er å jobbe i et team med bruk av Scrum, men også tilegnet seg god og relevant teknisk kunnskap innenfor maskinlæring og back/front-end programmeringsspråk og rammeverk. I tillegg har fire personer i gruppen fått fulltidsjobb under gjennomføring av prosjektet, innenfor relevante fagområder.

Resultatet av prosjektet er en chatbot applikasjon med høy kvalitet som implementerer dagsaktuelle teknologier til hensikt å effektivisere norske kommuner.

Innholdsfortegnelse

Figurliste

Akronymer

API - Application programming interface
BDD - Behaviour Driven Development
CI - Continuous Integration
FAT - Factory Acceptance Test
IoT - Internet of Things
JSON - JavaScript Object Notation
ML - Maskinlæringsrammeverk
MVC - Model View Controller
MVP - Minimum Viable Product
NLP - Natural language processing
OOAD - Objektorientert Analyse og design
RPA - Robotics Process Automation

REST - Representational State Transfer
SDK - Software development kit
TFS - Team Foundation Server

VSTS - Visual Studio Team Services

1. Introduksjon

Dette kapittelet introduserer prosjektet ved å beskrive hva vi har utviklet, hvem vi er, samt våre forutsetninger for å utvikle produktet, i tillegg til hvem vi har utviklet systemet for. Dette kapittelet vil også ta for seg hvilke overordnede mål vi har ved prosjektet.

1.1 Produktet

Systemet vi har utviklet heter ApiBot. ApiBot er en tekstbasert chatbot som kommuner i Norge kan bruke for å effektivisere tjenester som mottar og besvarer henvendelser fra innbyggere i kommunen. Applikasjonen vil automatisere arbeidsoppgaver arbeidere ved norske kommuner har i dag relatert til å håndtere henvendelser fra innbyggere. Fordi en stor andel av disse henvendelsene er like, kan ApiBot ved hjelp av maskinlæring bidra til å frigjøre ressurser norske kommuner bruker på kommunikasjon med innbyggere.

ApiBot bruker maskinlæring for å tolke spørsmål innsendt av sluttbrukere på en slik måte at spørsmålene kan kategoriseres. Kort forklart er maskinlæring en kontinuerlig automatisk forbedring av en prosess for å øke effektiviteten eller forbedre resultatet. *(Shavlik et al., 1990).* Deretter benytter ApiBot disse kategoriene for å finne relevant data fra kommunale datasett eller andre APIer som tilgjengeliggjør store mengder data. Applikasjonen kjører på en server og er tilgjengelig via API kall som tar inn spørsmål og svarer intelligent med informasjon relatert til spørsmål funnet i kommunens datasett. All kommunikasjon både til og fra chatboten skal fungere på norsk. I tillegg er ApiBot tilgjengelig for innbyggere i kommuner gjennom en utvidelse som kan kjøres på eksisterende kommunale nettsider.

1.2 Gruppemedlemmer og roller

Bitfarmers består av Mathias Hartveit, Sindre Haugeland, Christian Moen, Martin Nenseth, Erlend Thorvik og Erlend Wiklem. Alle gruppemedlemmene har tidligere samarbeidet med hverandre i andre emner på studiet i blant annet fag som er organisert som utviklingsprosjekter. Sammensatt besitter gruppen svært mye kunnskap, da mange har stor lidenskap og reell arbeidserfaring med “backend” og “frontend” utvikling. I tillegg har gruppemedlemmene valgt ulike valgfag på studiet, slik at gruppen har kompetanse på samtlige emner ved studieprogrammet bachelor IT og informasjonssystemer, med unntak av innovasjon og entreprenørskapsfag. At gruppen er trygge på hverandre og generelt har mye teknisk kompetanse har vært en forutsetning for å produsere et produkt av høy kvalitet på en effektiv måte.

Gruppemedlemmene har ikke blitt tildelt dedikerte roller, da det ikke har vært behov for dette fordi produktet vi har utviklet krevde grunnleggende serverside logikk før klienten kunne utvikles. Dette er med unntak av Martin som har vært scrum-master.

1.3 Oppdragsgiver

Prosjektet har blitt utført på oppdrag fra EVRY Norge avdeling Kristiansand. Våre kontaktpersoner har vært produkteier Merethe Sjøberg og teknisk samt organisatorisk veileder Espen Limi.

EVRY er norges største IT-selskap, med mer enn 8500 ansatte fordelt på ni land. EVRY leverer tjenester på områder som drift, konsulenttjenester og rådgivning *(EVRY, 2018).*

Det også verdt å nevne at Kristiansand kommune er en passende potensiell kunde av klientapplikasjonen. Det har gjennom prosjektet blitt holdt en dialog med Kristiansand Kommune for å blant annet avdekke behov og interesse for produktet vi har utviklet.

1.4 Mål ved prosjektet

Vi har valgt en oppdragsgiver og et prosjekt som krever høy kvalitet og god anvendelse av tekniske ferdigheter og metoder vi har lært gjennom bachelorgraden vår. Et av målene med prosjektet har vært å teste kunnskapen vi har opparbeidet oss de siste tre årene, ved å sette disse ferdighetene ut i livet i et virkelig prosjekt. I tillegg har det vært naturlig å oppsøke erfarne utviklere tilgjengelig for oss på EVRY for å bekrefte og avkrefte teorier og ideer vi har hatt, slik at vi har kunnet utvikle oss faglig.

Et annet sentralt mål ved prosjektet har vært å opparbeide kompetanse på områder som er aktuelle i markedet for å få en innholdsrik og relevant bacheloroppgave. Dette har innebåret å arbeide med teknologier og metoder som gir oss kunnskaper vi får bruk for i arbeidslivet.

1.5 Valg av produkt

Da oppdragsgiver ikke hadde et konkret oppdrag til oss, iverksatte vi en prosess med oppdragsgiver før semesterstart med å velge et passende prosjekt. Først og fremst ble vi enige om hvilke krav som måtte ligge til grunn for prosjektet, blant annet skulle det være relevant for markedet og utfordrende for oss. Det ble blant annet foreslått å utvikle noen “Proof of concept” produkter innenfor “Internet of Things” og “Robotics Process Automation”, men fordi vi ønsket å komme så nært et reelt og utfordrende prosjekt som mulig valgte vi ingen av disse.

Alle på gruppen har tidligere deltatt på Kristiansands lokale “Hackathon” hvor store mengder åpen data ble benyttet for å utvikle nye konsepter. Derfor ønsket vi å utforske hvilke mulighet disse dataene hadde. Ved å studere disse dataene og kommunisere med Servicetorvet i Kristiansand, fant vi et potensielt behov for å effektivisere de ovennevnte henvendelsene kommuner får fra innbyggere. Deretter skisserte vi omfanget av produktet, diskuterte det med oppdragsgiver og emneansvarlig i IS-304, og fikk aksept til å begynne.

1.5.1 Eksisterende løsninger

Det finnes allerede et par eksisterende løsninger innenfor samme problemdomene ApiBot forsøker å løse. Løsningene som viser høyest grad av modenhet er Kommune-Kari *(Kari, 2018)* og Kommuneforlagets chatbot *(Chatbot, 2018)*. Disse løsningene overlapper med vårt produkt på noen områder, men mangler funksjonalitet vi tilbyr på andre. For eksempel tilbyr ingen av løsningene direkte API-kall til chatbotten som muliggjør integrering av ApiBot i tredjepartsprogrammer, noe som vil si at ApiBot lett kan legges til i andre applikasjoner, og brukes på andre måter enn vi forutså under utviklingen av produktet.

2. Metode

Dette kapittelet beskriver og forklarer hvilke valg vi har gjort av utviklingsmetode og administrative verktøy, samt beskrive hva disse innebærer og eventuelle alternativer til disse.

2.1 Scrum

I utviklingen av ApiBot valgte vi å anvende metoden Scrum. Årsaken til at valget falt på Scrum var at samtlige i gruppen hadde tidligere erfaring med metoden, men også at vi så det kunne være fordelaktig med en smidig tilnærming under utviklingen av ApiBot. Dette var grunnet usikkerhet knyttet til valg av tekniske rammeverk og ønsket funksjonalitet, noe som gjorde at en smidig metode der muligheter for å håndtere raske endringer godt var høyst nødvendig, kontra en mer rigid fremgangsmåte som fossefallsmodellen.

Scrum oppfordrer til hyppig kontakt med kunde, i vårt tilfellet representert ved produkteier fra EVRY, som åpnet muligheten for hyppige og kontinuerlige tilbakemeldinger. At prosjektets fremgang kontinuerlig har blitt evaluert og vært mottakelig for endringsforslag fra produkteier har potensielt ført til at vi har unngått dyre

endringskostnader sent i prosjektet, som er en merkbar og ikke usannsynlig risiko ved for eksempel fossefallsmodellen *(Beck, 1999)*.

For å løfte fokuset fra tung, tidkrevende dokumentasjon av systemets brukerkrav, diskuterte vi verdier av systemets funksjonalitet for sluttbrukere gjennom brukerhistorier. Det er disse brukerhistoriene som danner systemets "product backlog".

Scrum har blitt fulgt på en så tradisjonell måte som mulig uten å gjøre noen vesentlige endringer i utførelsen, for å gjøre bruken av Scrum så autentisk som mulig. Derfor utnevnte vi tidlig i prosjektet en Scrum master, satt opp struktur og tidsrammer for sprinter, overholdt daily scrum møter og holdt nær kontakt med produkteier under utviklingen. "Planning poker" ble også tatt i bruk under hvert sprint-planleggingsmøte, hvor hver enkelt arbeidsoppgave ble gitt et grovt tidsestimat mellom 1 til 8 timer for å unngå for store og uoversiktlige arbeidsoppgaver.

Hver sprint varte to uker. Sprintenes lengde ble diskutert, men vi antok at kortere sprinter kunne føre til at tiden brukt på det administrative arbeidet (planlegging og gjennomgang) ville ta en stor del av tiden tilgjengelig i sprinten. På den andre siden kunne lengre sprinter gjøre det utfordrende å planlegge detaljerte ting langt fremover i sprinten.

2.2 Objektorientert analyse og design

Fremgangsmåten vår for analyse og design ble gjort gjennom å ta i bruk objektorientert analyse og design (OOAD). Dette for å bedre kommunikasjonen mellom produkteier og utviklingsteamet, slik at alle delte det samme bilde av hvordan systemet skulle bli utformet. Vi gjorde dette ved å utforme en rekke forskjellige dynamiske og statiske modeller, som kontinuerlig ble diskutert med produkteier og revurdert eller forbedret. Klassediagrammet fungerte som vår viktigste statiske modell, og ga samtlige gruppemedlemmer den nødvendige forståelsen for hvordan ApiBot skulle bygges opp *(Vedlegg 5 - Klassediagram backend)*. De dynamiske modellene som sekvensdiagram, "flow-chart", hendelsestabell og lignende, ga utviklingsteamet nytten av å se hvordan gangen i systemet skulle være.

2.3 VSTS - Metodeverktøy

Visual Studio Team Services (VSTS) har blitt benyttet for å organisere prosjektets "product backlog" i et "scrum-board", som hele Scrum-teamet og produkteier har hatt tilgang til. Enhver "backlog-gjenstand" inneholder konkrete arbeidsoppgaver deltakere av scrum-teamet enten kan tildele seg selv eller få tildelt. Disse arbeidsoppgavene inneholder tittel og relevant metadata, samt estimert forbruk i timer. Ved fullført sprint

planlegging legges nye arbeidsoppgaver til som backlog elementer med estimert antall timer, som VSTS automatisk produserer et burndown-chart for de oppgitte estimatene. I tillegg har vi benyttet VSTS til å rapportere og loggføre bugs. Dette verktøyet har tillatt oss å beskrive i detalj hvordan man gjenskaper problemet samt akseptansekriterier som beskriver hvilke kriterier som må være oppfylt for å suksessfullt løse feilen *(Visual Studio Team Services, 2018).*

VSTS tilbyr levering av kodebase (repository) med Git eller TFS som versjonskontroll. Dersom man benytter seg av kodebase og et metodeverktøy fra VSTS har man anledning til å integrere disse, hvor man da for eksempel har mulighet til å lage nye versjonskontrollgrener (branches) basert på en arbeidsoppgave fra "scrum-boardet". Ved opprettelse av "pull requests" på en slik gren vil de aktuelle arbeidsoppgavene grenen tar utgangspunkt i bli listet opp, og det er enkelt for de(n) som evaluerer "pull requesten" å se hvilke oppgaver de foreslåtte endringene er relatert til.

En annen funksjonalitet VSTS tilbyr som påvirket valget av metodeverktøy, var muligheten for å enkelt integrere automatiserte tester. Disse automatiske testene er tilgjengelig gjennom egendefinerte testplaner, hvor man setter opp testmiljø for et eller flere programmeringsspråk. Gjennom VSTS har vi satt opp bestemte hendelser som iverksetter de respektive testene, noe som for oss har gjort kontinuerlig integrasjon til en trygg prosess *(Vedlegg 9 - Kvalitetsdokument, Kontinuerlig integrasjon).*

2.4 Google drive - administrasjonsverktøy

Google Drive ble brukt til å administrere dokumenter og diagrammer. I tillegg har Google Docs mulighet til at flere kan jobbe på samme dokument samtidig. Google Drive ble valgt hovedsakelig fordi samarbeidsverktøyene er gode, da flere personer kan jobbe samlet om èt dokument og at den frie diskplassen er tilstrekkelig nok til å fylle prosjektet vårt *(Google Drive, 2018)*.

2.5 Slack

Slack er en skybasert "chat-klient" som har til hensikt å gi bedre kommunikasjonsflyt innad i et team, ved hjelp av forskjellige samarbeidsfunksjoner *(Slack, 2018)*. Dette er alt fra chat kanaler, til vedvarende meldinger i disse, samt funksjonalitet som gjør at man kan søke etter alt innhold i de forskjellige kanalene. Vi satt opp såkalte "web-hooks" som lytter til endringer i VSTS, som for eksempel nye versjonskontrollgrener, "commits", "pull requests", "backlog" elementer osv. Denne funksjonaliteten gjør det enklere for samtlige utviklere i gruppen å holde en oversikt over status og fremgang i sprintene.

Bruken av Slack gjorde det enkelt for oss å ha et felles område hvor vi kunne kommunisere og dele arbeid strukturert, med den ekstra fordelen at tidligere arbeid og informasjon kunne lokaliseres raskt dersom nødvendig.

3. Analyse

For å starte produksjonen av et prosjekt er det vesentlig for alle involverte å ha forståelse for hvilke utfordringer prosjektet skal forsøke å løse. Derfor så vi det som nødvendig å avdekke hvilke brukerkrav som fantes i problemdomenet, før vi startet noen form for implementasjon. I denne prosessen opparbeidet vi oss nok kunnskap rundt applikasjonens problemdomene, på en slik måte at hele gruppen fikk en felles forståelse for hvilke utfordringer systemet hadde allerede i begynnelsen av prosjektet.

3.1 Informasjonsinnhenting og intervju

For å avdekke brukerkrav gjennomførte vi et intervju med lederen for Kristiansand kommunes Servicetorg *(Vedlegg 2 - Intervju Enhetsleder Servicetorvet)*. Hensikten med intervjuet var å avdekke hvordan kommunen håndterer forespørsler fra innbyggere, hva slags henvendelser de typisk får, og hvor mye ressurser de bruker på disse henvendelsene. Ved å kartlegge dette økte vi vår forståelse for problemdomenet, og nærmere bestemte hvordan Kristiansand kommune løser dette i dag. I tillegg etablerte vi i dette intervjuet kontakt med en potensielt viktig samarbeidspartner vi kunne bruke gjennom prosjektet for å hente ut mer informasjon.

3.2 Ikke-funksjonelle krav

Som et bidrag til både felles forståelse mellom utviklere internt og produkteier, og for ytterligere tydeliggjøring av prosjektets krav, utformet vi ikke-funksjonelle krav for applikasjonen. Samtlige ikke-funksjonelle krav beskrives og forklares ved følgende attributter: Ikke-funksjonelt krav, forklaring, krav og begrunnelse for kravet (*Vedlegg 4 - ikke-funksjonelle krav)*.

3.3 Brukerroller

Med utgangspunkt i våre og produkteiers antakelser farget av informasjonen vi hentet ut fra det ovennevnte intervjuet, så vi at systemet hadde behov for å betjene tre brukergrupper.

- Innbygger: Innbyggere vil være de mest typiske brukerne av sluttproduktet, derfor vil systemets brukerkrav legge stor vekt på nettopp denne brukergruppen.
- API-bruker: API-brukere er tredjeparts utviklere som tilgjengeliggjør chatbottens logikk ved å utføre API-kall. Denne brukergruppen vil derfor ha anledning til å implementere egne applikasjoner hvor de benytter ApiBots funksjonaliteter.
- Maskinlæringsansvarlig: Maskinlæringsansvarlige er administrative brukere som har ansvar for å trene maskinlæringen ApiBot benytter. Denne gruppen skal i hovedsak sjekke og verifisere spørringer gjort mot APIet, med hensikt å forbedre treffsikkerheten på de forskjellige kategoriene.

3.4 Brukerhistorier

For å opprettholde et brukerorientert fokus under utviklingen utformet vi brukerhistorier som inneholder en handling utført av en bestemt brukergruppe med ønske om å oppnå et resultat. For å sørge for at kunden får et produkt den virkelig ønsker og behøver ble samtlige brukerhistorier utformet i samarbeid med produkteier. I praksis har dette utspilt seg ved at vi som utviklingsteam har laget forslag til brukerhistorier med prioriteringer, akseptansekriterier og akseptansetester, og fått tilbakemelding fra produkteier. Dersom produkteier har hatt ønske om endring har vi diskutert disse innspillene internt i Scrum-teamet og deretter foreslått mulige løsninger til produkteier *(Vedlegg 3 - Brukerhistorier)*.

3.4.1 Prioritering: MoSCoW

For å prioritere brukerhistorier benyttet vi MoSCoW. Hensikten med å benytte MoSCoW er å nå frem til en felles forståelse mellom Scrum-teamet og produkteier om hvilke funksjonaliteter som er absolutt nødvendig og mindre nødvendig i systemet. Ved å prioritere brukerhistorier på denne måten tydeliggjør scrum-teamet hvilke funksjonaliteter som er viktigst, og derfor hvilke som vil bli implementert først *(MoSCoW method, 2018)*.

Must have

Krav prioritert som must have er vital funksjonalitet som kreves for å få et fungerende system. I dette prosjektet vil brukerhistorier som omhandler ApiBots evne til å kommunisere enkelt, mulighet for maskinlæring, og tilgjengeligheten av APIene være helt kritisk for å levere et fungerende system.

Should have
Krav prioritert som “should have” er viktig funksjonalitet, men ikke vital for å levere et fungerende system. I dette prosjektet er brukerhistorier med denne prioriteten typisk avansert funksjonalitet rundt maskinlæring og svar fra ApiBot.

Could have
Krav prioritert som “could have” er funksjonalitet som er ønsket fordi det potensielt kan øke brukeres tilfredshet, men som ikke er vesentlig for å implementere et fungerende system. Disse brukerhistoriene vil bli implementert om vi har nok ressurser tilgjengelig.

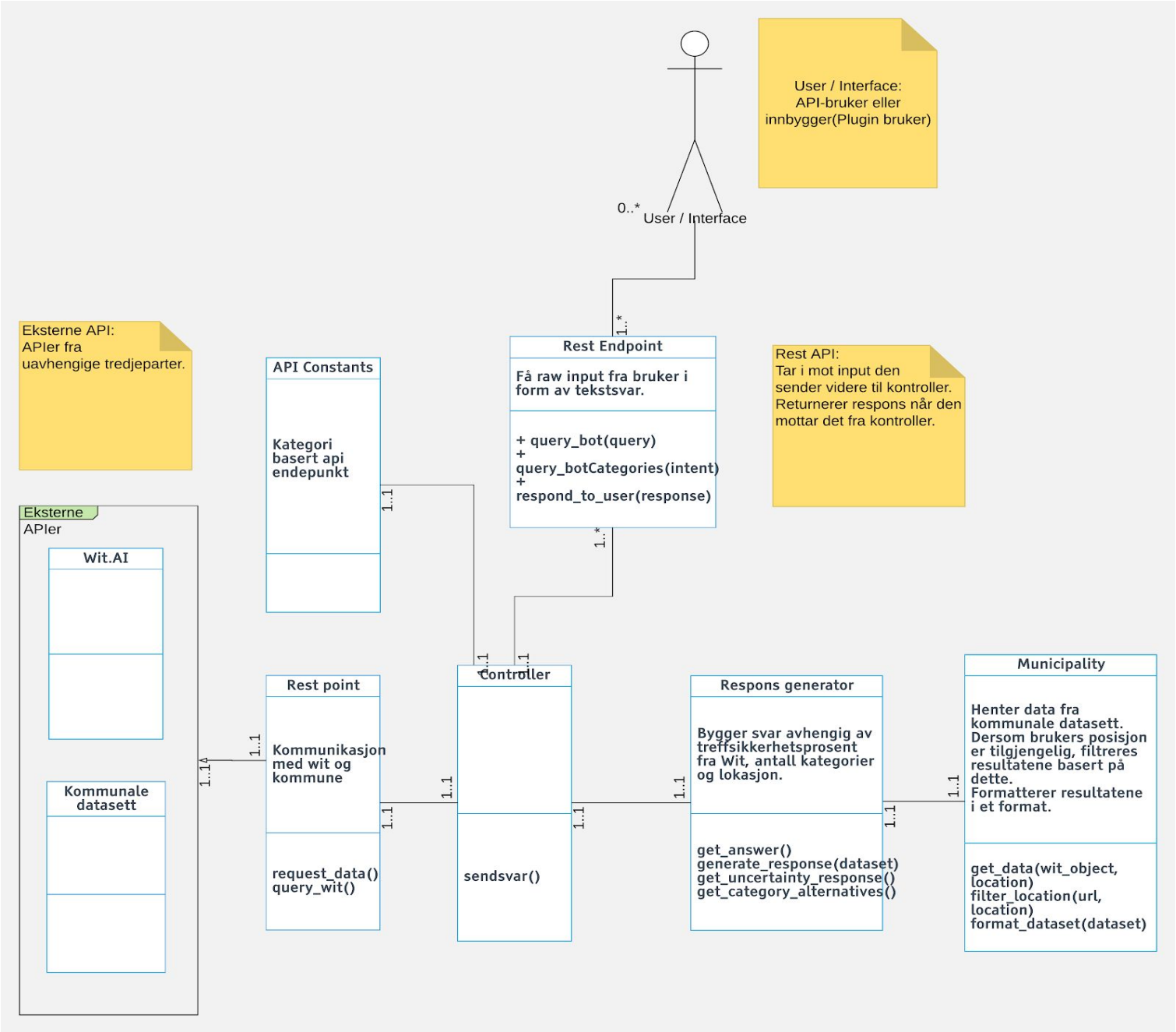
Won't have
Krav prioritert som “won’t have” er funksjonalitet som av ulike årsaker ikke vil bli implementert i systemet. Brukerhistorie #15 er prioritert som “won’t have” fordi vi ønsker å tydeliggjøre hva omfanget av prosjektet *ikke* er.

4. Design

Første utkast til designet av applikasjonen ble gjennomført etter at brukerkrav var avdekket. Siden den mest fundamentale delen av applikasjonen er chatbottens logikk, fokuserte vi i denne innledende designfasen på å utforme modeller som beskriver og forklarer dette. Målet med dette var å lage grove skisser for *hvordan* systemets ulike komponenter oppfører seg isolert og i samhandling med hverandre. Enkelte av disse designmodellene bidrar også til å tydelig foreslå hvilke data, datatyper og metoder de ulike komponentene er ansvarlig for. Denne informasjonen som både beskriver og forklarer modulers oppførsel fra ulike perspektiv kan bidra til et mer gjennomtenkt system- og moduldesign som alle utviklere er inneforstått og komfortable med.

4.1 Klassediagram

Klassediagrammet *(Figur 1)* illustrerer de ulike klassene i “backend” delen av systemet, samt hvordan de er relatert til hverandre. Klassediagrammet er en av de mest sentrale modellene i prosjektet, da det ikke bare er den første utarbeidede modellen som beskriver og forklarer applikasjon, men også i stor grad påvirker systemarkitekturen.



Figur 1 - Klassediagram (Vedlegg 5 - Full størrelse klassediagram backend)

User / Interface er ikke modellert som en klasse da dette representerer de faktiske brukernes interaksjon med systemet. Brukere, uansett type, initierer kontakt med ApiBot ved å gjøre et eller flere kall til “REST endpoint”.
REST endpoint klassen håndterer kontakt med et vilkårlig antall brukere, og består av tre ulike funksjoner:

- respondToUser som tar et svar generert av andre klasser i systemet som parameter, og returnerer dette til brukeren.
- queryBot som tar et spørsmål som parameter, og sender det videre i systemet.
- queryBotCategories som tar en “intent” som parameter. Denne funksjonen vil typisk bli kalt dersom bruker stiller et spørsmål med flere kategorier, respondToUser kalles med en liste over aktuelle kategorier som parameter, og bruker sender spørring til dette endepunktet med en av disse kategoriene som parameter.

Controller kan ses på som en klasse som integrerer og initierer kall til resten av systemet. Ved kall til en spørring i REST endpoint iverksettes prosessene i Controller avhengig av hvilken type spørring som kalles i REST endpoint. Dersom queryBot kalles i REST endpoint vil controller gjøre kall til maskinlæringsrammeverket (ML) gjennom REST point med spørsmålstrengen som parameter. ML vil returnere et objekt med blant annet treffsikkerhetsprosent og antatt(e) kategorier.

Deretter vil Controller kalle Respons Generator med objektet fra ML som parameter. Det finnes tre ulike scenarioer for hvilke kall som utføres i Response Generator, avhengig av treffsikkerheten mottatt fra ML.

- Hvis treffsikkerhetsprosent fra ML er over 85% på en kategori vil Response Generator bygge svar ved å hente og eventuelt filtrere basert på brukers lokasjon, og formatere data fra Municipality klassen. I så fall returneres et objekt med denne dataen, som Controller sender videre til Rest Points respond_to_user.
- Hvis treffsikkerhetsprosent er over 85% på flere kategorier vil Respons Generator kun returnere disse kategoriene til Controller. I såfall vil Controller kalle Rest Endpoints respond_to_user funksjon med kategoriene og oppfordring til bruk av query_bot_categories funksjonen til bruker.
- Hvis treffsikkerheten er under 70% vil Respons Generator returnere en streng til Controller som forklarer at den er usikker på spørsmålet fra brukeren. Også denne returverdien sendes til Rest Endpoints respond_to_user.

4.2 Funksjonsliste

Da produktet består av to systemer, chatbottens logikk og en “front-end” “plugin”, ble funksjonslisten vi utformet todelt. Funksjonslisten beskriver og forklarer de mest sentrale funksjonalitetene vi forutså på dette tidspunktet i applikasjonen. Disse funksjonalitetene er blitt beskrevet og forklart ved fire attributter; funksjonsnavn, kompleksitet, type og begrunnelse for funksjonaliteten.

Funksjonslisten har blitt brukt som et effektivt bidrag til å estimere tid i ulike sprint planlegginger da den ikke bare beskriver selve funksjonaliteten, men også kompleksiteten. I tillegg har funksjonslisten på grunn av sin omfattende deskriptive natur blitt brukt som et verktøy for konkretisering av arbeidsoppgaver se *(Vedlegg 6 - Funksjonsliste)*.

4.3 Øvrige modeller

For å ytterligere designe *hvordan* systemet skulle fungere valgte vi å utarbeide fire modeller til, da vi anså verdien av flere og mer detaljerte perspektiver av applikasjons-domenet som potensielt stort. I tillegg til verdien av å utfordre klassediagrammet og funksjonslisten så vi at utformingen av flere designmodeller kunne bidra til enda mer diskusjon og refleksjon i gruppen. Derfor utarbeidet vi funksjonelle krav, en hendelsestabell, et sekvensdiagram, og et "flowchart" se *(Vedlegg 7, 8 - Funksjonelle krav, Hendelsestabell)*.

Som et av flere resultater av disse nye modellene opparbeidet vi oss ny informasjon om hvordan systemet ikke bare kan, men *burde* fungere. Dette førte blant annet til at vi valgte å gjøre vesentlige endringer i klassediagrammet for å representere de nye funnene. Eksempelvis brakte utformingen av hendelsestabellen informasjon om at en tidligere antatt relasjon i klassediagram ikke var korrekt.

4.4 Prototype

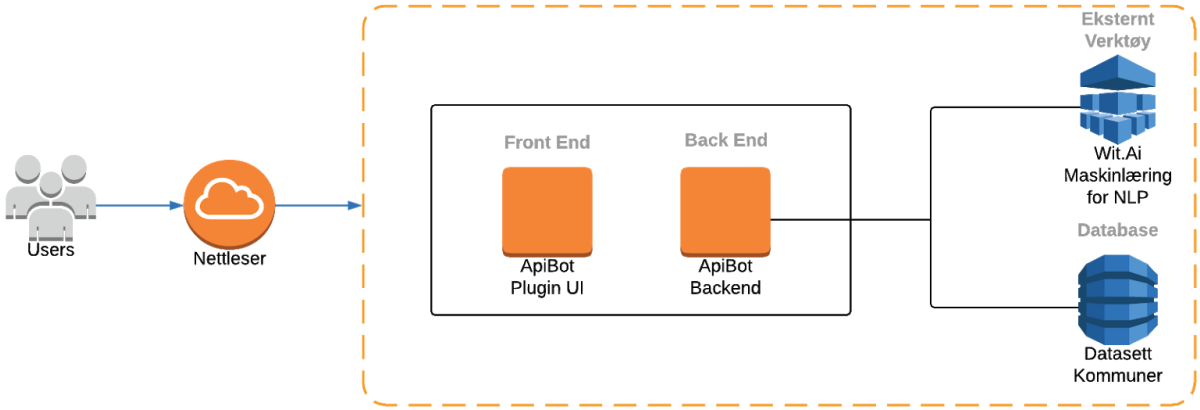
Den første prototypen av ApiBot ble implementert i Sprint 3 for å kunne presentere progresjon i prosjektet til produkteier. Brukbarheten av prototypen ble ikke prioritert da vi heller ønsket å vise implementerte funksjonaliteter, som for eksempel at ApiBot på tidspunktet klarte å sende og motta data gjennom APIet vi hadde implementert.

Figur 2 - Prototype ApiBot

Figur 2 illustrerer hvordan kommunikasjon med ApiBot chatten fungerer. På venstre side stiller brukeren spørsmålet “Hvor er parkering?”, hvor bruker da får tilbake en liste over parkeringsplasser fra ApiBot. På høyre side sender brukeren inn en setning Apibot ikke forstår, og ApiBot svarer derfor med en passende feilmelding.

4.5 Systemarkitektur

ApiBot består av flere overordnede deler, brukere kommuniserer i dag med applikasjonen gjennom sin nettleser, hvor ApiBot “Plugin UI” brukes. Dette regnes som “front end” delen av applikasjonen *(Figur 3)*. Hovedparten av chatbottens logikk utføres i “backend” delen av systemet. Der tas alle forespørsler imot og blir behandlet for forståelse og relevans. Backend har også koblinger mot Wit.Ai (ML) og Kommuners datasett (databaser) for å kunne kategorisere spørsmål og hente relevant data for disse.



Figur 3 - Systemarkitektur diagram

Figur 3 illustrerer ApiBots systemarkitektur. Brukere av typen innbygger initierer kontakt med ApiBots "frontend" chat, som ved spørsmål mottatt fra innbyggere gjør kall til serversiden i systemet. Serversiden gjør spørring til ML og mottar en kategori ML antar relaterer seg til innbyggernes spørsmål. Denne kategorien brukes til å hente ut data fra kommunale datasett. Deretter genereres et passende svar som inkluderer den kommunale dataen på serversiden, før det så sendes tilbake til chatten hvor det er tilgjengelig for sluttbrukeren.

5. Teknologivalg

Vi skal i dette kapittelet redegjøre og drøfte de mest sentrale teknologivalgene vi har måttet ta gjennom prosjektet.

5.1 Maskinlæringsrammeverk

Vi anerkjente tidlig i prosjektet at å lage vårt eget maskinlæringsrammeverk fra bunn ikke var aktuelt da det krever relativt høy forståelse for matematikk, i tillegg til at vi regner med det ville tatt mer ressurser enn vi har tilgjengelig i dette prosjektet. Derfor var det naturlig å undersøke hvilke kriterier applikasjonen vår hadde til maskinlæringsrammeverk, for så å ta et valg som tilfredsstilte disse best.

Blant de viktigste kriteriene var det at maskinlæringsmmeverket måtte være i stand til å forstå norsk. Dersom den naturlige språkgjenkjenningen til maskinlæringsrammeverket ikke forstod norsk, ville vi måttet eksponert applikasjonen for et nytt, og potensielt risikabelt ledd for oversetting av spørsmål og svar til brukere. Spesielt bemerkelsesverdig er det at oversettelsesprogrammer ikke nødvendigvis klarer å skille mellom stedsnavn, som potensielt ville bli mye brukt i applikasjonen.

Da Wit.ai var det eneste kjente maskinlærerammeverket som både støttet norsk og var gratis, samt tilfredsstilte applikasjonens primære behov, falt valget på nettopp Wit.ai *(25 Chatbot Platforms: A Comparative Table, 2017).*

Wit.ai er et "open source" maskinlæringsrammeverk eid av Facebook. Wit har støtte for svært mange språk i forhold til andre maskinlæringsrammeverk, deriblant norsk. I tillegg er Wit kostnadsfritt å bruke, uansett om bruken er kommersiell eller privat *(Natural Language for Developers, 2018).*

5.2 Backend

Serversiden av applikasjonen implementerer all logikk presentert i klassediagrammet *(Vedlegg 5 - Klassediagram backend).* Serversidens ansvarsområde er brutt ned til spesifikke funksjonaliteter i funksjonslisten som beskriver hver av disse med attributtene funksjon, kompleksitet, type og begrunnelse for funksjonalitet *(Vedlegg 6 - Funksjonsliste backend).*

Vi så det som fordelaktig at serversiden på enklest og tryggest vis kunne interagere med maskinlæringsrammeverket, og anerkjente derfor muligheten til å benytte en populær SDK for Wit som et godt valg. Wit.ai har i skrivende stund (24.04.2018) tre programmeringsspråk de støtter med SDKer, Python, Ruby og Node.js. Det var også mulig å ta i bruk wit.ais HTTP API direkte.
Et naturlig steg for oss var å undersøke disse miljøene for å se hvilke som er mest populære. Vi fant ut at Wits SDK for Python var den mest populære, i den forstand at det er mest aktivitet i "repositorien", blant de tre. I tillegg har flere gruppemedlemmer god erfaring med Python, som gjorde oss trygge på at vi var i stand til å utvikle en serverside applikasjon av god kvalitet. Derfor valgte vi Python som programmeringsspråk for "backend".

5.2.1 Flask

Flask er et rammeverk som tilbyr web-funksjonalitet i Python *(Flask, 2018).* Ved valg av web-funksjonalitet, hadde vi i hovedsak to muligheter, Flask eller Django.
I hovedsak hadde vi et kriterie til et slikt rammeverk, nemlig at det skulle være mulig å sette opp et REST API til systemet vårt. Begge de ovennevnte rammeverkene er i stand til å løse dette, men Flask har et betraktelig mer minimalistisk design som bidrar til at den er merkbart raskere enn Django. Django tilbyr en rekke funksjonaliteter Flask ikke har, men da vi ikke hadde andre behov enn simpel implementasjon av REST API valgte vi Flask *(Django, 2018), (Brown, 2016).*

5.2.2 Flasgger

Flasgger er et rammeverk til Flask som baserer seg på Swagger for å tilby lett leselig API dokumentasjon tilgjengelig i nettleseren, samtidig som det tilbyr testing innad i rammeverket for besøkende *(Flasgger, 2018) (Swagger, 2018)*.

Vi valgte Flasgger for å unngå at brukere behøver å laste ned eventuelle tredjeparts programvare som for eksempel Postman for å teste APIet vårt.

5.3 Frontend

Klientsiden av applikasjonen implementerer en chat hvor ApiBots serverside logikk er tilgjengelig gjennom kall til APIet, se *(Kap. 4.4 Prototype)*. Chatten kan settes "på toppen" av eksisterende, uavhengige nettsider på en slik måte at den ikke forstyrrer nettsiders brukergrensesnitt eller funksjonalitet. Dersom vi setter chatten inn i "Model View Controller" (MVC) designmønsteret kan vi si at chatten representerer et "view" lag, da den i hovedsak skal representere data i form av tekst og kart.

Ved valg av "frontend" programmeringsspråk eller rammeverk utførte vi en kort analyse hvor vi først og fremst vurderte hvor godt ulike JavaScript rammeverk var egnet applikasjonsdomenet vårt. Når vi vurderte disse rammeverkene hadde vi flere kriterier vi la til grunn. Fordi vår kompetanse og erfaring påvirker vår evne til å utvikle en applikasjon med høy kvalitet og høy grad av sikkerhet la vi nettopp dette til grunn som en betydelig faktor. I tillegg vurderte vi nøye hvilke rammeverk som er mest populære ved blant annet å se på markedsandeler de ulike rammeverkene har. Vi antar at jo mer populært et rammeverk er, jo mer ressurser vil det i utgangspunktet være tilgjengelig for oss, dermed er også popularitet en vesentlig faktor. Eksempelvis ser vi at søk på Angular og React per 24.04.2018 returnerer henholdsvis 107 968 og 82 428 spørsmål på StackOverflow *(Angular, 2018) (React, 2018)*.

Internt i gruppen har flere god erfaring med React.js. I tillegg ser vi at store aktører i næringslivet på Sørlandet som EVRY og skatteetaten behøver kompetanse på React.js. Derfor falt valget naturlig på React.js som "frontend" rammeverk for chatten. React.js er et JavaScript bibliotek for utvikling av brukergrensesnitt på "web". Biblioteket retter seg mest mot utvikling av webapplikasjoner, men brukes også ofte i utvikling av generelle nettsider. React.js kalles for et komponent-basert bibliotek ettersom utviklere oppfordres til å lage individuelle komponenter for nettsiden som kombineres til en helhet. Dette gir gode muligheter for gjenbruk av komponenter i den samme nettsiden eller andre applikasjoner.

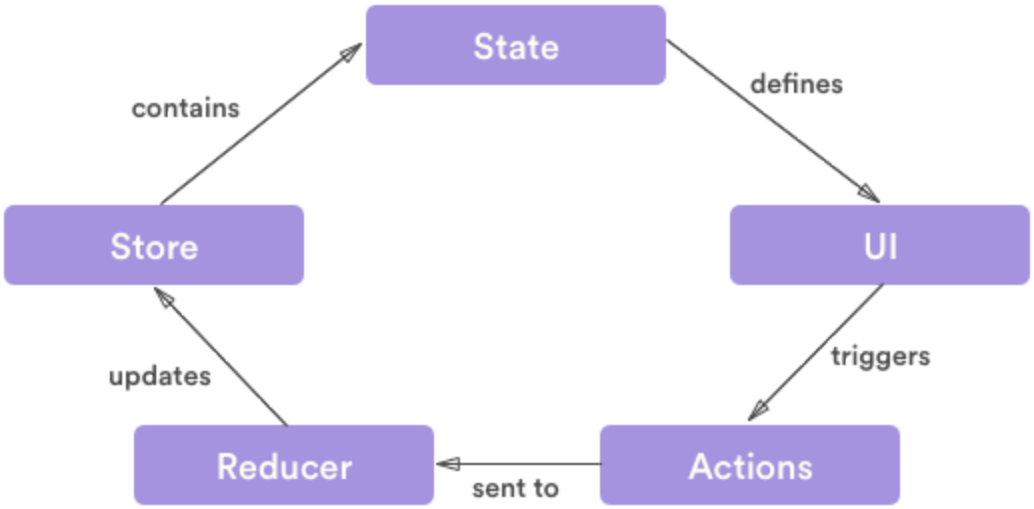
I motsetning til andre rammeverk, som Angular og Vue for utvikling av “web” applikasjon er React biblioteket mer effektiv på oppdatering av informasjon. React bruker en virtuell representasjon av “Document Object Model” (DOM) som den konstant sjekker opp mot den nåværende visningen (render). Dersom data eller elementer i den virtuelle modellen endres, vil React kun oppdatere elementet som endret seg. I tidligere rammeverk og bibliotek ville en endring i elementer kreve en total gjenoppbygging av DOM, uansett hvor mye som endret seg. I både små og store applikasjoner har dette en tydelig forbedring på ytelse per visnings-syklus.

Name	react-v16.1.0-keyed	angular-v5.0.0-keyed	vue-v2.5.3-keyed	angular-v1.6.3-keyed	glimmer-v0.8.0-keyed	ember-v2.16.2-keyed
create rows Duration for creating 1000 rows after the page loaded.	187.6 ± 4.3 (1.1)	185.7 ± 7.8 (1.1)	169.2 ± 3.6 (1.0)	223.0 ± 9.1 (1.3)	349.2 ± 21.3 (2.1)	361.2 ± 23.6 (2.1)
replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	165.2 ± 7.0 (1.0)	179.3 ± 6.5 (1.1)	161.8 ± 3.9 (1.0)	221.9 ± 13.5 (1.4)	245.6 ± 18.2 (1.5)	238.8 ± 7.7 (1.5)
partial update Time to update the text of every 10th row (with 5 warmup iterations) for a table with 10k rows.	93.6 ± 5.6 (1.3)	73.5 ± 4.9 (1.0)	168.1 ± 7.4 (2.3)	82.5 ± 2.8 (1.1)	116.6 ± 4.4 (1.6)	128.5 ± 3.2 (1.7)
select row Duration to highlight a row in response to a click on the row. (with 5 warmup iterations).	12.4 ± 4.1 (1.0)	7.6 ± 4.0 (1.0)	9.8 ± 2.5 (1.0)	9.0 ± 3.5 (1.0)	13.5 ± 3.6 (1.0)	8.6 ± 3.5 (1.0)
swap rows Time to swap 2 rows on a 1K table. (with 5 warmup iterations).	19.6 ± 4.7 (1.0)	20.1 ± 4.2 (1.1)	21.8 ± 4.5 (1.1)	19.0 ± 5.6 (1.0)	25.0 ± 4.9 (1.3)	20.6 ± 3.6 (1.1)
remove row Duration to remove a row. (with 5 warmup iterations).	51.5 ± 2.0 (1.1)	46.1 ± 2.6 (1.0)	52.5 ± 1.8 (1.1)	47.2 ± 1.9 (1.0)	60.2 ± 5.6 (1.3)	53.6 ± 2.5 (1.2)
create many rows Duration to create 10,000 rows	2033.7 ± 32.0 (1.3)	1682.0 ± 53.1 (1.1)	1521.4 ± 55.7 (1.0)	2137.3 ± 49.1 (1.4)	2321.8 ± 84.6 (1.5)	2406.0 ± 44.6 (1.6)
append rows to large table Duration for adding 1000 rows on a table of 10,000 rows.	271.8 ± 9.9 (1.1)	257.6 ± 11.1 (1.0)	338.4 ± 10.3 (1.3)	358.7 ± 65.4 (1.4)	435.1 ± 63.9 (1.7)	432.3 ± 31.4 (1.7)
clear rows Duration to clear the table filled with 10.000 rows.	224.4 ± 6.0 (1.1)	360.3 ± 16.4 (1.7)	240.9 ± 11.4 (1.1)	494.9 ± 57.8 (2.4)	209.6 ± 6.7 (1.0)	262.3 ± 10.2 (1.3)
startup time Time for loading, parsing and starting up	49.4 ± 0.7 (1.0)	88.8 ± 2.9 (1.8)	48.4 ± 2.4 (1.0)	103.5 ± 3.3 (2.1)	61.3 ± 1.5 (1.3)	163.4 ± 3.4 (3.4)
slowdown geometric mean	1.10	1.16	1.16	1.35	1.39	1.55

Figur 4 - React sammenligning med andre rammeverk

5.3.1 React Redux

Redux er et JavaScript rammeverk for tilstands-behandling innen "web"-applikasjoner. Redux tar applikasjons nåværende tilstand til et eneste uforanderlig JavaScript objekt. I tillegg tillater Redux en applikasjons enkelte komponenter å hente ut kun informasjonen de trenger. Resultatet er en uavhengig data syklus som lar utvikleren utføre handlinger og kommandoer uten å måtte bry seg om hvilke deler av datastrukturen som reagerer eller lytter til endringene *(Redux,* 2018).



Figur 5 - React Redux

Modellen ovenfor beskriver den gjennomgående syklusen i en Redux applikasjon. "Store" objektet er stedet hvor applikasjonens data blir lagret. Den inneholder data som er hentet fra databaser, internett eller andre filer. "Store" inneholder også applikasjonens "State"-objekt. "State"-objektet former og definerer utseendet til applikasjonens "User Interface" (UI), f.eks om en knapp skal være tilgjengelig eller ikke. Dersom en bruker interagerer med UI (f.eks trykker på en knapp) vil det utløse "Actions" i systemet. "Actions" blir sendt ut fra komponentet til resten av datastrukturen uten å vite hvem som tar i mot denne meldingen. Disse meldingene blir dermed fanget opp av "Reducers" som inneholder logikk og endringer i "State" basert på typen "Action" de mottar. Når en "Reducer" er ferdig med å behandle en hendelse og har endret "State" (f.eks knapp er satt til "utilgjengelig"), oppdateres applikasjonens "Store" til den nye versjonen. Denne gjennomgangen begynner dermed på nytt.

Vi valgte å bruke Redux i vårt prosjekt ettersom det er et av de mest brukte rammeverkene for tilstands-behandling innen React.js, noe som gir oss mange ressurser for hvordan en Redux applikasjon skal utvikles. I tillegg tilbyr Redux veldig god skalerbarhet. For en liten applikasjon kan Redux bli mye arbeid uten at en får mye igjen for det, men i vårt tilfelle ønsket vi å tilrettelegge for at applikasjonen kan skalere over tid.

5.3.2 Axios

Axios er et JavaScript bibliotek som forenkler og forbedrer bruk av HTTP dataoverføring. Axios lar utviklere enklere bruke forespørsler til en API/server ved å automatisk gjøre om responsen til et JSON data objekt som er enklere å håndtere. Axios har også god feilhåndtering, samt mulighet for å avbryte forespørsler *(Axios, 2018)*. Dette er fordeler man ikke kan få gjennom Fetch, som er et standard bibliotek for å foreta API forespørsler, og var dermed grunnlaget for at vi valgte å gå for Axios som API bibliotek, fremfor Fetch *(Arnold, 2017).*

5.4 Versjonskontroll

Prosjektets kodebase er lagret i VSTS (*Kap. 2.3. VSTS- Metodeverktøy*) hvor vi har brukt versjonskontrollprogramvaren Git som holder orden på filhistorikken og versjonerer filer etterhvert som de endres. Ved hjelp av git har individuelle utviklere i teamet kunnet jobbe med filer i kodebasen på forskjellige versjonskontrollgrener ved å laste opp og ned de siste versjonene for så å sammenligne forskjellige versjoner og senere slå sammen koden *(Blischak et al., 2016).*

6. Prosjektgjennomføring

Dette kapittelet dokumenterer gjennomføringen av prosjektet. Gjennomføring av prosjektet har skjedd ved hjelp av sprinter på to uker med mål om å levere nye versjoner av produktet etter hver sprint. Kapittelet dokumenterer alle sprintene kronologisk for å skape et riktig bilde av prosjektets livssyklus. Sprintene som dokumenteres korrelerer med prosjektets faktiske iterasjoner.

Sprint planlegging

Felles for alle sprintene, med unntak av pre-sprinten, er at utforming av sprint “backloggene” skjedde ved at vi brøt ned elementer fra “product backlog” på en slik måte at vi satt igjen med sprint backlogger med håndfaste og overkommelige arbeidsoppgaver for hver sprint.

I sprinter hvor vi har implementert brukerhistorier var vi nødt til å plukke ut spesifikke funksjonaliteter tilhørende hver brukerhistorie for å skape håndterbare oppgaver. Dette ble gjort ved å diskutere hvilke funksjonaliteter vi antok de ulike brukerhistoriene krevde basert på modeller som ikke-funksjonelle krav, funksjonelle krav, funksjonslisten og klassediagrammet *(Vedlegg - 4, 5, 6, 7 Ikke-funksjonelle Krav, Klassediagram, Funksjonsliste, Funksjonelle krav).*

Estimering av arbeid
Tidskostnaden av hver arbeidsoppgave i alle sprintene ble estimert ved å gjennomføre "planning poker". Dette ble gjort i samarbeid med produkteier i samtlige sprinter.

Sprint review og retrospekt
Samtlige sprinter ble avsluttet med et sprint "review" med oppdragsgiver, og et internt retrospekt møte.

Kvalitetssikring
For å sikre høy prosess og produktkvalitet har vi fulgt rutiner og regler for ulike prosesser i prosjektet som alle er beskrevet i kvalitetsdokumentet og kapittelet kvalitetssikring, se *(Vedlegg 9 - Kvalitetsdokument) (Kap. 7 Kvalitetssikring).*

6.1 Styringskomiteemøter

Som enda et styringselement for både oppdragsgiver og institutt for informasjonssystemer ble det gjennomført tre styringskomiteemøter i løpet av prosjektets utførelse. Hensikten med disse møtene har vært å presentere prosjektets status, vesentlige utfordringer, prioritere ressurser, samt diskutere og foreta sentrale avgjørelser. Ved styringskomitemøtene ble det også gitt tilbakemeldinger fra veileder, produkteier og teknisk veileder basert på prosjektets status. Dette ga føringer på hva gruppen måtte fokusere på videre i utviklingsprosessen *(Vedlegg 13 - Referat styringskomiteemøter).*

6.2 Sprinter

Dette kapittelet beskriver, forklarer og reflekterer over innholdet i hver av prosjektets syv sprinter. Et mer overordnet blikk som også presenterer hver sprints milepæler finnes i ukeplanen vår, *se (Vedlegg 12 - Ukeplan).*

6.2.1 Pre-Sprint (Uke 2)

Pre-Sprinten startet 8. Januar og hadde en varighet på 5 dager. Denne forberedende sprinten skulle utforme prosjektets fremdriftsplan. Fremdriftsplanen tok for seg hvilken metode vi skulle anvende gjennom prosjektet, arbeidstid og en grov ukeplan som inneholdt en oversikt over overordnede arbeidsoppgaver med milepæler. Da vi ikke hadde vedtatt metoden for prosjektet ble det ikke foretatt noen formell planlegging eller tidsestimering av arbeidsoppgavene for denne sprinten.

Utførelse

Under pre-sprinten ble det hovedsakelig lagt planer og grunnlag for hvordan prosjektet skulle utformes. Det ble også brukt mye tid innad i gruppen på diskusjoner rundt prosjektets omfang, og spesifikt hva som faktisk skulle produseres. Denne konkretiseringen av prosjektets omfang ble diskutert med oppdragsgiver, som aksepterte innholdet.

Det ble også utarbeidet en ukeplan for å fastslå prosjektets ønskede fremdrift. Da spesifikke elementer i prosjektet ikke var avklart enda, som for eksempel brukerhistorier, ble denne versjonen av ukeplanen relativt overordnet.

Resultat

Som et resultat av pre-sprinten fikk vi utarbeidet fremdriftsplanen (Vedlegg 15 - Fremdriftsplan) og førsteutkast til ukeplanen som inkluderte detaljene og milepælene til sprint 1. Denne inneholdt også noen grove estimater i form av milepæler vi ønsket ferdig implementert i kommende sprinter (Vedlegg 12 - Ukeplan Pre sprint).

Refleksjon

Strategien med å ha en kort pre-sprint for å tilrettelegge arbeid for de kommende sprintene var en beslutning vi mener hadde stor verdi for prosjektets helhet. Dette fordi vi fikk tidlig i prosjektet satt klare rammer for hva prosjektet skulle inneholde, samtidig som gruppen fikk en sterkere felles forståelse for produktet.

6.2.2 Sprint 1 (Uke 3 og 4)

Sprint 1 startet den 15.januar med en varighet på 8 arbeidsdager og ble avsluttet den 25.januar.

Sprint planlegging

I første sprint planlegging møte ble vi enige med produkteier om å gjennomføre en innledende analyse- og designfase til prosjektet som hovedsakelig skulle inneholde

intervju av relevante ressurspersoner og utforming av brukerkrav, i tillegg til en rekke designmodeller.

Både vi og produkteier så det også som passende at når problem- og applikasjons-domene i høy grad var utformet at vi tok stilling til hvilke utviklingsverktøy vi skulle bruke i prosjektet. I tillegg skulle vi i denne sprinten opparbeide oss innledende kompetanse på disse ulike utviklingsverktøyene.

Siden mye av arbeidet i de kommende sprintene ville basere seg på arbeidet som ble gjort i denne sprinten, forutså vi at det ville gå mye tid til diskusjon rundt de forskjellige statiske og dynamiske modellene, brukerhistoriene og spesifikasjonene for prosjektet.

I denne sprinten hadde vi ca. 220 arbeidstimer totalt til rådighet. Derfor planla vi i denne sprinten arbeidsoppgaver som til sammen, etter våre estimeringer på tidspunktet, ville ta 180 timer å utføre. Det ble ved planleggingen tatt høyde for at uforutsette hendelser som forelesninger og nødvendige møter kunne oppstå, derav bufferen på 40 timer.

Utførelse

Vi startet sprinten med å ta kontakt med Kristiansand servicetorv, ettersom tilgang til relevant data var helt avgjørende for at vår chatbot løsning skulle kunne gi verdifulle tilbakemeldinger til sluttbrukere. Dette var også viktig for å avdekke hvor stort behovet for en chatbot kunne være for kommunen. Da vi fikk avtalt og gjennomført et intervju med enhetslederen for Kristiansand kommune, fikk vi svar på mange relevante spørsmål og en oversikt av type og antall henvendelser kommunen typisk får. Videre fikk vi innsikt i at det er over 22 faste ansatte som blant annet arbeider med å håndtere henvendelser fra innbyggere. Basert på vår forklaring av ApiBot og enhetslederens antagelser om den og teknologien den implementerer, anslo hun at grovt 20% av dagens arbeidsoppgaver kunne automatiseres ved å benytte en slik chatbot *(Vedlegg 2 - Intervju enhetsleder servicetorvet)*.

Dessverre fikk vi ikke tilgang til Kristiansands datasett, da datasettene ikke var strukturert nok til å tilgjengeliggjøres. Likevel ble vi tipset om at Stavanger kommune besatt over 140 datasett åpent tilgjengelig for alle, som vi kunne bruke som utgangspunkt.

Med dataen fra dette intervjuet, var det mulig å avdekke brukerkrav *(Vedlegg 3 - Brukerhistorier)* og ikke-funksjonelle krav *(Vedlegg 4 - Ikke funksjonelle krav)*, da det ga oss økt forståelse for problemdomenet ApiBot skal løse *(Vedlegg 2 - Intervju Enhetsleder Servicetorvet)*.

Innenfor design utarbeidet vi flere modeller som hjalp gruppen til å få en bedre helhetlig felles forståelse av hvordan systemet skulle fungere.

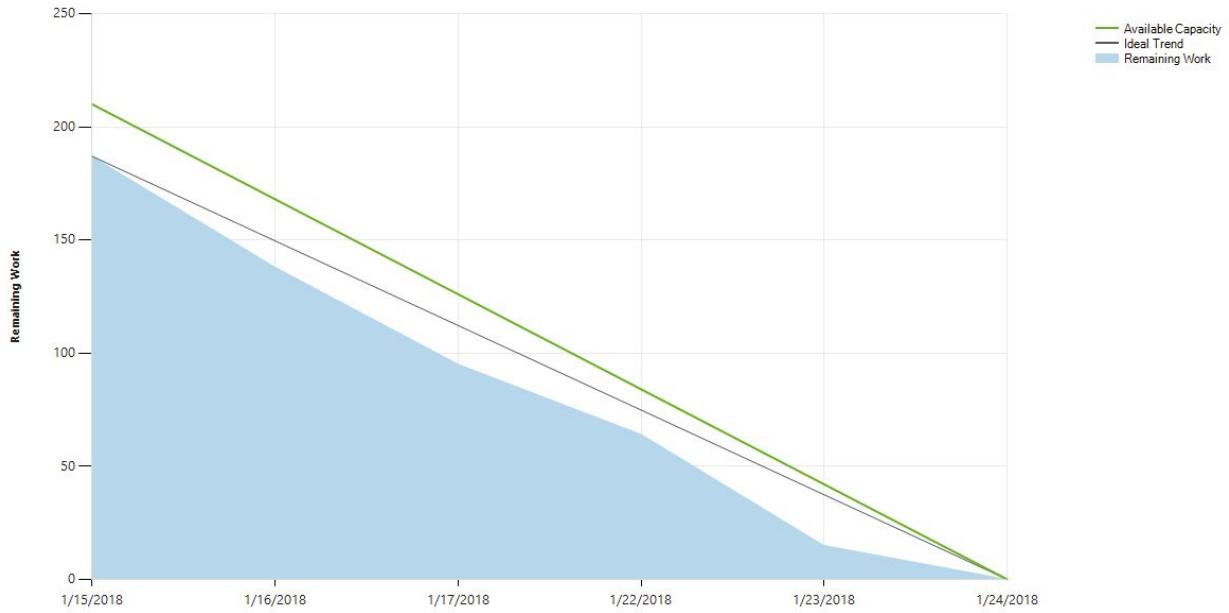
Basert på vår kunnskap fra systemutviklings fagene ved Universitetet i Agder, samt anvendelse av OOAD utformet følgende modeller: rikt bilde, klassediagram, sekvensdiagram, “flowchart”, funksjonsliste og hendelsestabell *(Kap. 4. Design)*. Modellene ble ansett som ferdige i denne sprinten foruten om klassediagrammet, hvor vi forventet endring på grunn av modellens kompleksitet. Dette var både fordi klassediagrammet i vårt tilfelle skulle representere hele serversiden av systemet, noe som var vanskelig å forutse så tidlig i prosjektet, og fordi informasjon fra andre modeller ga oss ny kunnskap vi måtte integrere i klassediagrammet. Klassediagrammet ble derfor revidert flere ganger i denne og senere sprinter, ettersom vi så en nødvendighet for dette.

I denne sprinten ble det gjort flere viktige valg av utviklingsverktøy vi skulle bruke. Først og fremst måtte vi finne et kostnadsfritt ML med støtte for norsk språkgjenning som vi kunne bruke, samt et programmeringsspråk for å blant annet integrere dette. Sentralt for begge disse avgjørelsene var det å utrede hvilke kriterier applikasjonens problemdomenet hadde. I tillegg vektla vi faktorer som Wit.ai sine SDKers popularitet samt vår kompetanse når vi valgte serverside programmeringsspråk, se *(Kap. 5.1 Maskinlæringsrammeverk, 5.2 Backend)*.

De som ikke hadde tidligere erfaring med programmering i Python brukte tid denne sprinten på å sette seg inn i det, hvor de fokuserte på objektorientert programmering og enhetstester.

Resultat

Milepælene for denne sprinten relaterte seg til informasjonsinnhenting, utforming av brukerkrav og applikasjonsdomene, i tillegg til valg og kompetansebygging innenfor utviklingsverktøy. Med unntak av å ferdigstille brukerhistoriene og klassediagrammet ble samtlige milepæler for denne sprinten oppnådd (Vedlegg 12 - Ukeplan *Sprint 1 Milepæler*).



Figur 6 - Burndown Chart, Sprint 1

Figur 6 viser hvordan "burndownchartet" vårt så ut etter at den første sprinten var gjennomført. Ut i fra figuren ser vi at mengden arbeid og hvor mye kapasitet vi estimerte, stemte godt. Under hele sprinten følger mengden resterende arbeid den ideale trenden for sprinten. Årsaken til dette er at oppgavene under denne sprinten ble brutt ned til et overkommelig nivå, og at vi ikke møtte på noen store utfordringen under selve sprinten.

Refleksjon

Vi mener fremgangsmåten ved å innhente informasjon fra mulige brukere av systemet, samt utarbeide forskjellige modeller for å gi oss et felles overordnet blikk for hvordan systemet skulle operere, var en fornuftig måte å begynne den første sprinten. Dette bidro til en betraktelig sterkere felles forståelse internt i utviklingsgruppen og opp mot produkteier, av hva systemet skulle løse, og nettopp hvordan dette burde skje.

Ved å anvende OOAD for å utforme de ovennevnte modellene oppnådde vi bedre kommunikasjon både internt i utviklingsteamet og med produkteier. Dette er fordi OOAD krever at man finner objekter som tar utgangspunkt i prosjektets krav, grupperer de, eksplisitt beskriver hvordan de interagerer med hverandre og skal oppføre seg.

Når vi estimerte tidskostnaden for hver arbeidsoppgave i sprinten opplevde vi at flere i utviklingsteamet satt med individuelle oppfatninger av hva de ulike modellene skulle illustrere. Dette førte til at allerede under utførelsen av "planning poker" benyttet vi

muligheten til å klargjøre og enes om hver arbeidsoppgaves hensikt. Arbeidsoppgaven som gikk inn i sprint "backloggen" som vi i ettertid så vi underestimerte mest var "Utforming av brukerhistorier med akseptansekriterier og tester", som var estimert til å ta 32 timer. Dette tok nærmere 50 timer å gjennomføre fordi brukerhistoriene har vært gjennom flere iterasjoner for å tilfredsstille produkteiers ønsker.

Under retrospekt møtet som ble gjennomført til slutt i sprinten oppdaget vi at vi ville vært tjent med å utføre "peer review" på samtlige arbeidsoppgaver før vi flagget de som fullført for å unngå å unødvendig måtte revidere arbeidsoppgaver. Det ble derfor innført en regel, som senere ble beskrevet i kvalitetsdokumentet, om hverandrevurdering på alt arbeid. I tillegg opplevde samtlige i utviklingsteamet at vi har hatt mange gode diskusjoner, og at en forutsetning for slike diskusjoner er å være kritiske til hverandre på en positiv måte.

6.2.3 Sprint 2 (Uke 5 og 6)

Sprint 2 startet den 29.januar og ble avsluttet etter 8 arbeidsdager den 8.februar.

Sprint Planlegging

Hovedfokuset under denne sprinten var revidering av brukerhistoriene og klassediagrammet slik at de representerte en virkelighet alle interessenter kunne enes om. Et annet sentralt tema for denne sprinten var utarbeidelsen av kvalitetsdokumentet.

Da vi ønsket å levere noe fungerende programvare, og ikke bare dokumenter denne sprinten, ble vi enige med produkteier om å sette opp en instans for APIet vårt, i tillegg til strukturen for automatiserte tester i VSTS.

Det første styringskomiteemøtet skulle gjennomføres tidlig neste sprint, så det ble satt av tid i denne sprinten til å forberede dette. Under estimeringen av de ni utvalgte arbeidsoppgavene for denne sprint "backloggen" estimerte vi en totalt tidskostnad på 312 timer. Dette er opp mot forventet kapasitet på 336 timer. Også i denne sprinten tok vi høyde for uforutsette hendelser vi antok ville inntreffe.

Utførelse

Klassediagrammet og brukerhistoriene våre fikk en ny gjennomgang etter interne diskusjoner i utviklingsteamet samt med produkteier, som førte til vesentlige endringer *(Kap. 3.4 Brukerhistorier, 4.3 Øvrige Modeller)*.

Det ble også utarbeidet et kvalitetsdokument under denne sprinten. Kvalitetsdokumentet dokumenterer forskjellige rutiner, kriterier, standarder og retningslinjer rundt testing,

implementasjon, risikovurdering og involvering av produkteier *(Vedlegg 9 - Kvalitetsdokument).*

Denne sprinten ble det gjennomført en fullstendig risikovurdering, der vi kartla forskjellige hendelser som kunne ha en innvirkning på prosjektet og hvordan de kunne påvirke oss. De forskjellige hendelsene ble kategorisert etter alvorlighetsgrad og sannsynligheten for at de kunne inntreffe. Alle risikovurderingene ble samlet i en matrise for bedre oversikt over samtlige risikoer. Denne matrisen ble strukturert på en slik måte at vi enkelt kunne håndtere nye risikovurderinger som kom underveis i prosjektet fortløpende for så å legge de inn i matrisen *(Vedlegg 10 - Risikomatrise).*

Under denne sprinten utforsket vi "continuous integration" build prosesser i VSTS. I første omgang benyttet vi oss av forhåndsdefinerte CI moduler, da vi antok at disse ville dekke behovene våre for testing av serverside koden. De automatiske testene satt opp i CI ble satt til å kjøre etter at en spesifikk "trigger" ble utløst, nemlig at ny kode integreres i "development" eller "master" grenene.

Resultat
Milepælene i denne sprinten omhandlet revidering av klassediagram og brukerhistorier, dokumentering av elementer i kvalitetsdokumentet, oppsett av API instansen og API dokumentasjon. Brukerhistorier ble utbedret og sendt til produkteier, men vi rakk ikke å få aksept eller ytterligere endringsforslag tilbake før sprintens tid var omme. Derfor ble samtlige milepæler, med unntak av brukerhistoriene, oppnådd denne sprinten *(Vedlegg 12 - Ukeplan Sprint 2).*

Refleksjon
Kvalitetsdokumentet og revisjoner av det ble utarbeidet denne sprinten sammen med produkteier og teknisk veileder fra EVRY. Disse ga oss lærerike tilbakemeldinger og innspill på elementer vi burde inkludert i dokumentet. I utgangspunktet estimerte vi utformingen av kvalitetsdokumentet til å ta 32 timer, men fordi vi fikk tilbakemelding på dokumentet, og selv så muligheten til å legge til flere potensielt viktige elementer, valgte vi å prioritere å bruke mer ressurser på denne arbeidsoppgaven i denne sprinten.

Vi opplevde noen utfordringer med automatisering av tester i VSTS' CI relatert til oppsett av Python tester. Dette skyldes i hovedsak at vi angrep selve arbeidsoppgaven før vi foretok noen undersøkelser av hvordan oppsett og kjøring av testplaner i VSTS fungerer. Vi ville vært godt tjent med å foreta en grundigere analyse av dette, da vi antar at dette ville økt vår forståelse for teknologien, og potensielt derfor kunne integrert testene på betraktelig mye kortere tid. Det ble i utgangspunktet satt av 6 timer til og fullstendig

integrere og sette opp struktur for automatiserte tester i CI, mens det i realiteten tok nærmere 16 timer.

Vi grovt overestimerte tidskostnaden planlegging av styringskomiteemøte ville ta, og det var derfor ikke nødvendig å gjøre vesentlige endringer i sprinten, til tross for underestimering av andre oppgaver. Vi valgte å nedprioritere kompetansebygging i programmeringsspråket Python for å komme i mål med milepæler for sprinten.

Under retrospekt møtet for sprinten kom det frem punkter alle på utviklingsteamet mente var gode. Det å fortsette og kreditere hverandre opplever samtlige er noe vi må fortsette med da det gir alle den respekten og eierskapsfølelsen til prosjektet de fortjener.

6.2.4 Sprint 3 (Uke 7 og 8)

Sprint 3 startet den 12.februar og ble avsluttet 22.februar, etter 7 arbeidsdager.

Sprint planlegging

I denne sprinten fokuserte vi på implementasjonen av brukerhistoriene 1 og 2 *(Vedlegg 3 - Brukerhistorier 1, 2)*, og oppgavene knyttet til disse. Et mål produkteier ønsket denne sprinten skulle ha, var å lage en demo med et enkelt brukergrensesnitt av brukerhistoriene som skulle presenteres på “sprint review” møtet. I tillegg skulle verktøyene for kontinuerlig integrering settes opp til å kjøre nye testtyper ettersom de ble lagt til i prosjektet. Som tidligere nevnt, var det tidlig i denne sprinten første styringskomiteemøte skulle gjennomføres, og det ble derfor satt av tid til det.

Etter å ha brutt ned brukerhistoriene til mindre håndterbare oppgaver satt alle utviklere med antagelser om hvor komplekse oppgavene var, og derfor hvor mye tid de ville kreve. Det var derfor mulig å gjennomføre en fruktbar tidsestimering. Med en maksimal kapasitet på 294 timer for hele sprinten budsjetterte vi med arbeidsoppgaver forventet til å koste 240 timer. Dette gapet på 54 timer skyldes at vi tok høyde for forventede forelesninger og møter.

Utførelse

I denne sprinten implementerte vi store deler av skallet til arkitekturen i systemet. Det vil si at vi opprettet klasser for alle klassene i klassediagrammet, hvor logikken til flere av klassene ikke ble implementert da de ikke skulle ferdigstilles denne sprinten.

Brukerhistorie nummer en er en svært stor brukerhistorie som dekker mye av ApiBots applikasjonsdomene, da den krever at API brukere kan sende simple spørringer med spørsmål, og få tilbake simple relevante svar. For å suksessfullt ferdigstille hele

brukerhistorien implementerte vi mye logikk i klassene “Rest Endpoint”, “Controller”, “Rest Point”, “API constants” og “Respons generator” fra klassediagrammet *(Vedlegg 5 - Klassediagram backend)*. I tillegg til implementasjonen av disse klassene var det nødvendig å trene maskinlæringsrammeverket på to kategorier for å både nå denne milepælen og illustrere ovenfor produkteier at det som er implementert fungerer.

Den andre brukerhistorien som ble implementert i sprinten omhandler tilgjengelighet og dokumentasjon av APIet i en så stor grad at tredjeparter skal kunne benytte APIet i egne applikasjoner. En vesentlig del av denne brukerhistorien ble ferdigstilt i sprint 2, nemlig oppsett av API instansen. Derfor dreide arbeidet i denne sprinten om å utrede, velge og implementere et rammeverk for API dokumentasjon.

For å tilfredsstille produkteiers ønske om en visuell prototype av brukerhistoriene implementerte vi en simpel chat, mye lik den tiltenkte funksjonaliteten i brukerhistorie 7 *(Vedlegg 3 - Brukerhistorier 7),* hvor vi koblet APIet vårt på *(Kap. 4.4 Prototype)*.

Første styringskomiteemøte ble gjennomført tidlig i denne sprinten hvor vi fikk presentert prosjektet for første gang til produkteier og veileder. Her fikk vi tilbakemelding av produkteier på at vi har vært for passive i kommunikasjonen, da både hun og teknisk veileder er usikre på hvordan prosjektets fremgang lå an *(Vedlegg 13 - Referat styringskomiteemøte 1)*.

Under “Sprint Review” møtet for sprinten kom det frem at vi ikke bryter ned arbeidsoppgavene i sprinter nok, da vi hadde enkelte arbeidsoppgaver på over 60 timer. Dette hindrer utviklere i teamet å ansvarliggjøre seg selv for en spesifikk håndterbar oppgave. I tillegg ble vi rådet til å stå under daily scrum møter, noe vi ikke hadde praktisert frem til dette tidspunktet. Ved å stå under daily scrum kan møtene bli mer konsise og effektive, da folk generelt ikke ønsker å stå samt oppmerksomheten blir sentrert rundt den som snakker. Videre viste vi frem prototypen med stor suksess, da oppdragsgiver ble imponert og fornøyd med prosjektets fremgang.
Selve testingen av koden ble i første omgang gjort via enkle enhetstester. Hovedsakelig ved hjelp av “assert-statements” som sjekker resultatet på metodekall fra de forskjellige modulene som ble implementert under denne sprinten. Espen kommenterte at vi testet for naivt, det vil si vi forventet at verdiene som ble sendt inn var av samme type og skrevet på en sånn måte at funksjonene våre i de fleste tilfeller ville fungere. Vi ble derfor anbefalt å også implementere negative tester.

Resultat
Milepælene tilhørende sprinten relaterte seg til spesifikke funksjonaliteter brukerhistorie 1 og 2 behøver, samt en prototype for å illustrere implementert funksjonalitet. Samtlige milepæler ble oppnådd denne sprinten, men det er verdt å merke seg at flere av klassene som er implementert vil bli utvidet senere for å tilfredsstille andre brukerhistorier *(Vedlegg 12 - Ukeplan Sprint 3)*.

Refleksjon
Som en respons på tilbakemeldingen fra styringskomiteemøte utarbeidet vi en plan for å kontinuerlig rapportere prosjektets status, fremgang, ressursforbruk og utfordringer i form av en ukentlig rapport tilsendt oppdragsgiver *(Vedlegg 17 - Eksempel ukentlig rapport)*. Hvorvidt dette var et tilstrekkelig tiltak for oppdragsgiver ble diskutert med produkteier kort tid etter styringskomiteemøte, og det ble etablert en enighet om at dette var et svært positivt tiltak.

Vi tok med oss bemerkningene fra Sprint Review møtet og besluttet å stå oppreist under daily scrum, samt bryte ned oppgaver på en slik måte at de er estimert til å ta maksimalt 8 timer. I tillegg besluttet vi at vi burde prioritere å forbruke ressurser til å bygge kompetanse på negative tester slik at vi allerede fra neste sprint kan implementere tester i tilstrekkelig grad.

Under Sprint Retrospekt møtet kom det frem at vi har sentrale utfordringer relatert til anvendelsen av BDD *(Kap. 7.1.1 Utfordringer)*, og vi ble derfor enige om å adoptere Test Driven Development (TDD) isteden. I tillegg hadde vi et tilfelle hvor det ble pushet en commit direkte til development grenen ved et uhell, noe som ikke skal forekomme i henhold til grenmodellen vi følger. *(Vedlegg 9 - Kvalitetsdokument Branching modell),* det ble etablert en enighet i teamet om at alle har ansvar for å unngå dette.

6.2.5 Sprint 4 (Uke 9 og 10)
Sprint 4 startet den 26.februar og ble avsluttet den 8.mars, etter 8 arbeidsdager.

Sprint Planlegging
I likhet med sprint 3 fortsatte vi utviklingen av Apibot med videre implementering av brukerhistoriene 3 til 7. Dette innebar viktig funksjonalitet som blant annet “web-plugin” og svaralternativer dersom Apibot var usikker på spørsmålet gitt.

Fordi vi har valgt rammeverk som tilfredsstiller enkelte behov til applikasjonen, var brukerhistoriene 3,4 og 5 allerede implementert gjennom maskinlæringsrammeverket.

Derfor har vi ikke direkte selv implementert disse på noen annen måte enn at de er integrert i applikasjonen *(Vedlegg 3 - Brukerhistorier 3, 4, 5, 6, 7).*

Basert på tilbakemelding fra teknisk veileder om at vi testet for naivt prioriterte vi å bruke tid på å både bygge kompetanse og implementere negative tester i systemet.

Som et resultat av tilbakemelding fra oppdragsgiver om å bryte ned oppgaver bedre, brøt vi ned brukerhistoriene til oppgaver som ikke oversteg tidsestimater på åtte timer per oppgave. Dette førte til at vi fikk betraktelig flere elementer i sprint backloggen, men færre estimert timer per element.

I denne sprinten hadde vi en kapasitet på 210 timer og anslo at arbeid relatert til implementering ville ta 124 timer. Arbeid med rapporten ble også planlagt i denne sprinten, hvor kvalitetssikring, analyse og metodikk skulle bli dokumentert. Disse arbeidsoppgavene ble ikke tatt direkte med i estimeringen av arbeidsoppgaver, først og fremst fordi vi ikke ønsket at rapportarbeid skulle påvirke sprintens “burndown chart”. I tillegg ønsket vi ikke å flagge rapportoppgaver som “ferdige”, da vi var klare over at disse med intensjon vil bli arbeidet med over tid, etterhvert som prosjektet utspiller seg.

Utførelse

Ved oppstart av sprinten oppdaget vi at flere av brukerhistoriene vi skulle implementere, nesten kunne anses som ferdig ettersom de valgte rammeverkene allerede implementerte de. Brukerhistorie 3 og 5 som begge tok for seg funksjonalitet til maskinlæringsrammeverket, som trening og tillegg av flere entiteter, var begge funksjonaliteter som allerede eksisterte. Brukerhistorie 4 var også implementert av maskinlæringsrammverket, da det allerede støttet norsk.

Vi fortsatte arbeidet med opplæring av maskinlæringen i denne sprinten ved at vi implementerte to nye kategorier for spørringer i maskinlæringsrammeverket og tilgjengeliggjorde de relevante datasettene på serversiden av systemet.

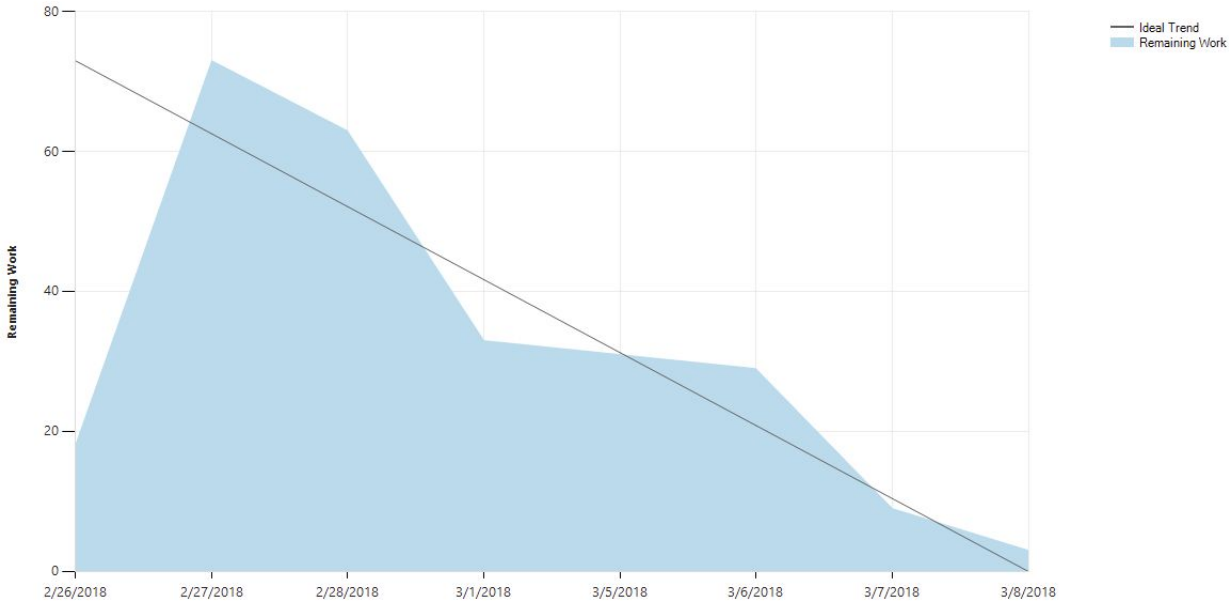
Vi utviklet funksjonalitet som gjør det mulig for ApiBot å stille oppfølgingsspørsmål i form av svaralternativer av sannsynlige kategorier dersom den er usikker på brukerens spørsmål *(Vedlegg 3 - Brukerhistorier 6).*

Vi arbeidet gjennom sprint 4 med å implementere en “plugin” for ApiBot “frontend” chatten. Deretter satte vi opp og implementerte en nettside hvor “pluginen” fungerte som en demonstrasjon av både modulariteten og selve kommunikasjonen med chatbotten.

Endringene som ble diskutert under retrospekt møtet i sprint 3 ble forsøkt implementert under denne sprinten. Det vil si at vi implementerte full TDD tilnærming for måten vi testet og utviklet de forskjellige modulene på, samt endret og forbedret tidligere tester og testklasser. I tillegg til å endre testrammeverket endret vi også hvilke typer tester vi implementerte for både de nylig implementerte og eldre modulene ved at vi også implementerte negative tester. I praksis betydde dette at vi implementerte tester som sendte inn feilaktig input, i den forstand at parameteret enten er av feil type eller lengde.

Resultat

Milepælene denne sprinten dreide seg i hovedsak om å implementere de resterende "must" og "should have" brukerhistoriene, 3, 4, 5, 6, 7, i tillegg hadde sprinten egne milepæler relater til å dokumentere prosjektet i rapporten. Alle milepælene med unntak av dokumentasjon av kvalitetssikring i rapport ble oppnådd i denne sprinten. Årsaken til at kvalitetssikring ikke ble fullstendig dokumentert denne sprinten er at det er et større kapittel i rapporten som med stor sannsynlighet vil endre seg utover prosjektet, dermed valgte vi å ikke flagge oppgaven som ferdig på dette tidspunktet. (Vedlegg 3 - *Brukerhistorie 3, 4, 5, 6 og 7*) *(Vedlegg 12 - Ukeplan Sprint 4)*.



Figur 7 - Burndown Chart, Sprint 4

Figur 7 viser "burndown chart" fra den fjerde sprinten vår. Av figuren kan vi lese at på dag to i sprinten nådde vi høydepunktet for gjenstående arbeid i sprinten. Årsaken til dette er at på sprintens første dag var tre nøkkelpersoner fraværende grunnet sykdom, og det ble derfor besluttet å avvente sprint planlegging til neste dag slik at alle utviklere

kunne bidra med sine viktige innspill. Forskjellen fra “burndown chartet” fra den første sprinten er at noen av oppgavene ble tidligere vurdert til opp mot 64 timers arbeid, mens denne sprinten ikke hadde arbeidsoppgaver som oversteg 8 timer.

Refleksjon
Fordi tre av brukerhistoriene ble flagget som ferdige som et resultat av valgte utviklingsverktøy, ser vi i ettertid at vi kunne ha estimert disse godkjenningene mye tidligere dersom vi hadde undersøkt maskinlæringsrammeverket enda nøyere.

Implementeringen av brukerhistorie 6 var en spesielt interessant problemstilling, da den kunne løses på flere, svært ulike måter *(Vedlegg 3 - Brukerhistorier 6)*. For eksempel kunne det blitt løst ved å implementere en sesjonsbasert løsning hvor brukere får egne sesjoner. Utfordringen med en slik løsning er at det umulig å avgjøre når en sesjon er ferdig, og derfor er det ikke mulig å vite når en bruker stiller et helt nytt spørsmål som ikke er relevant til de tidligere spørsmålene. Den andre løsningen er å implementere et nytt endepunkt som kun tar i mot kategorier og gjør oppslag direkte i datasettene, for så å generere en generell svarstreng som returneres til sluttbrukeren. Disse mulige løsningene ble presentert til teknisk veileder hos EVRY, Espen Limi. Vi foreslo å implementere den sistnevnte løsningen, da den kostet mindre ressurser å implementere, samtidig som den løste utfordringen på en mer presis måte ved ulike spørsmål ikke påvirker hverandre. Veilederen var enig, og vi implementerte derfor denne løsningen.

I vår sprint retrospekt spesifiserte vi at kvalitet og estimering skulle fortsette å spille en sentral rolle i fremtidige sprinter og planleggingsfaser. Estimering fortsatte å være en lærende prosess gjennom alle våre sprinter. Vi oppdaget også verdien av å bryte opp brukerhistorier til mindre oppgaver som ikke skulle ta mer enn åtte timer å fullføre. Verdien av dette var en mer flytende arbeidsprosess som tillot bedre planlegging og estimering, i tillegg var det enklere for utviklere i teamet å ansvarliggjøre seg selv for hver dag som gikk i sprinten ved å plukke arbeidsoppgaver hver dag. Dette tiltaket slo også godt ut på resultatet av antall timer brukt vs. estimert, da spriket ble mindre.

6.2.6 Sprint 5 (Uke 12)
Sprint 5 startet den 19.mars og ble avsluttet 22.mars etter 4 arbeidsdager. Årsaken til at sprinten hadde en kortere varighet enn normalt skyldes eksamen i IS-305.

Sprint planlegging
Målet for denne sprinten var å ferdigstille de resterende brukerhistoriene i prosjektet *(Vedlegg 3 - brukerhistorier 8, 9, 10 og 11)*. Disse innebar implementasjon av ny svarkategori fra Apibot som byggetillatelse, hjelp funksjonalitet i chatten i form av en

brukerguide, benytte GPS-lokasjon fra sluttbrukere for å generere bedre svar og implementasjon av kartrammeverk *(Vedlegg 3 - Brukerhistorier 8, 9, 10, 11)*.

Denne sprinten hadde vi en total kapasitet på 168 timer. Som det kommer frem i funksjonslisten er elementene relatert til disse brukerhistoriene av kompleksistetstypen “medium” eller “simpel". Derfor ble det estimert at elementene i denne sprintens “backlog” ikke ville ta mer enn 100 timer å fullføre. De resterende timene vi hadde tilgjengelig var planlagt å forbruke på forelesning i IS-304 samt skriving av rapport. Under sprint planleggingen etablerte vi en enighet med produkteier om å kutte to brukerhistorier fra prosjektets “product backlog”. Dette ble gjort for å kunne implementere andre brukerhistorier med tilstrekkelig kvalitet, samt bruke mer tid på utarbeidelse av prosjektrapport.

Utførelse

Det ble implementert en hjelpefunksjon som kjøres automatisk når chatten starter eller dersom sluttbruker sender inn strengen “hjelp”. I tillegg ble det implementert funksjonalitet i chatten for å hente inn og utnytte innbyggeres posisjon. Ved å benytte seg av innbyggeres posisjon menes det at posisjonen brukes aktivt i chatbottens logikk for å hente så nær data som mulig, samt generere svar som også tar utgangspunkt i brukers posisjon *(Vedlegg 3 - Brukerhistorier 8)*.

Det ble også gjort arbeid opp mot brukerhistorien som omhandler presentasjon av vedlegg til innbyggere *(Vedlegg 3 - Brukerhistorier 9)*. Kjernefunksjonalitet som å slå opp og returnere vedlegg, samt trening av maskinlæreren på kategorien “Byggesaker” ble suksessfullt ferdigstilt. For å implementere hele brukerhistorien måtte det også implementeres funksjonalitet i chatten for å åpne vedlegg slik at innbyggere faktisk kunne lese de. Det er flere måter å åpne vedlegg på i en chat som er en del av en nettside, eksempelvis kan vedlegget åpnes direkte i chatten eller i en ny fane sluttbrukerens nettleser. Her valgte vi i utgangspunktet å åpne vedlegget direkte i chatten.

Fordi nettlesere allerede har funksjonalitet som kan hente brukeres posisjon, i tillegg til at vi tidligere har lagt til rette for å håndtere brukeres posisjon, krevde det lite ressurser å implementere posisjonsdeling og posisjonsavhengige svar *(Vedlegg 3 - Brukerhistorier 10*).

Under implementering av kartrammeverket i ApiBot møtte vi på utfordringer. Når innbyggere stiller spørsmål i chatten og svaret fra APIet vårt inneholder GPS koordinater, presenteres kartet til innbyggere hvor markører representerer steder innbyggeren

sannsynligvis er interessert i basert på spørsmålet *(Vedlegg 3 - Brukerhistorier 11)*. Spesielt utfordrende for oss var det å selektere hvilke data hver markør skulle presentere til innbyggere, da vi ønsket å kun presentere relevant data. Vi tok derfor utgangspunkt i de mest vanlige henvendelse som servicetorvet hadde gitt oss (Vedlegg 2 - Intervju enhetsleder servicetorvet). I tillegg opplevde vi utfordringer i forbindelse med integrering av kartrammeverket Google Maps da vi ikke hadde satt opp restriksjoner for hvilke maskiner og IP-adresser som har tilgang til å kalle dette eksterne APIet. Dette førte til at det gikk noen timer til debugging for å faktisk finne ut hva som var problemet. Når problemet åpenbarte seg ble det raskt fikset, da vi simpelthen kunne gi oss selv autorisasjon.

Resultat
Sprintens milepæler var spesifikke funksjonaliteter relatert til brukerhistorie 8, 9, 10 og 11 *(Vedlegg 12 - Ukeplan Sprint 5)*.

“Hjelp” funksjonaliteten ble suksessfullt implementert i denne sprinten, hvor den ga brukeren en hjelpe-streng tilbake dersom de spurte om dette.

“Vedlegg” funksjonaliteten ble ikke ferdig i denne sprinten. Gjennom diskusjon og tilbakemelding fra produkteier, bestemte vi oss for at vedlegget skulle åpnes i en ny fane, som gjorde problemstillingen betraktelig mye enklere, siden ApiBot trengte kun å levere en lenke. Da produkteier ønsket en liten justering av denne funksjonaliteten ble ikke brukerhistorien flagget som “ferdig” i prosjektets “product backlog”, til tross for at mye relevant funksjonalitet var implementert.

Til tross for de ovennevnte utfordringene ble kartrammeverket integrert i chatten, på en slik måte at det kunne åpnes, og markørene ble generert i kartet med ønsket data. Vi oppdaget likevel en feil i kartet som resulterte i at markørene ble plassert på feil sted. Vi rakk ikke å fikse dette i denne sprinten, og måtte derfor flytte det resterende arbeidet relatert til denne brukerhistorien til neste sprint. Da vi ikke var klar over disse utfordringene når vi utførte sprintens planlegging, hadde vi ikke estimert tilstrekkelig med tid for å kunne ferdigstille denne brukerhistorien. I utgangspunktet var det forventet at det skulle ta totalt 32 timer å implementere alle funksjonalitetene i forbindelse med kartet, men bare i denne sprinten tok disse oppgavene over 50 timer å gjennomføre.

På grunn av dette ble samtlige milepæler med unntak av brukerhistorie 9 og 11 ferdigstilt denne sprinten.

Refleksjon
Ved å hente ut data fra intervjuet fant vi ut hvilke data som innbyggere sannsynligvis foretrekker å få presentert når de henvender seg til kommunen. Dette beviser at intervjuet vi tidligere hadde gjennomført også på dette tidspunktet i prosjektet hadde en verdi, da vi hadde et behov for reell statisk data.

Dersom vi tidligere hadde opparbeidet oss god forståelse for aktuelle kartrammeverk vi kunne benytte, kunne vi potensielt ha estimert mer nøyaktig eller til og med unngått noen av utfordringene vi hadde.

Fordi vi tidligere i prosjektet var klar over at brukerens posisjon på et tidspunkt skulle inkluderes i generering av svar har vi vært konsistente med å legge til posisjonsobjekter som parametere der hvor det er relevant. Vi antar at dette har spart oss for å måtte refaktorere både arkitekturen og enkelte funksjoner på serversiden.

"Vedlegg" funksjonaliteten ble et godt eksempel på at det er viktig med en kontinuerlig dialog med produkteier, for å etablere enighet om hvordan en funksjonalitet burde se ut. I dette tilfellet var det heldigvis ikke mye tid som ble tapt, da vi ikke behøvde å gjøre vesentlige endringer. Vi så derimot at dette var et tydelig tilfelle hvor tidlig kontakt med produkteier ved usikkerhet kan resultere i tidsbesparelse, og potensielt derfor også et bedre sluttprodukt.

6.2.7 Sprint 6 (Uke 13 og 14)
Sprint 6 startet Mandag 26. mars og ble avsluttet Torsdag 4. april, etter 6 arbeidsdager. Årsaken til at sprinten hadde færre arbeidsdager enn normalt skyldes helligdager i påsken.

Sprint planlegging
Målet for denne sprinten var å ferdigstille resten av applikasjonen, ved å fullføre brukerhistorie 9 og 11 ettersom disse ikke ble ferdig i sprint 5. Disse brukerhistoriene inneholder viktig funksjonalitet som for eksempel returnering av eventuelle skjemaer som byggetillatelser, samt kart funksjonalitet *(Vedlegg 3 - Brukerhistorier 9, 11)*.

I denne sprinten skulle et Factory Acceptance Test (FAT) møte gjennomføres sammen med produkteier for å kvalitetssjekke det endelige produktet. Dette møtet krevde at vi forberedte oss og selv testet brukerhistorienes akseptanse- kriterier og tester før møtet.

Videre ble det planlagt å arbeide ytterligere med rapporten ved å dokumentere sentrale valg, analyse og kvalitetssikring.

Vi estimerte at sprintens arbeidsoppgaver relatert til å rette opp feil og legge til funksjonalitet i brukerhistorie 9 og 11, samt planlegge, forberede og gjennomføre FAT ville ta 120 timer. Totalt for sprinten hadde vi en kapasitet på 252 timer. De resterende timene vi hadde til rådighet var tiltenkt arbeid med rapport.

Utførelse
I denne sprinten brukte vi majoriteten av tiden på å ferdigstille applikasjonen som skulle gjennomgås i FAT møtet. Dette innebærte å implementere løsninger for å ferdigsstille brukerhistorie 9 og 11, samt teste akseptanse- tester og kriterier for samtlige brukerhistorier.

En tilbakemelding fra styringskomiteemøte var at vi burde formalisere håndteringen av "bugs" *(Vedlegg 13 - Referat styringskomiteemøte styringskomiteemøte 2)*. Derfor standardiserte vi prosessene relatert til håndteringen bugs *(Vedlegg 9 - Kvalitetsdokument 5. Feil og bugs)*. Da VSTS tilbyr gode løsninger for rapportering av feil som lett integreres i metodeverktøyet valgte vi å anvende VSTS også for loggføring av bugs.

Under FAT møtet i slutten av sprinten ble en grundig gjennomgang av produktet vårt gjennomført. Hensikten med dette møtet var å avdekke om produktet vårt tilfredsstilte produkteiers forventninger til produktet. Vi presenterte produktet ved å gå gjennom samtlige akseptane- tester og kriterier for produkteier og teknisk veileder. Teknisk veileder ønsket også å utføre negative test scenarioer som det er forventet at applikasjonen ikke skal respondere normalt på, for å kvalitetssikre applikasjonens robusthet. Eksempelvis testet vi å ikke dele posisjonen med applikasjonen når den ber om det for å se om applikasjonen håndterer slike uventede tilfeller.

Resultat
Vi implementerte de manglende brukerhistoriene, utbedret rapporten og utførte FAT møte. Resultatet av FAT møtet endte med at 6 av 11 brukerhistorier ikke ble fullstendig godkjent, grunnet forskjellige feil eller andre mangler, se *(Vedlegg 11 - FAT1 Referat)*.
Dermed ble ikke milepælen "Applikasjon ferdigstilt" oppnådd.

I forbindelse med rapporten utarbeidet vi flere vesentlige kapitler som for eksempel teknologivalg, analyse og kvalitetssikring. Likevel, valgte vi å ikke flagge milepælene relatert til rapport for denne sprinten som oppnådd, da vi antok at de enten ville få vesentlige tillegg eller bli endret *(Vedlegg 12 - Ukeplan Sprint 6)*.

Refleksjon
Det viste seg at vi ikke hadde utført tilstrekkelig negative tester da vi på FAT møtet avdekket noen feil. En av feilene som ble avdekket relaterte seg til problemstillingen "Hva om brukerne ikke ga oss tilgang til plassering". Dette var et scenario vi hadde oversett under utviklingen, og vi lærte derfor viktigheten av negative tester, da det potensielt kunne bidratt til å forebygge uventet oppførsel fra programmet under denne problemstillingen. Dette er en svært viktig lærdom å ha med seg videre, da tester er essensielt for å vite at funksjonaliteten er som ønsket samtidig som tester lett kan oppdage endringer som påvirker funksjonaliteten i koden.

6.2.8 Sprint 7 (Uke 15 og 16)
Sprint 7 startet mandag 9. april, og fullført Torsdag 19. april, etter 8 arbeidsdager.

Sprint planlegging
Under Sprint 7 endret vi fokus fra utvikling av ny funksjonalitet til forbedring av produktet samt utbedring av feil/forbedringer avdekket under det første FAT møtet. Sprintens "backlog" bestod av 15 elementer, der seks av dem var forbedringer av allerede implementert funksjonalitet, og ni var feil knyttet til forskjellige komponenter. Disse elementene, spesielt feilene, hadde sin opprinnelse fra det første FAT møtet *(Vedlegg 11 - FAT1 Referat)*. Disse feilene og manglene varierte i omfang og kompleksitet, men de var i stor grad mindre oppgaver som ikke krevde mye ressurser for å løse.

Siden FAT v1 ikke ble godkjent ble det avtalt å gjennomføre et nytt FAT møte under denne sprinten for å presentere løsningene på problemene fra FAT v1.

Det kom et ønske fra produkteier om ekstra funksjonalitet som gjaldt en funksjon som lot brukere spørre om fasiliteter i området rundt et bestemt sted. Fordi dette var utenfor prosjektets omfang, ønsket vi ikke å legge til ekstra funksjonalitet på dette tidspunktet, da det ikke gjenstod mye tid av prosjektet.

Sprint "backlog" elementene ble estimert til totalt 28 timers arbeid. Den totale kapasiteten for denne sprinten beregnet vi til 250 timer. Ut over de 28 timene som ble estimert for denne sprinten ble de resterende timene planlagt brukt på rapportskriving og forberedelser til det siste FAT møtet.

Utførelse
Som et ekstra ledd i kvalitetssikring av applikasjonen implementerte vi enhetstester for alle modulene som inneholdt feil på en slik måte at testene fremprovoserte feilene, før vi rettet de opp *(Vedlegg 9 - Kvalitetsdokument, Feil og bugs)*.

Den mest sentrale feilen som ble avdekket under det første FAT møtet var at “front end” chatten skapte feilaktig oppførsel når den ikke får tilgang til brukerens posisjon. Dette ble løst ved å gjøre brukerens posisjonsobjekt valgfritt i de aktuelle funksjonene, da applikasjonen ikke klarte å fullføre funksjonen uten påkrevd posisjon objekt både grunnet rettigheter og kompatibilitet til enhet.

Det dukket opp et uventet problem under sprinten som tok mye tid å løse. Vi nådde begrensningen på VSTS lisensen vår for bruk av skybaserte agenter som gjennomfører stegene i CI testingen vår. Det gjorde at vi måtte definere en ny “build agent” som kunne kjøres på serveren vår sammen med APIet. Dette var ikke en problemfri prosess, da definisjonene vi tidligere hadde brukt var forbeholdt en agent som kjører på en Windows maskin, mens vår server kjører linux. Derfor måtte vi definere våre egne steg ved hjelp av bash scripts som installerer og kjører testene på de forskjellige komponentene av systemet.

En annen feil vi fant etter å ha utført flere negative enhetstester og “ad hoc” systemtester var at når maskinlæringsrammeverket forsøkte å returnere flere ulike kategorier som entiteter, var ikke Wit.ai i stand til å skille de ulike kategoriene. Dette skyldtes strukturen vi tidligere hadde valgt å legge kategorier i, da vi regnet med at det var mulig å motta flere objekter av lik type fra Wit.ai. Vi erkjenner at denne feilaktige antakelsen kunne vært observert tidligere i prosjektet, spesielt når vi iverksatte implementasjon av brukerhistorien som tar for seg flere kategorier *(Vedlegg 3 - Brukerhistorier 6).* Derfor burde vi i sprint 4 avdekket dette ved å bekrefte, enten ved å studere dokumentasjon eller teste at det er mulig å returnere flere objekter av samme type fra Wit.ai. Det skal sies at det ikke krevde mye ressurser å løse denne feilen, da strukturen i Wit.ai enkelt kunne omorganiseres. Som et resultat av dette fjernet vi en klasse fra backend som ble overflødig, og vi reduserte derfor antall operasjoner og effektiviserte applikasjonen.

Resultat

Målet ved sprinten var å ferdigstille applikasjonen og utføre de forbedringene som ble avdekket under det første FAT møtet. Alle oppgavene i sprintens “backlog” ble gjennomført i god tid før de skulle presenteres til produkteier på det andre FAT møtet. Det lot oss fokusere på andre utfordringer som uforventet dukket opp under sprinten, eksempelvis utfordringene med “build agenten”. Den nye build agenten ble satt opp på vår private server på en slik måte at samtlige tester kjøres på lik linje med den forhåndsdefinerte agenten som ble brukt tidligere.

Det andre FAT møtet ble gjennomført mot slutten av denne sprinten hvor samtlige brukerhistorier denne gangen ble godkjent. Det var derfor å anse at produktet betraktes

som ferdig og klart for levering. Alle milepæler for sprinten ble oppnådd *(Vedlegg 12 - Ukeplan Sprint 7).*

Refleksjon
Gjennomføringen av det andre FAT møtet konkluderte denne sprinten. Selve møtet ble gjennomført på en bedre måte enn det første, siden vi var mer forberedt på hva som forventes av applikasjonen. Med dette menes det at vi forstod at ved et FAT møte forventes det at applikasjonen så og si er klar for lansering.

Introduksjonen til disse møtene ser vi på som særdeles lærerik, det gjorde det tydelig for oss hvordan et produkt evalueres av teamet og produkteieren før en eventuell endelig leveranse. Det ble også tydelig hvor viktig helhetlige tester av systemet var. I ettertid ser vi at det hadde vært fornuftig å inkludere utenforstående brukere uten noen forkunnskaper til systemet for å gjennomføre formelle brukertester. Dette kunne ført til at flere feil og forbedringspotensialer ble avdekket før FAT møtet.

Vi etablerte en enighet med produkteier om å ikke implementere hennes ønske om stedsnavn som gyldig input til applikasjonen da dette ville økt prosjektets omfang, og dermed kunne påvirket kvaliteten på sluttproduktet negativt. Til tross for dette har vi vært klar over at en slik funksjonalitet kan være relevant for sluttbrukere av ApiBot. Derfor har vi lagt til rette for at det på et senere tidspunkt enkelt kan implementeres, ved å ha ferdigstilt en modul for konvertering av stedsnavn til GPS koordinater, samt tillate GPS koordinater som parameter der det vil være aktuelt.

Som et resultat av å implementere tester som fremprovoserer feil før en løsning implementeres har vi vært trygge på at løsningene faktisk løser feilene. Dette er fordi det har vært mulig for oss å se at testen provoserer frem en feil i systemet, mens etter vi har implementert en løsning har det vært mulig å se om testen er i stand til å bringe frem den samme feilen. Det kan også hevdes at disse testene vil ha verdi i fremtiden, fordi om samme feilen skulle oppstå på et senere tidspunkt vil testene fange opp disse, og det vil derfor potensielt være enkelt å se hvor feilen oppstår.

Problemene knyttet til de automatiske testene i “CI-en” vår under sprinten belyste hvilke problemer vi kunne møte på dersom vi ble for avhengige av en tredjepart. Ved å sette opp vår egen agent fikk vi derimot mer kontroll over hvordan testingen ble gjennomført, i tillegg så vi en merkbar forbedring i tidsbruken ved hver build.

Vi antok tidlig i prosjektet at Wit.ai fungerte på en spesifkk måte, når den i realiteten fungerte på en helt annen. Slike antakelser, bevisste eller ubevisste, kan skape

utfordringer på et ubestemt tidspunkt som det kan være vanskelig å lokalisere. Derfor har vi lært at det kan være kritisk å opparbeide seg god forståelse for teknologien som anvendes for å unngå feilaktige antakelser.

Under refaktorering av den redundante klassen, feilet andre enhets- og integrasjons-tester som var avhengige av elementer fra denne klassen. Dette gjorde oss derfor tidlig oppmerksom på ringvirkninger denne refaktoreringen hadde, som gjorde at vi raskt kunne løse disse for å forhindre flere feil. Denne situasjonen er et bevis på viktigheten av å gjøre kontinuerlig testing innenfor systemutvikling, da det forenklet og kvalitetssikret refaktoreringen.

Siden dette var den siste sprinten vår, gjennomførte vi en prosjekt dekkende retrospekt der forskjellige aspekter ved prosjektet ble diskutert. Både positive og negative punkter fra tidligere retrospekt ble tatt frem og det ble drøftet hvordan disse var blitt løst eller videreført. Et vesentlig element som ble tatt frem her var at vi erfarte at antallet modeller vi utarbeidet tidlig i prosjektet *(Kap. 6.2.2 Sprint 1)*, var i det meste laget, ettersom vi fikk mindre bruk for flere av disse. Modellene vi anså som relativt overflødig, som først og fremst bare hjalp oss indirekte ved at vi fikk økt felles produktforståelse var rike bilder, tilstandsdiagram, hendelsestabell, sekvensdiagram og flowchart. Vi kunne derfor utnyttet tiden bedre ved å redusert antall overflødige modeller produsert. Likevel erkjenner vi at der og da, skapte samtlige modeller diskusjoner internt i utviklingsteamet som alle bidro til en sterkere felles forståelse av systemets applikasjonsdomene og har derfor gitt oss stor verdi.

Erfaringene vi har opparbeidet oss gjennom dette prosjektet vil være nyttig for fremtidige prosjekter. En oppsummering av vårt læringsutbytte presenteres i kapittel 8 (*Kap. 8. Konklusjon)*.

6.2.9 Uke 17-20

I ukene 17 - 20 arbeidet vi med å ferdigstille rapporten. I tillegg har vi brukt tid disse ukene på å forberede presentasjon av prosjektet for EVRYs lokale avdeling samt institutt for informasjonssystemers EXPO arrangement.

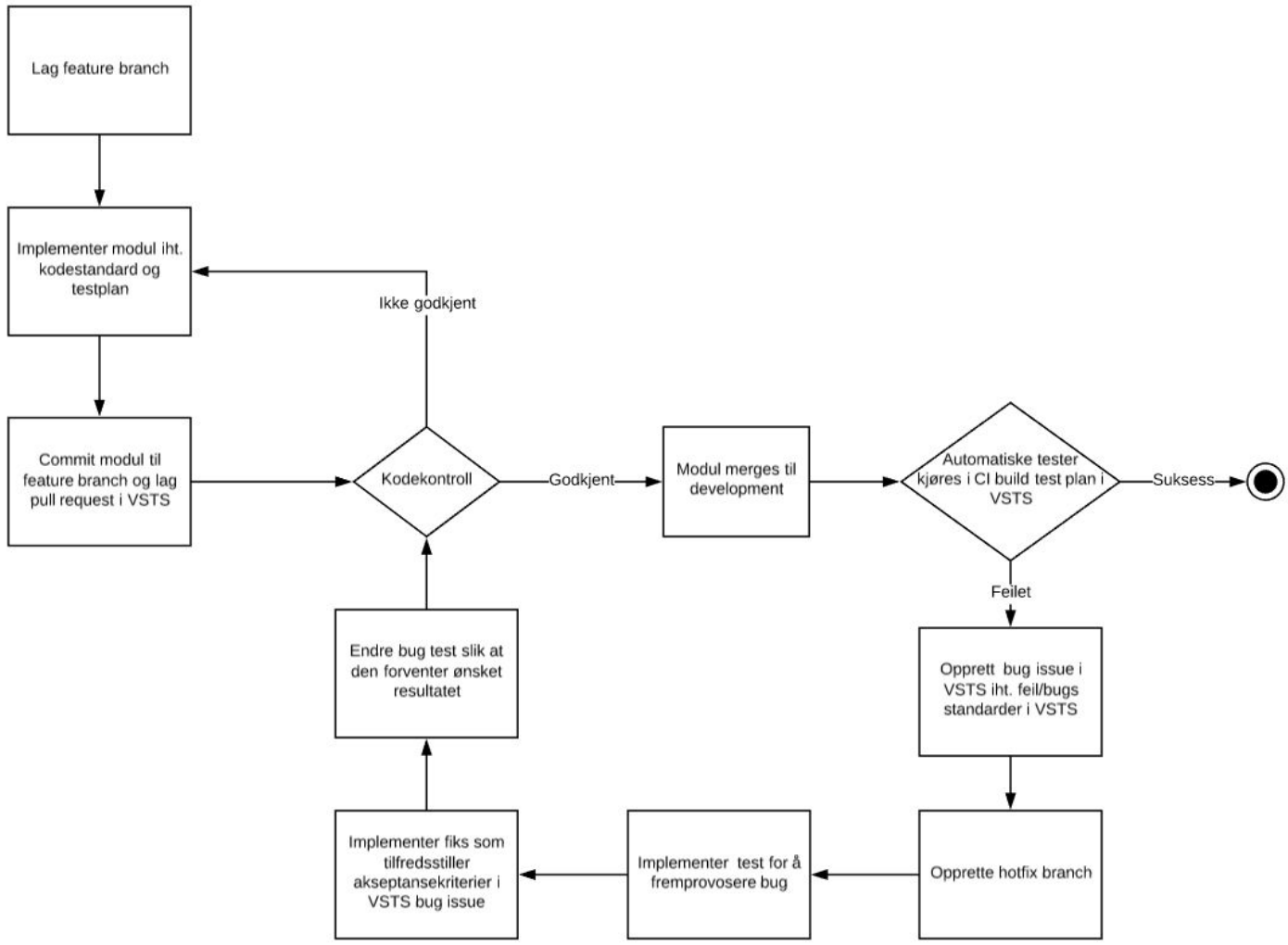
7. Kvalitetssikring

Tidlig i prosjektet ble det utarbeidet et detaljert kvalitetsdokument se (*Vedlegg 9 - Kvalitetsdokument)*. Hensikten med et slikt dokument var å definere rutiner og prosesser som vi som utviklere måtte forholde oss til under utvikling av ApiBot, for å sikre et sluttprodukt med høy kvalitet. Med høy produktkvalitet menes det blant annet at systemet tilfredsstiller sine funksjonelle og ikke funksjonelle krav, og at systemet imøtekommer kunde og sluttbrukers behov og forventninger. Produkt- og prosesskvalitet har vi overholdt ved å nøye undersøke hvilke tilgjengelige rammeverk som finnes og beste praksis konvensjoner som typisk brukes. Samtidig hadde vi et kritisk øye til de prosessene vi tok i bruk, i tilfelle disse kunne utbedres, eller ikke fungerte som de skulle.

Dette kapittelet trekker frem de viktigste rutinene og prosessene vi har anvendt for å øke produkt- og prosesskvaliteten i prosjektet.

Under følger en oversikt over det mest sentrale innholdet i kvalitetsdokumentet.

- Testplan *(Vedlegg 9 - Kvalitetsdokument, Testplan)*
- Kodestandarder *(Vedlegg 9 - Kvalitetsdokument, Kodestandarder)*
- Kode Kontroll *(Vedlegg 9 - Kvalitetsdokument, Kode Kontroll)*
- Feil og bugs *(Vedlegg 9 - Kvalitetsdokument, Feil og bugs)*
- Design standard *(Vedlegg 9 - Kvalitetsdokument, Design standard)*
- Kontinuerlig integrasjon *(Vedlegg 9 - Kvalitetsdokument, Kontinuerlig integrasjon)*
- Risikoanalyse *(Vedlegg 9 - Kvalitetsdokument, Risikoanalyse)*
- Involvering av oppdragsgiver *(Vedlegg 9 - Kvalitetsdokument, Involvering av oppdragsgiver)*



Figur 8 - Illustrasjon av elementer fra kvalitetsdokument.

Figur 8 illustrerer hvordan vesentlige elementer fra kvalitetsdokumentet henger sammen i et scenario der en ny funksjonalitet skal implementeres. Den første prosessen er å lage en egen versjonskontrollgren med utgangspunkt i nyeste stabile versjon, hvor utvikleren kan arbeide med modulen isolert. Deretter må utvikleren forholde seg til kodestandarder og testplanen definert i kvalitetsdokumentet, som blant annet vil si at før selve modulen implementeres, må han skrive test(er) med scenarioer for å følge Behaviour Driven Development *(Solis & Wang, 2011)*. Når modulen er implementert og testet skal utvikleren "pushe" endringene til sin gren for så å opprette en "pull request". Denne vil så bli vurdert av andre utviklere i henhold til kriterier for kodekontroll. Dersom "pull request" godkjennes blir koden den inneholder "merget" til "development" grenen. Når denne grenen mottar nye "commits" vil det kjøres en rekke spesifiserte enhets og integrasjonstester. Resultatet av disse testene sendes til en kanal i Slack slik at vi ser om testene lykkes eller ikke. Om testene ikke passerer vil en utvikler opprette en "bug issue"

i VSTS samt en “hotfix” gren. I denne grenen implementerer utvikleren først en test som fremprovoserer feilen, for så å implementere en tilfredsstillende fiks. Deretter må utvikleren opprette en ny “pull request” som gjennomgår samme prosessene, med samme krav, som en ordinær funksjonalitet. Det er verdt å nevne at prosessene relatert til “bugs” også gjennomføres dersom en “bug” fanges opp på et ubestemt tidspunkt etter den relaterte koden er lagt til i “development” grenen.

7.1 Testrammeverk

Vi har tatt i bruk rammeverk for testing innenfor de forskjellig programmeringsspråkene benyttet i prosjektet, med en overordnet strategi for hvordan vi tester innenfor disse *(Vedlegg 9 - Kvalitetsdokument, Testplan).*

For å strukturere tydelige rutiner og regler for når og hvordan vi skulle utføre testing på ulike deler av applikasjonens kode har vi hatt et ønske om å benytte et relevant, populært og velprøvd testrammeverk. Det finnes flere aktuelle testrammeverk som danner den overordnede strategien for testing, og vi har derfor undersøkt flere mulige rammeverk. Blant annet har vi sett på “Test Driven Development” (TDD), “Behavior Driven Development” (BDD) og “Acceptance Test-Driven Development”. Avgjørende for vårt valg var det i utgangspunktet at vi hadde gode muligheter for å definere gode beskrivelser av oppførselen til ulike funksjonaliteter under ulike betingelser. Slike beskrivelser kan føre til at utvikleres og produkteieres forståelse for en funksjonalitet blir likere, og det er derfor mulig å gjøre endringer før man implementerer en modul. I tillegg har en vesentlig faktor vært testrammeverkets kompatibilitet med prosjektets overordnede metode, Scrum. På grunn av disse faktorene valgte vi i utgangspunktet å anvende BDD.

BDD er et rammeverk for testing som kombinerer flere systemutviklingsmetodikker med hensikt til å eksplisitt beskrive en moduls oppførsel under ulike betingelser, i form av tekst. I tillegg fokuserer behaviour driven development, i likhet med “test driven development”, svært mye på enhetstester ved å kreve at før enhver modul implementeres skal en simpel versjon av testen implementeres. (*Test-driven development, 2018)*

At BDD er en test-dreven utviklingsmetode som er avhengig av korte utviklingssykluser, og benytter prinsipper fra OOAD for å beskrive funksjonalitet alle interessenter forstår, anså vi som positivt da det passer prosjektets behov og metode. Et praktisk eksempel på at BDD var et kompatibelt rammeverk for oss er BDDs måte å strukturere funksjonaliteters ønskede oppførsel. I stor likhet med brukerhistoriene våre beskriver

BDD attributter ved funksjonen som hvem, hva og hvorfor. I tillegg anså vi det som fordelaktig at vi gir oss selv sterkere verktøy for å kommunisere programvarens oppførsel under ulike betingelser tydelig til alle interessenter.

7.1.1 Utfordringer

En sentral utfordring ved anvendelsen av BDD for oss har vært tidspunktet vi valgte å innføre det på. Det ble ikke utredet og valgt en fullstendig teststrategi som inkluderte testmetode før i sprint 3, og på dette tidspunktet hadde vi allerede implementert de to mest sentrale brukerhistoriene i prosjektet. Moduler tilhørende disse ble testet ved hjelp av "Test Driven Development" (TDD).
Utformingen av beskrivelser av programvarens oppførsel i BDD skjer enten ved å definere disse allerede i starten av prosjektet, parallelt med brukerhistoriene, eller i starten av hver sprint som en del av sprint planlegging. Vi opplevde at fordi vi på dette tidspunktet allerede hadde implementert mye, ville vi måttet gjøre vesentlig mye arbeid for å adaptere BDD på en slik måte at også den eksisterende koden fulgte BDDs standarder og prinsipper. Vår bruk av BDD er detaljert dokumentert i Kvalitetsdokumentet kapittel 2.1 *(Vedlegg 9 - Kvalitetsdokument Behaviour Driven Development)*

I tillegg opplevde vi at når man skal lage noe nytt og ikke vet hvordan problemet skal løses, kan det være utfordrende å detaljert beskrive enkelte modulers tiltenkte oppførsel uten å ha studert teknologien godt på forhånd. BDD krever høy forståelse for teknologien og systemet på et tidlig stadium i utviklingen. Dette var en utfordring for oss da vi opplevde at vi ikke besatt denne nødvendige kompetansen på tidspunktet.

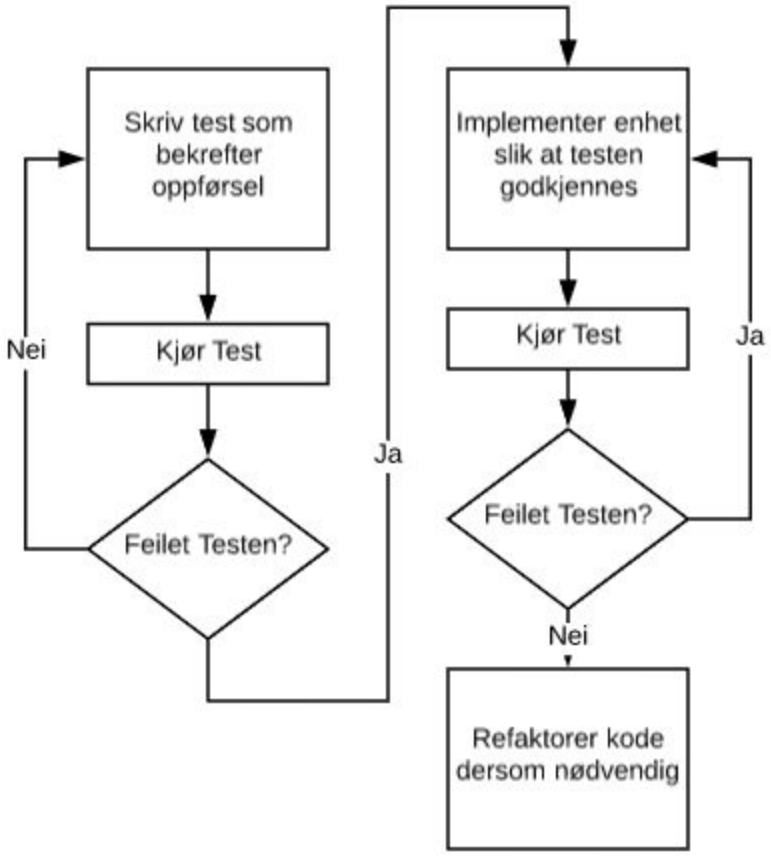
For å løse disse utfordringene, men likevel forsøke å beholde mye av verdien BDD kunne gitt oss, valgte vi å adoptere TDD. Hovedårsaken til dette er at BDD og TDD har store likheter, hvor den eneste forskjellen i hovedsak er at BDD beskriver applikasjonens oppførsel under bestemte betingelser. I tillegg så vi at vårt ønske om å teste samtlige moduler på en ekstensiv måte potensielt kunne bli oppfylt av TDD, da også denne metoden krever implementasjon av tester før enhver modul i seg selv implementeres.

7.2 Testplan

Testplanen vår tar for seg når og hvordan ulike tester skal utføres, og har underveis i prosjektet fungert som en mal for hvordan testing og implementasjon av en modul skal utføres. Ved å ha en slik gjennomtenkt mal sikrer vi at når noe implementeres må det gjøres i henhold til testplanen, som vil sikre at det som er implementert er av ønsket kvalitet. For detaljer om vår testplan se *(Vedlegg 9 - Kvalitetsdokument, Testplan).*

Selve testplanen tar stegvis for seg rutinene rundt en bestemt type testing, alt fra enhetstesting enten via Behave, Jest eller pythons Unittester til akseptansetester med produkteier dekkes her.

7.2.1 Enhetstester



Figur 9 - Enhetstest livssyklus

Fordi vi har fulgt TDD har det første steget i implementasjonen av en ny enhet vært å implementere enhetstesten. Denne skal feile, da modulen som testes ikke er implementert på dette tidspunktet. Deretter implementeres modulen på en slik måte at enhetstesten vil fungere. Ved å følge denne test-drevne utviklingsmetodikken oppnår vi økt grad av kvalitetssikring, i form av at modulene i systemet fungerer på tiltenkt måte, da alle moduler faktisk må testes.

7.2.2. Integrasjonstester

Hver gang to eller flere moduler integreres med hverandre implementeres og kjøres det integrasjonstester for å avdekke feil eller mangler i koblingen(e) mellom modulene. Eksempelvis har det blitt kjørt integrasjonstester mellom funksjoner i klassen relevans, som tar imot og behandler objekter fra maskinlæringsrammeverket og "restpoint", se *(Vedlegg 5 - klassediagram backend)*.

En annen måte integrasjonstesting ble gjort på var ved hjelp av "stubs" og "drivere", for å simulere interaksjonen med andre komponenter. Disse testene baserer seg på returverdier og en forhåndsdefinert struktur for hvordan de skulle se ut i den ferdig implementerte klassen. Det gjorde at vi kunne kjøre integrasjonstester med komponenter som ikke nødvendigvis var helt ferdig. Dette førte også til at implementasjonen av disse måtte følge planen lagt på forhånd for å unngå konflikter og problemer senere.

7.3 Involvering av produkteier

For å levere et produkt kunden faktisk ønsket var det kritisk å involvere produkteieren underveis i prosjektet på både tradisjonelle tidspunkt, som f.eks. sprint planlegging og "review", i tillegg til tidspunkt hvor vesentlige valg måtte gjøres.

Derfor ble produkteier kontaktet ved enhver anledning hvor avgjørelser som potensielt kunne påvirke prosjektet i stor grad måtte tas, for å ta hennes ønsker og anbefalinger med i betraktning når vi fattet en avgjørelse. Denne kontinuerlige kontakten fant sted både etter behov og ved sprint planlegging og "review". Et godt eksempel på kontakt med både produkteier og teknisk veileder utenom fastsatte møter, var avgjørelsen om hvorvidt vi måtte implementere sesjonskommunikasjon. I denne dialogen med oppdragsgiver kommuniserte vi ut to ulike, egenkomponerte forslag som begge kunne tilfredsstilt den aktuelle brukerhistoriens akseptansekriterier, samt gitt sluttbrukeren en god brukeropplevelse *(Kap. 6.2.5 Sprint 4 (Uke 9 og 10))*.

Da hver sprint hadde et maksimalt tidsforløp på to uker ble sprint planlegging og review både benyttet til å planlegge og styre prosjektet fra vår og oppdragsgivers side. I tillegg ble sprint reviews benyttet som en arena for oss for å presentere fremgang i prosjektet hvor vi kunne få tilbakemeldinger fra oppdragsgiver.

Med hyppige leveranser og kontinuerlig kontakt med produkteier, fikk vi jevne og kontinuerlige tilbakemeldinger på helheten og detaljer i prosjektet. Dette gjorde det mulig for produkteier å diskutere konkrete feil, mangler eller anbefalinger med oss, og vi

kunne derfor gjøre endringer enten i applikasjonen eller hvordan vi styrte prosjektet, i god tid.

7.3.1 FAT - Akseptansetesting

Vi foretok to Factory Acceptance Tester (FAT) sammen med produkteier og teknisk veileder for å avdekke om applikasjonen var i en tilfredsstillende tilstand for oppdragsgiver. Under testene gikk vi gjennom hver brukerhistorie og utførte de stipulerte akseptansetestene med sine respektive akseptansekriterier. I vårt første FAT møte ble det avdekket en del små problemer som kjapt ble rettet opp slik at det andre FAT møte uken etter ble applikasjonen godkjent se *(Vedlegg 11 - FAT1 oppsummert).*

7.4 Risikovurdering

I form av risikovurdering utarbeidet vi en risikomatrise som tok for seg konsekvensen og sannsynligheten for uforutsette hendelser, men også tiltak for å forhindre og redusere skaden av disse. Dette endte opp med å bli et nyttig verktøy for oss, ettersom vi i løpet av prosjektet 'mistet' flere gruppemedlemmer, i den forstanden at de fikk fulltidsjobb underveis i prosjektet. Denne problemstillingen ble håndtert ved at vi fortsatte å bryte ned arbeidet til små og håndterbare oppgaver, som alle i utviklingsteamet kunne gi seg selv i VSTS. Dette gjorde at selv om de manglende gruppemedlemmene ikke var til stede, kunne de enkelt ta på seg arbeidsoppgaver, samtidig som de kunne se hvordan fremgangen i prosjektet utspilte seg *(Vedlegg 10 - Risikomatrise) (Vedlegg 8 - Kvalitetsdokument Risikoanalyse).*

7.5 Kodekontroll

For å sikre god kvalitet på kodebasen vår under utvikling av ApiBot, har vi gjennomført kodekontroll ved å bruke "pull request" verktøyet i VSTS. Dette er en standard prosedyre for å slå sammen eksisterende kode med ny kode, der utvikler lager en "pull request" opp mot en eksisterende "branch" i kodebasen. Dermed ser en eller flere av de andre utviklerne i teamet over koden for å kontrollere at vi har fulgt standardene som er satt for prosjektet, og at endringen ikke gir uønsket endringer på eksisterende komponenter i kodebasen. Hvis feil eller mangler blir funnet må utvikler som laget "pull requesten" korrigere koden for så å få en ny kodekontroll før utvikler kan slå sammen koden med kodebasen.

Å kvalitetssikre koden vår på denne måten gjorde at vi oppdaget flere problemer før koden endte opp i development branchen. "Linus's Law" sier at flere øyne på samme kode fører til at flere feil og problemer avdekkes, samt sjansen for at noen har en enkel

løsning de kan presentere øker *(Linus's Law, 2018)*. I tillegg måtte også eieren av "pull requesten" forklare koden sin til andre gruppemedlemmer, noe som minner om en form for "rubber ducking", der man forklarer koden sin høyt, linje for linje til en badeand. Etterhvert som man går igjennom koden innser man hvor problemet ligger og kan gjøre en endring *(Rubber duck debugging, 2018)*.

Denne måten å drive kode kontroll på faller under 'verktøy assistert kode kontroll", da vi har brukt Team Services sin "pull request" funksjon *(Vedlegg 9 - Kvalitetsdokument, Kode kontroll ApiBot)*.

7.6 Guide for maskinlæringsansvarlig

På grunn av maskinlæring sin sentrale rolle i applikasjonen, valgte vi å produsere en guide som maskinlæringsansvarlige skulle kunne ta i bruk, for å lære seg hvordan de kunne trene og legge til nye kategorier til ApiBot. Denne guiden kvalitetssikret vi med at hvert enkelt medlem av gruppen selv gikk igjennom og godkjente denne ved at den var forståelig, samt gjennom tilbakemeldinger fra teknisk veileder. *(Vedlegg 14 - Maskinlæringsguide)*

8. Konklusjon

I vårt bachelorprosjekt fikk vi mulighet til å velge prosjekt selv og har valgt å lage en chatbot for kommunen som skal effektivisere og tilgjengeliggjøre kommunens tjenester for brukere gjennom flere plattformer.
Da oppdragsgiver godkjente prosjektforslaget var vi raskt i kontakt med Kristiansand kommune for å innhente nødvendig data og gjøre oss kjent med kommunens fremgangsmåte og saksbehandling ved mottak av henvendelser for å definere omfang og kompleksitet i forbindelse med analysen.

Under prosjektet tilegnet vi oss en rekke kunnskap rundt forskjellige aspekter ved prosjektgjennomføring. Vi har valgt å trekke frem de viktigste punktene.

I prosjektgjennomføringen tok vi i bruk Scrum som metode, hvor vi benyttet VSTS som Scrum verktøy. Ingen i utviklingsteamet hadde erfaring med bruk av VSTS, og vi støtte derfor på en del utfordringer under startfasen av prosjektet knyttet til bruken av dette verktøyet. I tillegg har vi også opplevd utfordringer i sammenheng med Scrum, som tidsestimering. Dette var utfordringer vi tidlig tilpasset oss til, som resulterte i at vi fikk en god utvikling og progresjon gjennom hele bachelorprosjektet. Vi har også fått

kompetanse i prosessen fra å utvikle en applikasjon fra en ide til et ferdig produkt sluttbrukere kan aksessere.

Estimering av de forskjellige oppgavene under prosjektet er et punkt der vi så forbedringer gjennom hele prosjektet, noe som også belyses tidligere i rapporten under gjennomgangen av de enkelte sprintene. I starten av prosjektet brøt vi ikke oppgavene ned tilstrekkelig, og endte derfor opp med estimater på enkelte arbeidsoppgaver på over 60 timer. Det førte i enkelte tilfeller til at når disse arbeidsoppgavene skulle utføres måtte utviklere bryte ned arbeidsoppgaven for seg selv. Utover i prosjektet ble estimeringene mer nøyaktige, etter tilbakemeldinger fra teknisk veileder hos EVRY, og erfaringer fra tidligere sprinter.

En fallgruve som vi endte opp i tidlig i prosjektet er planlegginsparadokset. Det ble brukt mye tid på å utarbeide flere forskjellige modeller, hvor nytten av disse ikke sto i stil til mengden tid som ble brukt for å lage de. Dersom prosjektet skulle gjennomføres på nytt ville vi laget modeller etter behov istedenfor å lage alle under oppstarten av prosjektet.

Prosjekter har faste rammer for tid, kostnad og funksjonalitet, der resultatet av denne balansen er kvaliteten av produktet. Prosjektet vårt hadde en tidsramme på et semester, det vil si at tidsrommet prosjektet vårt skulle gjennomføres i var fast. Kostnadsrammen vår ble da antall arbeidstimer vi hadde tilgjengelig gjennom hele prosjektet, ut ifra de arbeidsdagene og timene vi satt i starten. Den ønskede funksjonaliteten ble dekket av brukerhistoriene våre. For å oppnå ønsket kvalitet på produktet vårt ble det tydelig under sprint 5 at funksjonaliteten fra to av brukerhistoriene våre burde kuttes til fordel for kvalitetssikring av de allerede implementerte brukerhistoriene. Som resultat av dette fikk vi mer tid til å forbedre eksisterende deler av produktet, og økte derfor kvaliteten på produktet. Gjennom prosjektet så vi at ved å kontinuerlig vurdere og endre de fleksible elementene i jerntriangelet, fant vi den balansen som skapte et akseptabelt kvalitetsnivå.

Det å forholde seg til en produkteier i et prosjekt var en ny erfaring for samtlige i gruppen. Vi fikk raskt tilbakemelding på at mengden rapportering til produkteier og detaljnivået på disse rapportene var for lavt. Ut over i prosjektet ble det satt opp en mer strukturert måte for å involvere produkteieren på et nivå som passet for alle involverte. Den kontinuerlige kommunikasjonen mellom partene gjorde at utfordringer raskt ble drøftet og endringer iverksatt. Verdien av dette ble tydelig når vi møtte på større valg i prosjektet. Klar og konsis kommunikasjon gjorde at vi raskt kunne fortsette utviklingen slik produkteier ønsket.

Kvalitetsplanen vår sammen med akseptansekriteriene for de forskjellige brukerhistoriene dannet kvalitetsdokumentet vårt. Den felles forståelsen for hva gruppen så på som viktig for kvaliteten på produktet vårt, var essensielt for å utvikle et produkt av høy kvalitet. Den gjorde det tydelig for alle hva som var målet, samt hvordan man skulle oppnå det og hvordan det skulle måles i ettertid. Den ga oss også et inntrykk av hvordan man går frem for å gjenskape kvalitet fra prosjekt til prosjekt.

Under hele prosjektet hadde vi et fokus på testing av koden vår. Gjennom utdanningen har vi fått en grunnleggende introduksjon til testing av kode. I prosjektet fikk vi veiledning fra personer med erfaring fra virkelige prosjekter om hvordan man burde gjennomføre forskjellige nivåer av testing, samt hva som burde testes og hvor dypt disse testene skal gå. Vi ble også introdusert for et nytt aspekt ved prosjektgjennomføring, nemlig en FAT med produkteier. FAT møtene viste oss hvor nøye og hvilke typer tester man burde utføre på produktet før det formelt testes med kunde.

Produktet vi ønsket å utvikle i begynnelsen av prosjektet er nå ferdigstilt i henhold til brukerhistoriene og oppdragsgivers forventninger. Dersom vi ønsker å bringe vår chatbot videre, ville vi startet med å presentere resultatet for de relevante organisasjonene slik som Kristiansand Kommune. Vi ser at vi traff markedet bra med vår chatbot da det har kommet flere like løsninger under utviklingsfasen som har blitt publisert på nettsidene hos store norske bedrifter. Dermed antar vi at vårt produkt har stort potensiale for Kristiansand kommune og andre kommuner som sitter på og er i stand til å tilgjengeliggjøre mye offentlige data.

Å opparbeide kompetanse på aktuelle teknologier og metoder brukt i arbeidsmarkedet har vært svært lærerikt. Takket være gode støttespillere både fra universitetet og EVRY, samt samtlige gruppemedlemmers engasjement har vi økt kompetansen vår på flere aspekter innenfor systemutvikling. Det kan også være grunnen til at allerede iløpet av april hadde fire fra gruppen fått relevant jobb under skriving av bacheloroppgaven.

Litteraturliste

25 Chatbot Platforms: A Comparative Table, (2017, Mai), hentet fra: https://chatbotsjournal.com/25-chatbot-platforms-a-comparative-table-aeefc932eaff.

Angular, sist sett 24.04.2018, hentet fra https://stackoverflow.com/questions/tagged/angular.

Arnold, J, (2017, Februar, 09) Fetch vs. Axios.js for making http requests, hentet fra: https://medium.com/@thejasonfile/fetch-vs-axios-js-for-making-http-requests-2b261cdd3af5

Axios, sist sett 04.05.2018, hentet fra https://github.com/axios/axios.

Beck, K. (1999). Embracing change with extreme programming. *Computer*, *32*(10), 70–77. DOI: 10.1109/2.796139

Blischak, J. D., Davenport, E. R., & Wilson, G. (2016). A Quick Introduction to Version Control with Git and GitHub. PLOS Computational Biology, 12(1), e1004668.

Brown, Mike (2016, April, 15) Creating a RESTful API: Django REST Framework vs. Flask, hentet fra https://www.excella.com/insights/creating-a-restful-api-django-rest-framework-vs-flask

Chatbot, sist sett 07.03.2018, hentet fra: http://www.kommuneforlaget.no/digitale%20losninger/kommunikasjon/chatbot/.%20

Django, sist sett 04.05.2018, hentet fra https://www.djangoproject.com/

EVRY, sist sett 05.032018, hentet fra: https://no.wikipedia.org/wiki/EVRY

Flask, sist sett 04.05.018, hentet fra: http://flask.pocoo.org/.

Flassger, sist sett 04.05.2018, hentet fra: https://github.com/rochacbruno/flasgger.

Google Drive - Cloud Storage & File Backup for Photos, Docs & More, sist sett 30.04.2018, hentet fra https://www.google.com/drive/

Kari, sist sett 07.03.2018, hentet fra: https://prokom.no/kari/

Linus's Law, sist sett 06.01.2018, hentet fra https://en.wikipedia.org/w/index.php?title=Linus%27s_Law&oldid=818874010

MoSCoW method, sist sett 14.05.2018, hentet fra
https://en.wikipedia.org/wiki/MoSCoW_method.

Natural language for developers, sist sett 04.05.2018, hentet fra
https://wit.ai/.

Product Owner, sist sett 11.04.2018, hentet fra
https://www.mountaingoatsoftware.com/agile/scrum/roles/product-owner

React, sist sett 24.04.2018 hentet fra
https://stackoverflow.com/questions/tagged/react.

Redux (April 2018), hentet fra
https://redux.js.org/.

Rubber duck debugging, sist sett 17.04.2018, hentet fra
https://en.wikipedia.org/w/index.php?title=Rubber_duck_debugging&oldid=836926495

Scott W. Amber sist sett 07.03.2018, Examining the Agile Cost of Change Curve, hentet fra:
http://www.agilemodeling.com/essays/costOfChange.htm.%20

Scrum master, sist sett 11.04.2018, hentet fra:
https://www.mountaingoatsoftware.com/agile/scrum/roles/scrummaster

Shavlik, J. W., Dietterich, T., & Dietterich, T. G. (1990). Readings in Machine Learning. Morgan Kaufmann.

Slack. (n.d.). Where work happens, sist sett 01.05.2018, hentet fra
https://slack.com/

Solis, C., & Wang, X. (2011). A Study of the Characteristics of Behaviour Driven Development (pp. 383–387).

Swagger, 04.03.2018, hentet fra
https://swagger.io/.

Test-driven development, sist sett 06.03.2018, hentet fra:
https://en.wikipedia.org/wiki/Test-driven_development

Visual Studio Team Services, sist sett 01.05.2018, hentet fra:
https://www.visualstudio.com/team-services/

Vedlegg

Vedlegg 1. Uttalelse fra oppdragsgiver
Vedlegg 2. Intervju Enhetsleder Servicetorvet
Vedlegg 3. Brukerhistorier
Vedlegg 4. Ikke-funksjonelle krav
Vedlegg 5. Klassediagram Backend
Vedlegg 6. Funksjonsliste
Vedlegg 7. Funksjonelle krav
Vedlegg 8. Hendelsetabell
Vedlegg 9. Kvalitetsdokument
Vedlegg 10. Risikomatrise
Vedlegg 11. FAT1 oppsumert
Vedlegg 12. Ukeplan
Vedlegg 13. Referat styringskomiteemøter
Vedlegg 14. Maskinlæringsguide
Vedlegg 15. Fremdriftsplan
Vedlegg 16. Gruppekontrakt
Vedlegg 17. Eksempel ukentlig rapport
Vedlegg 18. Gruppereflekssjon

Vedlegg 1. Attest – Bachelorprosjekt – Våren 2018

EVRY ble høsten 2017 kontaktet av flere grupper som gjerne ønsket å gjennomføre sitt bachelorprosjekt hos oss. Gjennom intervjurunder med flere grupper ble gruppen som består av Mathias Hartveit, Sindre Haugeland, Christian Moen, Martin Nenseth, Erlend Thorvik og Erlend Wiklem den gruppen vi i EVRY ønsket å gjennomføre prosjektet med. Veiledere fra EVRY har vært Espen Limi (teknisk) og Merethe Sjøberg (prosjektgjennomføring/produkteier).

Studentene sto fritt til å selv velge hva prosjektet skulle være og hva slags teknologi de ville benytte. Gruppen valgte å utvikle en chatbot – Apibot - for bruk i kommuners servicetorg som ved bruk av maskinlæring kan gi brukere svar på spørsmål angående ting som f.eks nærmeste parkeringsmulighet. Gruppen valgte i denne forbindelse å ta i bruk teknologi de ikke kjente fra før for å gjøre oppgaven mer utfordrende. EVRY har hatt rollen som kunde og produkteier i prosjektet.

Det var i begynnelsen tydelig at det var uvant for gruppen å skulle arbeide for en kunde, og EVRY har stilt svært høye krav i form av krav til rapportering, involvering i prosjektet samt gjennomføring av FAT (akseptansetest) når det nærmet seg ferdig leveranse. Produktet ble godkjent av EVRY under en vellykket akseptansetest 18. April.

I styringskomitémøter har gruppen vært grundige, men kortfattede i sine presentasjoner av status, endringer, risiko og tatt opp problemstillinger på en konstruktiv måte, og har i tillegg oppført seg svært profesjonelt.

Gruppen har hatt en god prosess i prosjektet, hvor de har oppholdt seg fire dager i uken fra 08.00 – 15.00 i EVRY sine lokaler og har generelt hatt en strukturert arbeidsprosess. De er løsningsorienterte og har vært svært flinke til å gjøre gode og gjennomtenke vurderinger underveis, og foreta endringer hvis nødvendig for å minimere risiko i prosjektet, samt å involvere prosjekteier i denne prosessen.

Gruppen kjente hverandre godt fra før og det har vært moro å se dem gå fra å være en «gjeng» til å bli et svært samkjørt og strukturert Scrum-team med fokus på kvalitet i leveransen uten å gå på bekostning av kreativitet og løsningsevne.

De har bevist at de tar oppgaven seriøst og presterer på et svært høyt nivå både som gruppe og individuelt. EVRY har fått bli kjent med dem og har endt opp med å gjøre tre av dem til våre nye kollegaer i forskjellige avdelinger i Kristiansand. Prosjektet deres har i tillegg fått positiv oppmerksomhet på høyere nivå i organisasjonen da maskinlæring og kunstig intelligens er et satsingsområde i EVRY.

Kristiansand 09.05.2018

Merethe Sjøberg - Veileder

2. Intervju enhetsleder servicetorvet

Dato: 22.01.2018

Varighet: 1 time

Sted: Kristiansand Servicetorv

1. Hvor mange jobber med henvendelser på servicetorget

22 fast ansatt , 24 totalt

2. Hvor mange henvendelser får dere daglig?

Uvisst, men totalt for ett år: 38154.

3. Hvilke typer henvendelser får dere, og hvor mange av hver?

Arrangement: 663

Småbåt: 1099

Oppvekst: 1449

Skatt : 2164

Tekniske tjenester : 2119

Byggesak: 4118

Vært ordinær: 21442

Vært tourist: 3899

Ledsage bevis: 671

4. Hvilke typer henvendelser kommer hyppigst?

Se statistikk(spørsmål 3)

5. Hvor mange henvendelser kommer fra ansatte i kommunen? (skole/barnehage)

Skiller ikke på henvendelser, og oppmøte telles både ved oppmøte og der de blir sendt.

5.1. Hvem er det som fysisk møter opp for å henvende seg?

Flerkulturelle og analfabeter er de som fysisk møter opp og trenger mest hjelp.

På telefon kommer generelt alle spørsmål.

6. Må servicetorget ofte henvise videre, i så fall, hvor?

Ja, spesielt da vi får mange henvendelser på lønn. Det kommer ofte henvendelser til kommunen, når dette skal til leder. Kommunen bruker mye tid på å videreføre til riktig ledd.

7. Kunne en chatbot vært interessant for servicetorget?

Selvfølgelig, det er den veien det går.

8. Hva synes <u>du</u> om en chatbot for servicetorvet?

Spennende, digitalisering er noe kommunen må forholde oss til.

Vi har allerede Byggsøk som er en digital tjeneste for å søke via nett.

9. Hvilke funksjoner hadde vært ønsket i en chatbot?

Enkelt å bruke for sluttbrukere.

10. Hvem kan vi snakke med for å få tilgang til kommunens offentlige datasett?

Sjef IT tjeneste Kristiansand: Ingun Kvivik

11. Hvor mye kan en tekstbasert chatbot, som tidligere beskrevet, hjelpe kommunen?

Mer og mer på sikt, en slik automatiseringsrobott, grovt sett, kan spare kommunen for 20% manuelt arbeid med henvendelser.

Redegjørelse fra Helle på eget initiativ; KRS kommune har en chat hvor ledere for enkelte emner svarer henvendelser fra “kunder” med ferdig utfylte svar. Copy/paste.

Vedlegg 3. Brukerhistorier

1. **Som <u>API bruker</u> ønsker jeg å sende spørringer via ApiBots API for å finne informasjon om kommunale fasiliteter i nærområde.**
Beskrivelse: ApiBot skal gi svar på spørringer gitt fra en API bruker relatert til kommunale fasiliteter.
MoSCoW vurdering: Must have
MoSCoW begrunnelse: Funksjonaliteten brukerhistorien oppfordrer til (lokalsamfunnsrelevante spørsmål) er helt essensielt for å kunne tilby en applikasjon som kan bidra til å automatisere en kommunes henvendelser.
Akseptansekriterier: Gitt at API bruker sender spørring med spørsmålet 'Hvor kan jeg parkere?', skal hen få tilbake en liste over mulige parkeringsplasser. - ApiBot skal kunne ta i mot spørsmål fra API bruker, lete i datasett etter det mest sannsynlige svaret, og returnere det. Er ApiBot usikker, skal ApiBot informere om usikkerheten og be API bruker stille spørsmålet på en annen måte.
Akseptansetest: 1. Utfør POST api kall til https://api.apibot.no/test/message med parameteret "message": "jeg vil gå på toalettet". ● Returverdi: liste over relevante toaletter som finnes i kommunens datasett.

2. **Som <u>API bruker</u> ønsker jeg å ta i bruk ApiBot sitt API, slik at jeg kan dra nytte av ApiBots funksjonalitet i mine applikasjoner.**
Beskrivelse: Brukerhistorien oppfordrer til funksjonalitet som oppretter kontakt med APIet til ApiBot, slik at API bruker enkelt kan utføre API kall.
MoSCoW vurdering: Must have
Begrunnelse: Formålet med ApiBot er først og fremst å tilby API tjenester hvor ApiBot sine operasjoner kan aksesseres av andre utviklere. I tillegg anslår vi at kostnaden av å tilgjengeliggjøre ApiBot via et API som svært liten, mens gevinsten kan være stor for andre tredjeparter.
Akseptansekriterier: Gitt at API bruker besøker https://api.apibot.no/apidocs/ hvor API dokumentasjon er tilgjengelig, skal hen kunne lese dokumentasjon som gir nok kunnskap til å gjøre direkte API kall, eller implementere API kall i egen applikasjon. - Alle API kallene skal være tilgjengelig på et domene. - Domenet APIet ligger på skal ha ha en tydelig dokumentasjon for hvordan APIet kan kalles. - Hver funksjon som er tilgjengelig i APIet skal ha egen dokumentasjon som beskriver hva den gjør

Akseptansetest:

1. Besøk https://api.apibot.no/apidocs/
2. Klikk “show/hide”
3. Klikk på en funksjon
    - Returverdi: Dokumentasjon for funksjonen som beskriver hva den gjør, og hva den returnerer.

. 4. Utføre API kall i egen applikasjon ved å følge dokumentasjon.

3. Som maskinlæringsansvarlig ønsker jeg å kunne trene opp ApiBot med data, slik at den blir mer intelligent.

Beskrivelse:
Ved å trene opp ApiBot, vil man kunne danne et grunnlag for at ApiBot selv vil finne frem riktige svar på ulike spørsmål, uten dette vil ikke ApiBot være i stand til å kategorisere spørsmål.

MoSCoW vurdering: Must have

MoSCoW begrunnelse:
Uten implementasjonen av denne brukerhistorien, vil ikke systemet fungere da det bygges på maskinlæring og analyse av setningsoppbygging, derfor er opplæring, og hvilken grad ApiBot er i stand til å lære, essensielt.

Akseptansekriterier:
Gitt at maskinlæringsansvarlig lærer opp ApiBot med eget datasett og kategorisering, skal ApiBot komme med relevante svar.

- Maskinlæringsansvarlig skal lære opp ApiBot ved å stille spørsmål, gi disse spørsmålene en fornuftig kategori og legge til intensjonen til spørsmålet.
- Med intelligent menes det at ApiBot svarer relevant og logisk på det den blir spurt om eller informere om usikkerhet.

Akseptansetest:
Maskinlæringsansvarlige skal følge dokumentet “guide til maskinkæringsansvarlige”.

1. Gå til tildelt lenke til wit.ai
2. Logg inn på wit.ai kontoen med tildelt brukernavn og passord.
3. Under “Test how your app understands a sentence”, skriv inn spørsmålet “Hvor kan jeg gå på do i nærheten?”.
4. Marker ordet “nærheten” og legg til entiteten “proximity” med verdien “proximity_clode”.
5. Legg til intensjonen “toilet_get”.
6. Trykk “Validate”
    - Maskinlæringen forbedret på kategorien og intensjonen som er valgt.

4. Som API bruker ønsker jeg å sende spørringer med spørsmål på norsk og få svar på norsk, slik at jeg forstår svaret.

Beskrivelse:
ApiBot skal forstå og formulere svar på norsk.

Under intervjuet med enhetsleder for servicetorget i KRS kommune fikk vi informasjon som tilsier at de fleste med manglende norskkunnskaper som henvender seg til kommunen foretrekker å fysisk møte opp på servicetorget. I tillegg forstår og snakker de aller fleste i Norge norsk.
MoSCoW vurdering: Must have
MoSCoW begrunnelse: Fordi alle brukergrupper av ApiBot i utgangspunktet vil befinne seg i Norge, anser vi norsk som mest relevant, og det er derfor viktig at ApiBot behersker norsk.
Akseptansekriterier: Gitt at API bruker sender spørring med spørsmål, skal ApiBot svare på norsk. - Dersom API bruker ikke stiller spørsmål på norsk vil ikke ApiBot forstå spørsmålet, og vil som tidligere nevnt melde ifra om dette, samt be API bruker stille spørsmålet på en annen måte.
Akseptansetest: 1. Utfør POST api kall til https://api.apibot.no/test/message med parameteret "message": "Hvor kan jeg gå på do i nærheten". **Returverdi:** liste over relevante toaletter som finnes i kommunens datasett. **Akseptansetest 2:** 1. Utfør POST api kall til https://api.apibot.no/test/message med parameteret "message": "I would like to go to the bathroom". ● Returverdi**:** Svar fra ApiBot: "Jeg forstår ikke spørsmålet. Prøv å stille spørsmålet igjen på en annen måte."

5. **Som maskinlæringsansvarlig ønsker jeg muligheten å legge til kategorier til maskinlærings-rammeverket, slik at jeg kan trene ApiBot på spesifikke kategorier.**
Beskrivelse: Mengden informasjon og datasett kan endres i bruksperioden. For å implementere ny informasjon må maskinlæringsansvarlig kunne legge til og kategorisere ny informasjon som er tilgjengelig for systemet.
MoSCoW vurdering: Should have
MoSCoW begrunnelse: Med utgangspunkt i at det allerede er definert kategorier henviser denne brukerhistorien til opprettelse av nye kategorier etter den initielle treningen. Da verden er i konstant endring, vil også data ApiBot ønsker å bruke endres, og det er derfor behov for å holde ApiBot oppdatert.
Akseptansekriterier: Gitt at datasett blir lagt til eller utvidet, skal maskinlæringsansvarlig utarbeide passende kategorisering. - Kategorien maskinlæreansvarlig ønsker å legge til må være implementert i ApiBots API. - Dersom kategorien ikke er implementert i ApiBots API, skal maskinlæringsansvarlig rapportere om behov for ny kategori til utviklere, slik at de kan implementere ny kategori og struktur for denne. - Tillegg av kategorier maskinlæringsansvarlig gjør skal kun skje gjennom maskinlæringsrammeverket.

Akseptansetest:

1. Gå til tildelt lenke til wit.ai
2. Logg inn på wit.ai kontoen med tildelt brukernavn og passord.
3. Under “Test how your app understands a sentence”, skriv inn spørsmålet “Hvor kan jeg stemme?”.
4. Opprett intensjon “stemme” med en verdi som tilsvarer kategori implementert i APiet.
5. Trykk valider.
    - Maskinlæringen har en ny intent entitet som reflekterer en kategori i APIet.

6. **Som API bruker ønsker jeg at ApiBot stiller meg oppfølgingsspørsmål når den er usikker på hva jeg spør om, slik at jeg kan få relevant svar på det jeg spør om.**

Beskrivelse:
Dersom API bruker stiller ApiBot et vanskelig eller uklart spørsmål, skal ApiBot foreslå kategorier og/eller formuleringer jeg kan bruke for å stille et spørsmål ApiBot forstår bedre.

MoSCoW vurdering: Should have

MoSCoW begrunnelse:
Det er vesentlig at ApiBot er i stand til å kommunisere hva den ikke er i stand til å svare på, og tilby en alternativ løsning. Denne funksjonaliteten vil derfor øke brukeropplevelsen ApiBots brukere sitter med.

Akseptansekriterier
Gitt at API bruker sender spørring med spørsmålet “Miljøstasjon, jeg vil parkere, tømme søppel.” skal ApiBot returnere kategorier som forslag til ny spørring.

- Ved usikkerhet skal ApiBot foreslå mest sannsynlige kategorier det aktuelle spørsmålet er tilknyttet.

Akseptansetest:

1. Utfør POST API kall til https://api.apibot.no/test/message med parameteret “message”: “Miljøstasjon, jeg vil parkere, tømme søppel.”
    - Returverdi ApiBot: “Jeg er litt usikker på hva du vil frem til... Mente du et av disse alternativene?” categories array som inneholder navn og verdier.
2. Utfør POST API kall til https://api.apibot.no/message/chosen_category med parameteret “intent”: “recycling_get”
    - Returverdi**:** Liste over miljøstasjoner fra kommunens datasett.

7. **Som innbygger ønsker jeg mulighet til å bruke ApiBot via et brukergrensesnitt på eksisterende nettsider, slik at jeg kan interaktere med ApiBot.**

Beskrivelse:
Brukergrensesnittet skal kjøre oppå en eksisterende nettside som en utvidelse. Denne løsningen vil fungere som frontend av systemet, og vil være uavhengig av selve nettsiden det kjøres på.

MoSCoW: Should have

MoSCoW Begrunnelse:

For å kunne implementere et brukergrensesnitt hvor innbyggere kan interaktere med ApiBot, må ApiBots funksjonaliteter være tilstede. Uten funksjonalitet og krav til kvalitet, vil ikke ApiBot kunne produsere relevante og gode svar til innbyggere. Likevel erkjennes det at denne brukerhistorien vil ha stor verdi for produktet som en helhet, da det vil bli mulig å lansere systemet slik at de tiltenkte innbyggerne faktisk kan benytte ApiBot.
Akseptansekriterier: Gitt at innbyggere besøker en nettside ApiBot kjører på, skal innbyggere kunne interaktere med ApiBot gjennom en chat. - ApiBot skal kjøres oppå eksisterende nettsider. - ApiBot frontend skal i så stor grad som mulig følge wcag 2.0, samt andre kjente designstandarder, som øker brukernes tilgjengelighet og brukbarhet av applikasjonen.
Akseptansetest: 1. Gå til https://apibot.no/ hvor ApiBot er implementert og tilgjengelig i en utvidelse. 2. Åpne chatten nede til høyre. 3. Skriv "Jeg vil finne toalett i nærheten av meg." ● Returverdi**:** Et toalett fra kommunens datasett.

8. **Som innbygger ønsker jeg innledningsvis informasjon om hvordan jeg kan bruke ApiBot, slik at jeg enkelt kan komme i gang med å bruke ApiBot.**
Beskrivelse: Ved førstegangsbruk av ApiBot kan det være nødvendig med en innføring på hva ApiBot kan gjøre og hvordan nye innbyggere skal komme igang med å bruke den.
MoSCoW vurdering: Could have
MoSCoW begrunnelse: Ved å ha en introduksjonsguide for innbyggere, tillater ApiBot variasjon i innbyggers tekniske ferdigheter samt forståelse for ApiBots omfang.
Akseptansekriterier: Gitt at innbyggere initierer kontakt med ApiBot, skal innbygger få en guide om bruk av tjenesten. - Guiden inneholder en kort beskrivelse av hva ApiBot kan brukes til. - Guiden vises som ApiBots første melding i en sesjon med innbygger. - ApiBot skal kunne vise den samme introduksjonsguiden dersom brukeren spør om den ved å skrive "hjelp"
Akseptansetester: 1. Gå til https://apibot.no/ hvor ApiBot er implementert og tilgjengelig i en utvidelse. 2. Åpne chatten nede til høyre. ● Returverdi**:** Melding fra ApiBot som inneholder guide om bruk av ApiBot. 3. Skriv "hjelp" ● Returverdi**:** Melding fra ApiBot som inneholder guide om bruk av ApiBot.

9. **Som innbygger/API bruker ønsker jeg henvisning til forskjellige skjemaer og retningslinjer ut ifra temaet jeg stiller spørsmål rundt. (Byggetillatelse etc.)**

Beskrivelse: Basert på meningen i spørsmålet henvises innbygger til passende skjemaer, tillatelser og retningslinjer. Dette kan være byggetillatelseskjemaer eller andre skjemaer man typisk får gjennom kommunen.
MoSCoW vurdering: *Could Have*
MoSCoW begrunnelse: Funksjonaliteten er ikke sentral for alle innbyggere på samme måte som de andre brukerhistoriene med “Should Have” prioritet, fordi slike henvisninger ikke nødvendigvis omfatter alle innbyggere. Etter intervju med enhetsleder for servicetorget ved KRS kommune, ser vi at en stor andel(35%) henvendelser servicetorget får er relatert til byggesaker hvor det ofte er skjemaer inkludert. Selv om dette er den største brukergruppen, til tross for at de fortsatt er i mindretall totalt sett, blir denne funksjonaliteten definert som Could Have. Disse henvendelsene krever også ofte forståelse fra behandleren, som i enkelte tilfeller kan gjøre det utfordrende å generalisere forespørselen.
Akseptansekriterier Gitt at innbyggere/API bruker stiller et spørsmål om et eller flere skjemaer eller spørsmål som sannsynligvis er relatert til skjema(er), skal ApiBot gi henvisning til hvor man kan finne skjemaet.
Akseptansetest: 1. Gå til https://apibot.no/ hvor ApiBot er implementert og tilgjengelig i en utvidelse. 2. Åpne chatten nede til høyre. 3. Skriv “Jeg vil ha byggetillatelseskjema” Returverdi: Melding fra ApiBot: “Her har du vedlegg for byggetillatelse: $lenke”

10. **Som innbygger/API bruker ønsker jeg å dele min posisjon med ApiBot, slik at ApiBot også basert på min lokasjon kan gi meg relevante svar**
Beskrivelse: Ved en rekke tilfeller kan posisjonen til innbygger være relevant for hvilke svar innbygger ønsker. For eksempel kan spørsmål relatert til parkering og parker gi et mer presist svar for innbygger om innbyggers posisjon tas med i betraktning når ApiBot leter etter svar.
MoSCoW: Could have
MoSCoW Begrunnelse: Brukerhistorien foreslår funksjonalitet som for mange innbyggere kan være kjekk å ha, men vil ikke være relevant for alle innbyggere. I tillegg er vi, på nåværende tidspunkt, usikre på hvor mange innbyggere som ønsker denne funksjonaliteten.
Akseptansekriterier: Gitt at innbyggere besøker nettside ApiBot kjører på og stiller spørsmål, skal ApiBot ta innbyggers posisjon med i betraktning når den svarer, dersom innbygger har godtatt posisjonsdeling. - ApiBot skal be innbygger om tillatelse til å hente innbyggers GPS koordinater - ApiBot skal bruke brukerens posisjon til å svare på spørsmål der kunnskap om innbyggers posisjon er relevant for spørsmålet. - ApiBot i seg skal aldri lagre brukerens posisjon utover å prosessere den for å generere stedsbestemte svar.
Akseptansetest: 1. Gå til https://apibot.no/ hvor ApiBot er implementert og tilgjengelig i en utvidelse. 2. Åpne chatten nede til høyre.

3. Trykk “Ja” på “ønsker du å dele din posisjon for å forbedre svarene til ApiBot”?
4. Still spørsmålet “Jeg vil parkere nær meg.”

Returverdi fra ApiBot: Den nærmeste parkeringsplassen i nærheten av deg.

11. **Som innbygger ønsker jeg å få forslag fra ApiBot i et kart som er integrert i ApiBot frontend, slik at jeg enkelt kan se hvilke muligheter jeg har.**

Beskrivelse: Kartet skal implementeres i chatten på en slik måte at innbygger får oversikt over aktuelle muligheter et spørsmål kan ha.

MoSCoW: Could have

MoSCoW Begrunnelse: Brukerhistorien foreslår funksjonalitet som for mange innbyggere kan være kjekk å ha, men vil ikke være relevant for alle innbyggere. I tillegg er vi, på nåværende tidspunkt, usikre på hvor mange innbyggere som ønsker denne funksjonaliteten.

Akseptansekriterier:
Gitt at innbygger besøker nettside ApiBot kjører på og stiller spørsmål som omhandler lokasjoner, skal ApiBot svare innbygger med alternativer i tekstformat og kart.

- ApiBot skal be innbygger om tillatelse til å hente innbyggers GPS koordinater
- ApiBot skal kun dele innbyggers posisjon med kartrammeverket.
- Innbygger vil kun få kart som svar fra ApiBot dersom innbygger stiller spørsmål relatert til lokasjoner.
- Hvert svaralternativ fra ApiBot skal representeres som en markør i kartet.
- Når en markør blir trykket på, skal data om alternativet vises i kartet.

Akseptansetest:

1. Gå til “domenenavn” hvor ApiBot er implementert og tilgjengelig i en utvidelse.
2. Åpne chatten nede til høyre.
3. Trykk “Ja” på “ønsker du å dele din posisjon for å forbedre svarene til ApiBot”?
4. Still spørsmålet “Jeg vil parkere nær meg.”

Returverdi fra ApiBot: Kart med markører som representerer mulige parkeringsplasser.

12. **Som maskinlæringsansvarlig, ønsker jeg å ha et brukergrensesnitt hvor jeg kan se hvilke kategorier som er mest etterspurt og har minst treffsikkerhet, slik at jeg kan forbedre treffsikkerheten og kvaliteten på disse områdene.**

Beskrivelse:
Statistisk oversikt over populære kategorier, og spørsmål med lav treffsikkerhet vil være interessant for maskinlæringsansvarlig, fordi dette gir hen mulighet til å trene kategorier og spesifikke spørsmål som er mest relevant for innbyggere.
Denne løsningen vil kunne øke treffsikkerheten til ApiBot ettersom treneren vil kunne prioritere emner for opplæring.

MoSCoW vurdering: Could have

MoSCoW begrunnelse:

Denne brukerhistorien foreslår funksjonalitet som potensielt kan øke brukervennligheten til ApiBot mye, likevel vil ApiBot på grunn av ovennevnte Must Have og Should Have brukerhistorier, fungere uten denne brukerhistorien.
Akseptansekriterier Gitt at maskinlæringsansvarlig logger inn på “domenenavn”, skal hen få en oversikt over populære kategorier og spørsmål med lav treffsikkerhet. - Brukergrensesnittet til maskinlæringsansvarlige skal være tilgjengelig på et domene. - Kun maskinlæringsansvarlige skal ha tilgang til domenet. - Dataen på nettsiden skal være generert av forespørsler(spørsmål og spørsmåls metadata) fra innbyggere og API brukere. - Dataen skal lagres i en database.
Akseptansetest: 1. Gå til “domenenavn” hvor brukergrensesnittet for maskinlæringsansvarlige er tilgjengelig. 2. Logg inn med tildelt brukernavn og passord. 3. I oversikten du ser i brukergrensesnittet, finn den mest etterspurte kategorien, og kategorien med lavest treffsikkerhet. 4. Gå til tildelt lenke til Wit.ai. 5. Logg inn med tildelt brukernavn og passord. 6. Still treningsspørsmål relatert til den mest etterspurte kategorien. Resultalt: Økt treffsikkerhet på den mest etterspurte kategorien. 7. Still treingsspørsmål relatert til kategorien med lavest treffsikkerhet. Resultalt: Økt treffsikkerhet på kategorien med *opprinnelig* lavest treffsikkerhet.

13. **Som <u>innbygger</u> ønsker jeg å bidra til opplæring av ApiBot, ved å gi den tilbakemelding på om svaret er relevant eller ikke, for å gjøre ApiBot smartere.**
Beskrivelse: Dersom ApiBot er usikker på hvilket svar som er riktig, kan innbyggeren bidra til videre opplæring av ApiBot ved å gi tilbakemelding på om svaret er relevant eller ikke. Disse tilbakemeldingene skal komme i form som valg av kategorier som passer best til spørsmålet, eller om svaret er relevant (ja/nei). Jo “smartere” ApiBot blir, jo mer relevant vil svaralternativene være for innbyggere i situasjoner hvor ApiBot er usikker. Denne brukerhistorien må ses i sammenheng med brukerhistorie 12, da gjennom brukergrensesnittet til maskinlæringsansvarlig er nødvendigvig for å behandle forbedringstilbakemeldinger.
MoSCoW vurdering: Could have
MoSCoW begrunnelse: Kontinuerlig opplæring og forbedring av treffsikkerheten til ApiBot er en vesentlig del av prosjektet, derfor gjøres dette først og fremst av mennesker med dedikerte roller (maskinlæringsansvarlige). Likevel vil muligheten til å lære av innbyggere også være nyttig, men ikke kritisk for å utvikle et fungerende system.
Akseptansekriterier Gitt at innbygger stiller et spørsmål som ApiBot ikke kan svare helt sikkert på skal ApiBot informere om usikkerhet, avlevere antatt svar, og be innbygger velge en av X* oppgitte kategorier. - Ved usikkerhet skal ApiBot informere innbygger om usikkerheten i %.

- Ved usikkerhet under X%** skal ApiBot stille spørsmål til innbygger om hvilken kategori det stilte spørsmålet er mest relevant til.
    - Slike svar vil bli lagret i en database og gjennomgått av maskinlæringsansvarlig for godkjenning/underkjenning.
- Ved treffsikkerhet under X%** skal ApiBot avlevere et svar den antar er korrekt, for så å stille innbygger spørsmålet "Er svaret relevant eller ikke" for å kunne lete i en ny kategori, samt lagre denne statistikken til senere opplæring.

** Avhengiger av antall kategorier, men vil presentere de mest populære kategoriene.*
***Vil bli avgjort når vi forstår hvilke treffsikkerhetsprosenter som produserer korrekte svar.*

Akseptansetest:

1. Gå til "domenenavn" hvor ApiBot er implementert og tilgjengelig i en utvidelse.
2. Åpne chatten nede til høyre.
3. Skriv "Jeg vil bade, parkere, grille, sole meg, tømme søppel, og gå på do".

Returverdi fra ApiBot: "Jeg er kun 20% sikker på hva du spør om, kan du fortelle meg hvilken av de følgende kategoriene spørsmålet er mest relatert til? Parkering, tømme søppel.

4. Skriv "Parkering"

Returverdi fra ApiBot: "Takk, jeg vil nå lære av oppklaringen din. Du kan nå forsøke å stille ditt opprinnelige spørsmål på nytt med en annen formulering."

5. Skriv "Jeg vil parkere nær meg"

Returverdi fra ApiBot: "Du kan parkere på følgende steder nær deg, Sandens Parkeringshus, Slottet."

14. **!OBS! Utgått til fordel for #6.**
    Som <u>API bruker</u> ønsker jeg at ApiBot stiller meg oppfølgingsspørsmål relatert til mitt opprinnelige spørsmål, dersom ApiBot er usikker på hva jeg spør om, slik at jeg kan tydeliggjøre intensjonen min.

Beskrivelse:
Dersom API bruker stiller ApiBot et vanskelig eller uklart spørsmål, skal ApiBot fortsatt kunne gi reelle svar, enten ved hjelp av forslag, eller ledende spørsmål.

MoSCoW vurdering: Won't have

MoSCoW begrunnelse:
Brukerhistorien tar for seg et scenario der ApiBot ikke er fornøyd med graden av sikkerhet ved noen av svarene, men presenterer mulige alternativer som kan være relatert til meningen med spørsmålet.

Akseptansekriterier

- Gitt at API bruker stiller spørsmålet "hvor kan jeg parkere?", skal ApiBot klargjøre hva API bruker faktisk mener, ved å stille oppklarende spørsmål som "I hvilket område vil du parkere?" med svaralternativer.
- Ved usikkerhet skal ApiBot foreslå klargjørende svaralternativer som må velges, for å oppklare innbyggers intensjon. API brukers svaralternativ blir deretter behandlet som et ordinært spørsmål.

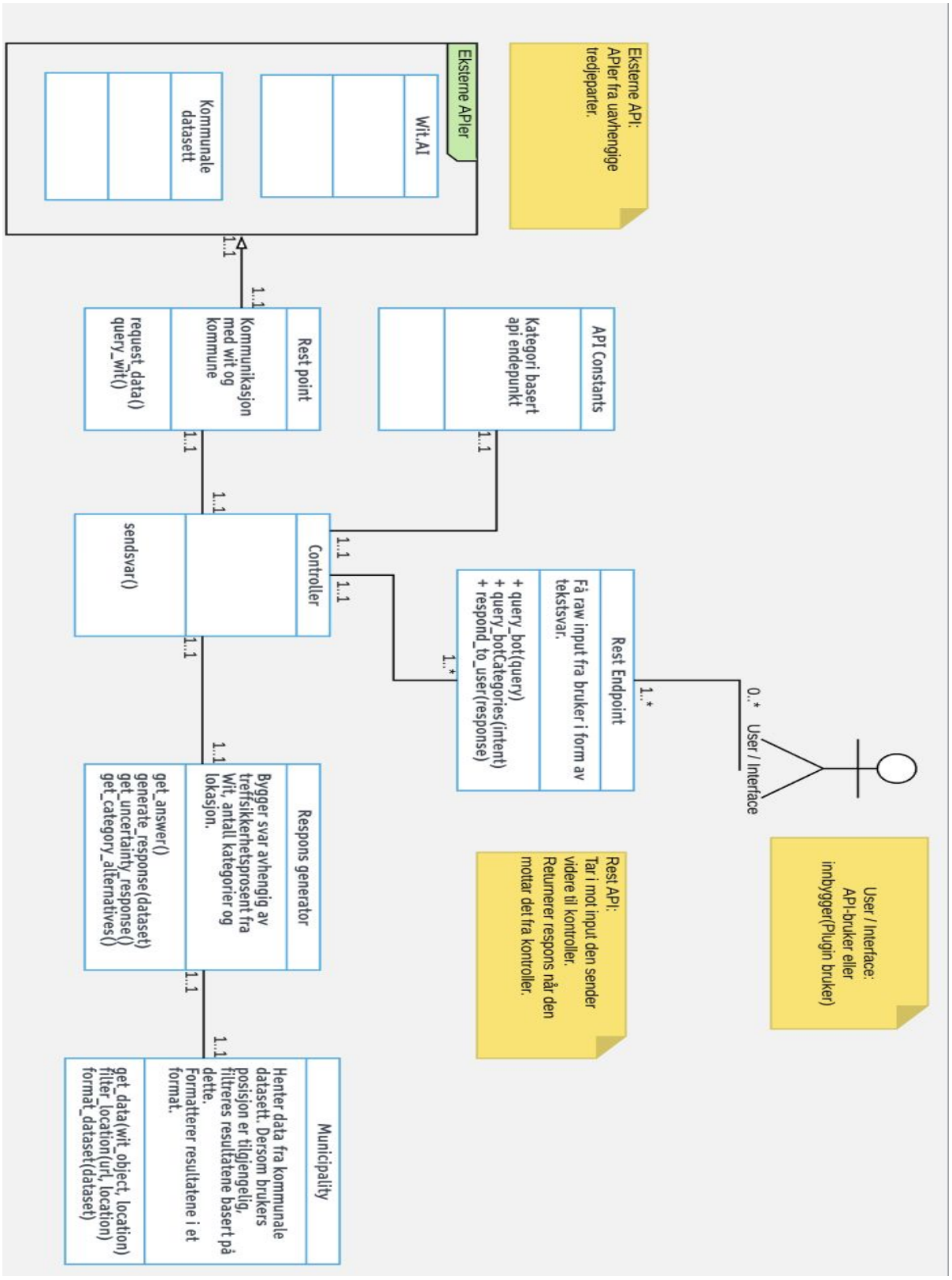
15. **Som innbygger ønsker jeg en mobilapplikasjon hvor jeg kan bruke ApiBot, slik at jeg enkelt tilgjengelig har en kilde til lokal informasjon.**
Beskrivelse: Mobilapplikasjon for iOS og android som implementerer chatbotten frontend, som bruker funksjonaliteter nevnt i tidligere brukerhistorier via API kall.
MoSCoW vurdering: Won't have
MoSCoW begrunnelse: Dette er en tjeneste vi ikke ønsker å bruke tid på i prosjektet, men det kan bli en tjeneste som implementeres på et senere tidspunkt, da arkitekturen vi foreslår for systemet tillater det. Brukerhistorien er ikke en del av omfanget i prosjektet. Brukerhistorien illustrerer hva systemet vi skal implementere ikke innebærer.
Akseptansekriterier: Ved bruk av en mobilapplikasjon, skal jeg ha mulighet til å chatte med ApiBot. - Applikasjonen følger anerkjente designprinsipper. - Applikasjonen skal være tilgjengelig på de to mest populære mobile plattformene iOS og Android.

Vedlegg 4. Ikke-funksjonelle krav

Ikke funksjonelt krav	Forklaring	Krav	Begrunnelse
Tilgjengelighet	Tilgjengelighetsgrad av apien, beskriver akseptable nivåer for "downtime" ved vedlikehold, feil og lignende	Selve APIen skal ha en oppetid så nært 100% som mulig. Ved nedgang av APIet skal feilen rettes så fort som mulig. Under vedlikehold, enten ved innføring av ny data og kategorier eller deployment av ny build skal nedetiden være under så kort som mulig.	I hvilken grad ApiBot må være tilgjengelig er uvisst, men på nåværende tidspunkt ser vi ikke behov for å avgrense tilgjengeligheten av systemet når det er lansert utenom oppdateringer og eventuelle rutinebaserte backup.
Treffisikkerhet	Graden av sikkerhet resultatet fra wit.api skal ha før forespørselen håndteres	- x% ett resultat fra query, - y% tre resultat fra query, <z% liste med resultater fra query	For å gi brukeren relevante svar til spørsmålene de stiller må det implementeres en grense på treffsikkerheten til intensjonen før forespørselen håndteres.
Universell utforming	Utforming av systemet slik at det er tilgjengelig for så mange som mulig	I utgangspunktet vil ApiBot kun være brukbar for alle som kan lese og skreve. Dette innebærer at brukergrensesnittdesignet må tilrettelegge for fargeblinde og svaksynte så langt det lar seg gjøre. Omfanget av prosjektet dekker ikke bruk av tale.	Kun tekst, ikke tale: Dekkes ikke av omfang.
Sikkerhet	Informasjonssikkerheten til systemet.	Høy grad av integritet på teknisk og organisatorisk plan blant utviklere og server ApiBot sine APIer befinner seg på.	ApiBot skal ikke lagre data om brukerne sine, og har heller ikke noe direkte ansvar for dataen som brukes(kommunedata aksesseres gjennom uavhengige APIer). Likevel skal ApiBot være en pålitelig informasjonskilde, og det er derfor vesentlig at maskinlæringen og chatbotten i seg selv ikke blir tuklet med av uvedkommende.
Maskinlæring	Kvalitetsikkring under opplæringsprossessen. Tilrettelegging for brukere under Aktiv Opplæring	Opplæring av ApiBot skal foretaes av oss som utviklere hvor vi opptrer som AI trenere. For utviklerne som skal lære opp APIBot vil det bli stilt krav til god forståelse for hvordan prossessen virker. Når	Ansvaret for trening og kategorisering av forespørsler må håndteres av personer med riktig kompetanse.

		brukere skal involveres i aktiv læring skal det foregå på en måte tilrettelagt av utviklerene som eliminerer eller begrenser muligheten for feil.	
Kapasitet	Antall maks brukere	Antall henvendelser KRS kommune fikk i 2017 var ca. 15000. Antall maks brukere ApiBot kan håndtere vil i utgangspunktet bli bestemt av hvilke begrensninger serveren ApiBot kjører på har. Wit.Ai(maskinlæringen) har ingen begrensinger	
Samsvar	Applikasjonen opperer i samsvar med reglementer og direktiver	Personopplysninger og sensitiv data håndteres i samsvar med personvernsreglemente t	Bruk og lagring av data må samsvare med relevante regler. I første omgang skal det ikke benyttes noe personlig data.
Responstid	Akseptabel responstid ved normal trafikk	ApiBot skal opprettholde en så god responstid som mulig slik at både utviklere og sluttbrukere skal få en god og rask kommunikasjon med systemet	Ettersom det siste leddet i ApiBot systemet er selve ApiBot er det viktig at vår del av løsningen opprettholder en god responstid slik at alle typer brukere ikke føler at systemet kjører tregt. Dermed vil det også være lettere for utviklere å feilsøke systemet ettersom ApiBot anses å være pålitelig
Lisensiering	Lisenser som dekker behovene for prosjektet	wit.ai bruker ingen bestemt type lisens, wit.ai kan benyttes til både kommersiel og privat bruk. Ved bruk av flere tredjeparts komponenter må lisensbruk revurderes.	Lisensiering er avhengig av lisensene som brukes av tredjepartskomponentene.
Avhengighet	Hvilke tredjeparter ApiBot er avhengig av for å fungere, evt. i hvilken grad ApiBot er avhengig av eksterne parter	ApiBot er avhengig av wit.ai for tekstanalyse, og data.norge.no for tilgang på data. Begge disse er kritisk for bruken av applikasjonen.	ApiBot er naturlig avhengig av datapunkter fra Kommunen den samarbeider med. I tillegg benytter ApiBot seg av wit.ai sin maskinlæring. ApiBot vil ikke fungere uten datapunkter og/eller wit.ai sin maskinlæring

Vedlegg 5. Klassediagram backend



Vedlegg 6. Funksjonsliste

	Funksjon	Kompleksitet	Type	Begrunnelse
Web-plugin | Front-end	Meldinger fra og til ApiBot i web-plugin	Simpel	Lese og skrive fra og til api-verktøyet	En chat som tillater en bruker å sende inn og motta meldinger til og fra ApiBot. Kompleksiteten for dette er satt til simpel, fordi det finnes flere open source rammeverk for meldingstjenester, i tillegg til at vi har erfaring med lignende funksjonalitet.
	Kartdata i web-plugin	Medium	Lese geo-data fra api'et og vise geo-data.	Bruker kan trenge kartdata relatert til ulike spørsmål. Denne kartdataen skal vises i chatten frontend. Kompleksiteten er satt til medium, da det krever å ta i bruk et rammeverk for kart(eks. Google maps) og implementere "noder" som representerer muligheter en bruker kan benytte seg av..
	Vedlegg i web-plugin	Medium	Lese vedlegg fra Api'et og presentere det.	Bruker trenger informasjon i form at filer og lignende. Kompleksitetsnivået er satt til medium da det kan kreve litt tid til å presentere vedlegg front-end. Dette er fordi vedleggene kan være av ulik type (url, pdf, o.l.).
	Nåværende posisjon i web-plugin	Simpel	Sende inn nåværende GEO-lokasjon på brukeren som sender inn melding	Benytte et rammeverk til å hente ut posisjonen til brukeren som sender inn spørsmål til ApiBot. Kompleksitetsnivået er satt til simpel da typiske rammeverk for dette er enkle å benytte, og gjør mye av arbeidet for oss.
Backend	Rest-Api	Simpel	Ruter-funksjon	En enkel ruter api, som tar imot forespørsler, og sender det videre til kalkulering innad i systemet. Gruppen har mye erfaring med å implementere slik funksjonalitet, og kan fra erfaring si at det tar lite tid, derfor er denne funksjonalitetens kompleksitet simpel.
	Sende/motta data til wit.ai	Simpel	Api send/retur	Denne funksjonen vil vidresende rå-tekst stringer til WIT.ai for å få ut meningen av det brukeren spurte om. Kompleksiteten er satt til simpel, fordi python biblioteket til wit.ai legger til rette for enkel kommunikasjon til og fra apien med et par linjer kode.
	Kalkulere spørring til Ekstern APi	Høy	Beregning	Ut ifra kategorier mottatt fra Wit.ai vil denne funksjonen kalkulere hvilke data systemet skal ta med seg videre til det eksterne APIet.

Backend				Kompleksitetsnivået til denne funksjonen er satt til høy fordi denne vil ha et stort antall mulige beregninger på ulike dataset som ApiBot vil ha tilgang til.
	Setningsbygger	Høy	Beregning	Genererer og returnerer en setning basert på hvilke data systemet får tilbake fra det eksterne APIet. Kompleksitetsnivået er satt til høy, ettersom antall mulige setninger potensielt kan være svært mange, grunnet de mange forskjellige datasettene som finnes.
	Send svar	Simpel		Sender den ferdige oppbygde setningen til bruker. Kompleksitetsnivået er satt til simpelt fordi funksjonen bare videresender det ferdige svaret til brukeren.
	Treffsikkerhetsavgjørelse	Simpel	Beregning	Ut ifra treffsikkerheten gitt fra Wit.ai, skal funksjonen finne ut om de kategoriene gitt har høy nok treffsikkerhet til at det skal bli gjort en videre spørring. Om treffsikkerheten er for lav, vil bruker få beskjed om dette, og spørsmålet slås ikke opp i det eksterne APIet stopper. Kompleksitetsnivået er satt til simpel, ettersom de tallene som beregnes er få, samt at det vil være fastsatte grenser for hvor bra treffsikkerheten må være.

Vedlegg 7. Funksjonelle krav

Funksjonelle krav	Forklaring	Input	Resultat / Output	Begrunnelse	Brukerhistorie
Ta i mot spørring	Tilgjengelig for kall for brukere og api-brukere	API kall	Svar på spørsmål	For at ApiBot skal kunne kommunisere med brukere og api-brukere må den kunne ta i mot spørringer	1,2,3,4,7,9,11
Kartdata	Ved svar som tillater kart-lokasjon, skal denne geo-dataen returneres.	Hvor er <objekt> i <område navn>?	Returnere geo-data som json format som kan benyttes til google maps eller andre tredjeparts-kartprogrammer.	Kartdata er en nyttig funksjon for ApiBot og dens brukere	1,7
Svare på spørring	En god standard er å generelt kunne svare på alle brukernes spørringer	Hei ApiBot, mitt navn er Pelle!	Hei på deg, Pelle. Du virker som en hyggelig fyr. Hva kan jeg hjelpe deg med i dag?	Del av hovedfunksjonene for bruker	1,2,3,4,7,9,11
Kan forklare når systemet ikke finner noen riktige svar	For å opprettholde "svar på spørring" standarden er det viktig at selv når systemet ikke finner mening i spørsmålet kan det fortsatt svare noe	"brrrrbrbr"	"Beklager, men jeg forstår ikke helt hva du sier. Her er noen ting du kan spørre meg om:"	For å opprettholde illusjonen om en levende samtale er det viktig at systemet alltid tibyr mulige løsninger eller fortsettelser til bruker	1,2,3,4,7,8,9,11
Kan lage vedlegg til svar	Svar som krever respons som inkluderer filer eller dokumenter skal inneholde denne filen	"Jeg trenger søknadsskjema for <X>"	"Her har du skjema for <X>. Er det noe mer jeg kan hjelpe med?"	Ved konkret spørring om skjema, direkte svar med vedlegg av skjema	11
Gi svar relatert til lokasjon	Svar basert på brukers posisjon	gps plassering og spørsmål	resultat i nærheten(Innenfor en viss radius)	For at dataen som returneres er relevant for bruker kan søk utføres basert på brukerens plassering.	12

Kan gi tekstmelding til svar	Ved vanlige spørsmål, skal et svar med tekst returneres	Hei	Hallo		1,2,3,4,7,9,11
Kan gi flere alternativer til svar	Ved forespørsel eller usikkerhet, skal ApiBot komme med forslag som kan hjelpe brukeren og snevre inn hva hen lurer på. Dette i form av alternativer som bot'en vurderer selv.	Vis meg alle museum i Kristiansand	Returnerer en liste over samtlige museum i Kristiansand	Nyttig dersom brukere ønsker mer informasjon på en gang, eller spørsmålet ikke var tydelig nok	1,2,3,4,7,8,9

Vedlegg 8. Hendelsetabell

Klasser -> Hendelser	User / Interface	Rest Endpoint	Controller	Rest Point	API Constants	Respons Generator	Municipality
Stille spørsmål til ApiBot	X						
Motta Spørsmål fra bruker		X	X				
Sjekke relevans				X			
Motta relevans			X				
Konstruere spørring			X	X	X		
Sende spørring for data				X			X
Behandle data			X				
Konstruer svar			X			X	X
Send svar til bruker		X					
Motta svar fra ApiBot	X						
Tjeneste sjekk (Error)		*X*	*X*	X			

Vedlegg 9. kvalitetsdokument

Figurliste

1. Introduksjon

Hensikten med dette dokumentet er å beskrive rutiner, prosesser og standarder alle utviklere i utviklingsteamet har måttet forholde seg til under utviklingen av ApiBot. Dette dokumentet er først og fremst til for alle deltakere av utviklingsteamet, da det tydeliggjør retningslinjer og krav som stilles under utviklingen av ApiBot. Kvalitetsdokumentet tar for seg alle rutiner og prosesser som vil være med på å øke kvaliteten på prosessen og produktet.

2. Testplan

En testplan er en detaljert plan over objektene, ressursene og rutinene som inngår i testingen av et system. Derfor vil denne testplanen ta for seg hvilke testmetodikker og testtyper som har inngått i testingen av ApiBot.

2.1. “Behaviour driven development”

Behaviour-driven development (BDD) er en programvareutviklingsprosess som kombinererer prinsipper fra både test-driven development(TDD), domain-driven design og object-oriented analysis and design for å forbedre samarbeidet mellom programvare-utviklere og forretningsutviklere. BDD benyttes for å implementere enhetstester. (BDD, 2018)

Stegene for å implementere kode ved å bruke BDD:

- Beskriver metoden eller klassens oppførsler i en eller flere situasjoner.
- Definerer en test for enheten det gjelder.
- Kjør alle tester og kontroller at den nye testen feiler.
- Implementerer koden til enheten slik at enheten passerer enhetstesten.

- Kjør alle tester for å validere at den nye testen fungerer isolert, og ikke påvirker andre deler av systemet på en utenkt, negativ måte.
- Refaktorerer kode.

Den fundamentale forskjellen på BDD og TDD, er at før testene blir skrevet, kreves det at det skrives scenarioer for hver enkelt test som tilhører èn enkelt brukerhistorie. Disse scenarioene formuleres på en slik måte at programvare-utviklerne kan bruke de som en "mal" for test skriving, samtidig som at de skal være forståelig for forretningsutviklerne. På denne måten fungerer disse scenarioene som en kommunikasjons-bro mellom de forskjellige rollene innad prosjektet. Derfor er det viktig at disse scenarioene forholder seg til èn enkelt standard, for å forhindre forvirring mellom medlemmer av prosjektet.

Et typisk eksempel på brukerhistorie scenario i ApiBot:
"Som innbygger ønsker jeg innledningsvis informasjon om hvordan jeg kan bruke ApiBot, slik at jeg enkelt kan komme i gang med å bruke ApiBot."

Scenario 1: Bruker kobler til ApiBot

Gitt at bruker trenger innledningsvis informasjon fra ApiBot
Når bruker kobles til ApiBot
Da skal ApiBot returnere en velkomst-streng

Scenario 2: Bruker ber om hjelp fra ApiBot

Gitt at bruker trenger hjelp eller informasjon fra ApiBot
Når bruker ber om hjelp til ApiBot
Da skal ApiBot returnere hjelp og informasjon

Denne fremgangsmåten har vært nyttig for oss, da den har ført til at enhver person som leser scenarioene har fått en tydelig oversikt over hva hver test skal gjøre på det laveste nivået, før det brukes ressurser på å implementere test eller enhet.

2.1.1. Rammeverk:

Behave er et python rammeverk som tilrettelegger for BDD, via naturlig språk direkte i koden. Behave lar oss merke forskjellige steg i testingen. Stegene over merkes med @given, @when, @then og utføres i en bestemt rekkefølge(Behave, 2018) Behave benytter Gherkin (Gherkin, 2018) for å definere beskrivelser av funksjonalitet, såkalte features. Hver feature har

en serie med scenarier som benytter seg av funksjonene som skal testet. Disse kjøres sekvensielt via de spesifiserte stegene med informasjon gitt i selve teststeget eller via en tabell.

```
Scenario Outline: User asks for addresses outside Norway

  Given the parser receives request with location
  When  the parser performs a search with <location>
  Then  the parser should handle addresses outside Norway by returning None

  Examples:
  |location|
  |{"location": [{"suggested": true,"confidence": 0.93785,"value": "Berlin","type": "value"}]}|
  |{"location": [{"suggested": true,"confidence": 0.93785,"value": "LONDON","type": "value"}]}|
  |{"location": [{"suggested": true,"confidence": 0.93785,"value": "China","type": "value"}]}|
```

Figur 1 - Behave scenario

I dette tilfellet kjører testen med verdiene for hver rad i tabellen i steget som benytter <location>.
Scenarioer spesifiseres i bestemte filer i features mappen.

Jest er et JavaScript testverktøy som lar en enkelt skrive og kjøre tester av JavaScript applikasjoner *(Jest, 01 Mai 2018)* . Jest, i likhet med React er utviklet og i bruk av Facebook, noe som gjorde det til en klar kombinasjon sammen med våre web-rammeverk. I tillegg tar vi i bruk Enzyme ved siden av Jest. Enzyme er utviklet av Airbnb og lar utviklere teste React-komponenter ved å ta “snapshots” i form av JSON datafiler som representerer hvordan et “ideelt” komponent skal se ut. Dermed kan Enzyme kontrollere at endringer i koden ikke påvirker komponenter negativt. Jest støtter også liknende funksjonalitet, men Enzyme stiller i vår mening sterkere ved å ha eksistert lengre på markedet og kan dermed tilby flere verktøy for denne kontrolleringen av komponenter.

2. 2. Overordnede testmetoder:

- Black-Box:
  Krever ingen kunnskap av testeren om selve systemet bak.
  Forventet utfall av et bestemt scenario med gitte input verdier.
- White-box:
  Baserer testene på bakgrunn av implementeringen i modulene.

Kunnskap og tilgang til koden.

Verdier som testes basert på aksepterte verdier i koden.

- Alle lovlige og ulovlige verdier, både valide og invalide, testes.

- Gray-box:

  Kombinasjon black-box og white-box

2.3. Unit-Tester

Unit testene i ApiBot skal utføres ved hjelp av Behaviour driven development, der man definerer en forventet oppførsel under et bestemt scenario.

Fremgangsmåte:

- Definer ønsket oppførsel
- Skriv test som bekrefter oppførsel
- Få testen til å feile
- Implementer enheten slik at testen passerer
- Refaktorer

Enhetstestene tar for seg testing av systemet på det laveste nivået i den forstand at den kun tester en enkel metode eller klasse. Da det er oss selv som har implementert både enhetstestene og enheten samt besitter mye kunnskap om systemet, har vi benyttet white-box som en overordnet test metode for enhetstesting.

2.4. Systemtester

Systemtestene tar for seg det helhetlige systemet, ofte testet gjennom black-box metoden av uavhengige tester. Da det er oss selv som har utført systemtester benyttet vi en kombinasjon mellom white og black-box, fordi vi har tilgang og forståelse for koden, men ønsker å kontrollere resultatet opp mot forventet resultat ved bestemt input. Det er verdt å nevne at det ikke kreves noen forkunnskaper for å utføre ulike systemtester av ApiBot, til tross for at vi har det.

2.5. Akseptansetester

Akseptansetester er en formalisering av programvarens ønskede funksjonaliteter. Når en akseptansetest testes undersøkes det i hvilken grad akseptansekriteriene tilfredsstiller brukerbehov, funksjonelle krav og forretningsprosesser. Akseptansetestene med sine respektive akseptansekriterier har blitt formelt testet mot slutten av prosjektet i form av FAT møter med oppdragsgiver. Da hver akseptansetest er beskrevet i konkrete detaljer og har tilhørende akseptansekriterier, vil akseptansetesting være en form for grey-box testing.

2.6. Regresjonstesting

Tester om eksisterende kode forholder seg uendret i atferd til tross refaktorering eller tillegg av interaksjon med interface(r). I utgangspunktet vil vi følge BDD også her, slik at vi får alle fordelene som tidligere nevnt BDD bringer med seg. Det vil si at før en modul blir refaktorert eller får nye interface, skal regresjonstesten implementeres. («Regression testing», 2018)
Det er typisk at alle eller flere av disse testene blir kjørt hver gang man implementerer en ny regresjonstest. Å "re-runne" alle disse testene hver gang vil etterhvert som man får mange tester potensielt være en tidkrevende prosess. Derfor finnes det flere ulike teknikker som forsøker å løse tids-utfordringen.
Retest all: Ved enhver ny regresjonstest testes alle andre, tidligere implementerte, regresjonstester.
Regression test selection: Velger ut noen tidligere implementerte regresjonstester dersom kostnaden av dette er mindre enn retest all teknikken.
Test case prioritization: Rangerer alle regresjonstestene i prioritert rekkefølge slik at de viktigste blir utført først. Hensikten med dette er å la de viktigste regresjonstestene avdekke flest og størst feil først.
Da vi ikke har hatt noen utfordringer med å kjøre alle testene ofte relatert til ytelse, har vi benyttet "retest all' metoden. Om dette i fremtidig utvikling blir en svært tidkrevende prosess kan vi anbefale å bytte ut denne med andre teknikker som er mer hensiktsmessig. Nøyaktig hvilken teknikk dette kan være, vil avhenge av blant annet applikasjonens behov.

2.7. Integrasjonstesting

Integrasjonstesting kombinerer flere enhetstester og tester disse som en gruppe. Hensikten med integrasjonstester er å avdekke feil/defekter i interfaces og interaksjoner mellom moduler. (Integration Testing, 01.02.18) Også her var det naturlig å anvende en testdreven metode. Integrasjonstester tar utgangspunkt i modeller og dokumenter utarbeidet i designfasen som forklarer dynamikken i systemet. Mer presist har det vært nødvendig å se på hvordan systemets komponenter skulle samhandle med hverandre. Enhver modul som inngår i en integrasjonstest må på forhånd være enhetstestet, dette er for å forsikre at alle modulene isolert fungerer som de skal. Det finnes to typiske teknikker for integrasjonstesting, "bottom up" og "top down".

Bottom-up

Bottum-up teknikken kan ses på som en utvidelse av enhetstestene fordi den fokuserer mye på modulenes utgående avhengigheter, altså hvilke effekt en modul har på andre. Bottom-Up tar utgangspunkt i moduler som ligger lavt i klassehierarkiet, og mimikerer andre klasser, som kanskje ikke finnes enda, høyere opp i hierarkiet.

Top down

Top-down teknikken tester systemet på en overordnet måte ved å vurdere om funksjonaliteter en modul samhandler med fungerer på tiltenkt måte. Top-down tester høytstående moduler i klassehierarkiet først, for så å teste seg nedover mot lavtstående moduler.

Avhengig av hvilke deler av systemet som har blitt testet og hvor i klassehierarkiet modulene befinner seg, har vi valgt en av disse teknikkene for integrasjonstesting. Eksempelvis har vi tidlig i prosjektet benyttet bottom-up metoden for å teste enkeltstående moduler som har vært avhengig av andre moduler som ikke var implementert på tidspunktet.

2.8. Struktur for anvendelse av testplan

Enhetstesting

- Unit-tester utføres etter BDD/TDD prinsippet.
- Ønsket atferd defineres og test skrives med den hensikt at den skal feile. Gjør den ikke det må scenarioet redefineres. *Dette punktet er ikke gjeldende fra uke 7 som en følge av adopsjon av TDD*.
- Etter at testen har feilet opprettes og kodes enheten slik at testen passerer.
- Dersom testen passerer refaktoreres koden etter behov.

Integrasjonstesting

- Integrasjonstest utføres black-box eller white-box tester.
- Etter at to eller flere enheter som samhandler er ferdig implementert, skal koblingen mellom disse testes. Formålet er å avdekke potensielle feil og/eller mangler i koblingen.

Regresjonstesting

- Regresjonstestingen skal ta for seg endringer som potensielt kan forgreine seg oppover i systemet.
- Målet er å teste alle enheter som påvirkes av en enkelt endring.

Systemtesting

- Tester hele systemet.

- Black box testing. Input og resultat kontrollerer opp mot et forventet resultat.

Akseptansetester

- Tester spesifikke akseptansetester med sine respektive akseptansekriterier.
- Kontrollere at viktig funksjonalitet fra brukerhistoriene er implementert.

Resultat anvendelse av testplan

Ved å gjennom prosjektet anvende denne testplanen har vi implementert totalt 53 tester. Av disse er nærmere 20% integrasjonstester, og 20% regresjonstester, mens de resterende 60% er negative eller positive enhetstester. Av systemtester har vi i stor grad anvendt Ad hoc testing for og tidlig teste at systemet har/ikke har vesentlige defekter. *(Ad hoc testing, 01 Mai 2018)*

Formell akseptansetesting har blitt gjennomført gjennom FAT møter sammen med produkteier.

2.9. Automatisering av tester

Automatisering av tester kjennetegnes ved at utvikler som kjører tester lager scripts eller benytter eksisterende programvare for å kjøre et sett med forhåndsbestemte tester automatisk. Det er typisk at slike automatiske tester blir kjørt når oppdateringer pushes til repository. Om en eller flere av disse testene feiler skal scriptene eller programmet som kjører testene rapportere tilbake til de aktuelle utviklerne. Ved å benytte automatisering av tester har vi potensielt sørget for at nye oppdateringer ikke inneholdt feil vi ellers ikke ville oppdaget på et senere, mer sårbart tidspunkt.

Vi har benyttet VSTS' Testplan for å automatisk kjøre enhets- og integrasjonstester ved enhver oppdatering i development eller master grenen.

3. Kodestandarder

Ved å følge Pep 8 Standarden får koden en gjennomgående stil, noe som reduserer hvor tydelig det er at koden skrives av flere personer. (PEP 8, 2018)

3.1. Navngivning

- Variabelnavn
    - Engelsk
    - Ord separert med understrek
    - Forklarende variabler
- Metodenavn
    - Engelsk
    - Ord separert med understrek

    - Forklarende metodenavn
- Klasser/Moduler
    - Ingen bruk av understrek eller punktum i modul navn
    - Bruk ètt beskrivende ord i modulnavn
    - Modulnavn i små bokstaver
    - Klassenavn som forklarer innholdet i klassen

3.2 Defensiv programmering

Defensiv programmering er en form for kvalitetssikring hvor man designer kode i den grad at man forsøker å forutse alle mulige måter en slutt bruker kan gjøre feil. Dette gjør at man kan ha klare løsninger dersom feil input skulle forekomme, som resulterer i at kvaliteten på koden økes, ettersom antall bugs blir redusert siden de håndteres og programmet vil oppføre seg mer forutsigbart. (Mariani, 2016)
Ved å implementere feilhåndtering som tar høyde for feilaktige parametere i funksjoner, som for eksempel en streng på steder hvor funksjonen forventer et tall, har vi potensielt unngått at systemet krasjer under kjøring.

3.4 ESLint

ESLint er en JavaScript linter vi har valgt å tilføye vårt prosjekt. ESLint fungerer som en plugin i programmene vi bruker for å skrive JavaScript. ESLint sjekker dermed vår kode opp mot rigide standarder og gir klar beskjed dersom noe ikke er som det skal. Dette hindrer både feil, og gjør koden standardisert og lesbar for hele teamet. (ESLint, 2018)

4. Kode kontroll

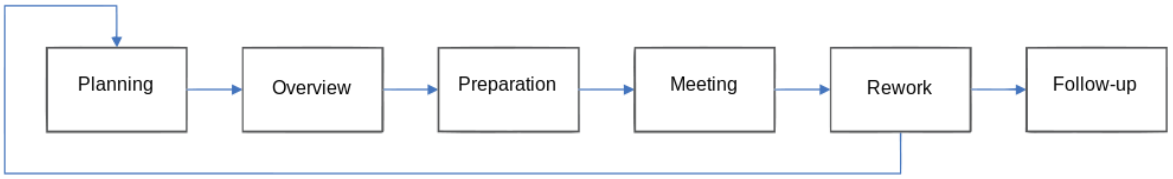
Kode kontroll (code review) er en systematisk gjennomgang av skrevet kildekode gjort av to eller flere utviklere, hvor hensikt er å avdekke svakheter, feil, ineffektivitet og lignende som kan ha blitt oversett i utviklingsfasen.

Kode kontroll består i hovedsak av to kategorier, som er formel kode kontroll, og ‘lettvekt’ kode kontroll (Kolawa, 2007).

4.1. Formell kode kontroll

En mer formell gjennomgang av kildekode, som består av detaljerte prosesser og forskjellige faser som deltakerne av kode kontrollen må forholde seg til. Et eksempel på formell kode

kontroll er 'Fagan inspection'., som består av seks overordnede operasjoner som følges («Fagan inspection», 2017):



Figur 2 - Fagan Inspection

4.2. Lettvekt kode kontroll

Mindre formell gjennomgang av kode, hvor flere forskjellige former for 'peer review' kan benyttes. For å nevne noen:

- Over-skulderen (Over-the-shoulder)
  Når kildekode er klar for å bli vurdert, finner man en kvalifisert kollega som kan kontrollere koden, mens utvikleren selv forklarer dem hva og hvorfor de har gjort som de har gjort.
- Sende rundt email (Email-pass-around)
  Når kildekode er klar for å bli vurdert, sendes denne rundt til forskjellige kvalifiserte kolleger over mail, hvor disse gjennomgåes så snart deres arbeidsflyt tillater det.
- Par-programmering (Pair programming)
  Utviklere jobber side om side på den samme kildekoden, som indirekte gjør at de sjekker hverandres arbeid, og derfor praktiserer kode kontroll.
  Verktøy assistert kode kontroll(Tool-assisted code review) benytter programvare for å forenkle kodekontrollprosessen. Dette gjør at man enkelt spore og se forslag til endring, eller faktiske endringer gjort av kolleger, som gjør man lett kan jobbe asynkront *(Swartflear, 2018)*.

4.3. Kode kontroll struktur

Malen for vår bruk av kode kontroll er definert fra en artikkel fra IBM der de tar fram de beste praksisene etter å ha studert "peer code control" fra 6000 programmerere fra over 100 bedrifter *(IBM, 2011)*.
Vi har brukt denne guiden for å definere kode kontroll for vårt prosjekt og vil følge følgende punkter:

1. Se på under 100 linjer i gangen.
2. Ta god tid ved kontroll av kode, og jobb maks 1 time i strekk.

3. Ved bruk av ekstern kode, skal kilde ligge vedlagt.
4. Sett mål for kodekontroll og bruk sjekkliste fra kodestandard.
5. Verifiser endret kode ved å teste koden underveis.
6. Kode må ha blitt kontrollert før den sendes til development.

4.4. Kode kontroll ApiBot

Under utviklingen av ApiBot har vi tatt i bruk en lettvekt kodekontroll, hvor utviklere gjør en 'pull request' når de har pushet ferdig kode som skal slås sammen med development eller master grenen. Dette vil si at før denne koden kan slås sammen, skal andre utviklere gjennomgå koden, for å forsikre seg om at arbeidet er av kvalitet og ikke inneholder seriøse mangler eller feil. Denne måten å drive kode kontroll på vil falle under 'verktøy assistert kode kontroll", da vi vil bruke Team Services sin pull request funksjon.

5. Feil og bugs

Fordi vi har valgt VSTS har vi hatt all teknologien nødvendig for å rapportere om bugs i ApiBot. Det vi derimot ikke får fra VSTS er rutiner for hvordan vi burde håndtere bugs. Derfor skal vi i dette kapittelet forklare hvordan bugs har blitt håndtert gjennom prosjektet.

5.1. Innmelding

Ved feil eller bugs **skal** det rapporteres via VSTS. Ved innmelding av feil eller bugs skal det legges ved følgende informasjon om feilen eller buggen.

- De ulike stegene man må gjøre for å gjenskape problemet.
- Akseptansekriterier som er kriterier som skal være innfridd for at innmeldingen skal kunne bli huket av som løst.
- Beskrivelse av problemet, altså system info: Hvilket operativsystem mm. Og annet som kan være relevant for å forstå problemet og hvor problemet ligger.

5.2. Utmelding

Ved utmelding av en feil eller en bug, skal akseptansetestene være innfridd. Det skal også skrives en eller flere tester som gjenskaper problemet og tester feilen eller buggen. Dette er for å forhindre at feilen oppstår på et senere tidspunkt og at feilen er løst.

6. Design Standard

I form av design standarder har vi valgt å følge WCAG 2.0 retningslinjene for å gjøre vår applikasjon mest mulig tilgjengelig for mennesker med forskjellige funksjonshemminger. *(WCAG 2.0, 2008)* I forhold til funksjonshemminger, har vi derimot ikke tatt hensyn til personer som er blind. Dette grunnet tidspress og kompleksitet av en slik oppgave.

7. Mappestruktur

Mappe og filstruktur vil følge den anbefalte strukturen fra “Hitchhikers Guide to Python” («The Hitchhiker’s Guide to Python!», 2016).

Under følger en oversikt over mappestrukturen og forklaring av oppsettet.

```
        README.rst
        LICENSE
        requirements.txt
app/
        __init__.py
        core.py
        helpers.py
    components/
            __init__.py
server/
        __init__.py
        core.py

docs/
        conf.py
        index.rst
    api/
        get.yml
tests/
        test_basic.py
        test_advanced.py
```

Figur 3 - Mappestruktur

Mappene **app** og **server** inneholder prosjektets hoveddeler. Disse mappene vil inneholde logikken bak både API endepunktet og ApiBots interne logikk.

LICENSE filen vil inneholde eventuelle lovlige lisenser denne koden skal dekkes under.

Requirements filen inneholder oversikt over alle avhengigheter nødvendige for å arbeide på prosjektet eller gjennomføre tester.

Docs mappen inneholder alle dokumentasjonsfiler som kreves for å bruke dette prosjektet, samt hvordan prosjektet og applikasjonen kan brukes videre. I tillegg ønsker vi å legge filer for API endepunkt dokumentasjon i denne mappen.

Tests mappen inneholder alle forskjellige tester som utføres i prosjektet.

8. Dokumentasjonsstruktur API

Generering av dokumentasjon for API brukere vil bli generert gjennom Flasgger utvidelsen til Python. Flassger lar oss enkelt koble sammen Flasks ruter sammen med dokumentasjon

skrevet i separate Yaml filer. Resultatet blir en enkel og minimalistisk referanse til evt dokumentasjon utvikleren ønsker å skrive til sine metoder.

```
@app.route('/message')
@swag_from('../docs/api/message.yml')
def message():
       return jsonify({
       'body': 'Hello there!'
})
```

Figur 4 Dokumentasjonsstruktur

I den separate Yaml filen blir selve teksten og mer detaljert dokumentasjon lagret.

```
Example endpoint returning a list of colors by palette

---
parameters:
  - name: palette
    in: path
    type: string
    enum: ['all', 'rgb', 'cmyk']
    required: true
    default: all
definitions:
  Palette:
    type: object
    properties:
      palette_name:
        type: array
        items:
          $ref: '#/definitions/Color'
  Color:
    type: string
responses:
  200:
    description: A list of colors (may be filtered by palette)
    schema:
      $ref: '#/definitions/Palette'
    examples:
      rgb: ['red', 'green', 'blue']
```

Figur 5 Yaml dokumentasjonsfil

Resultatet blir en automatisk generert nettside som presenterer en dynamisk dokumentasjon av endepunktene til ApiBot. Brukere kan også teste endepunktene live på denne nettsiden dersom de ønsker.

9. Kontinuerlig integrasjon

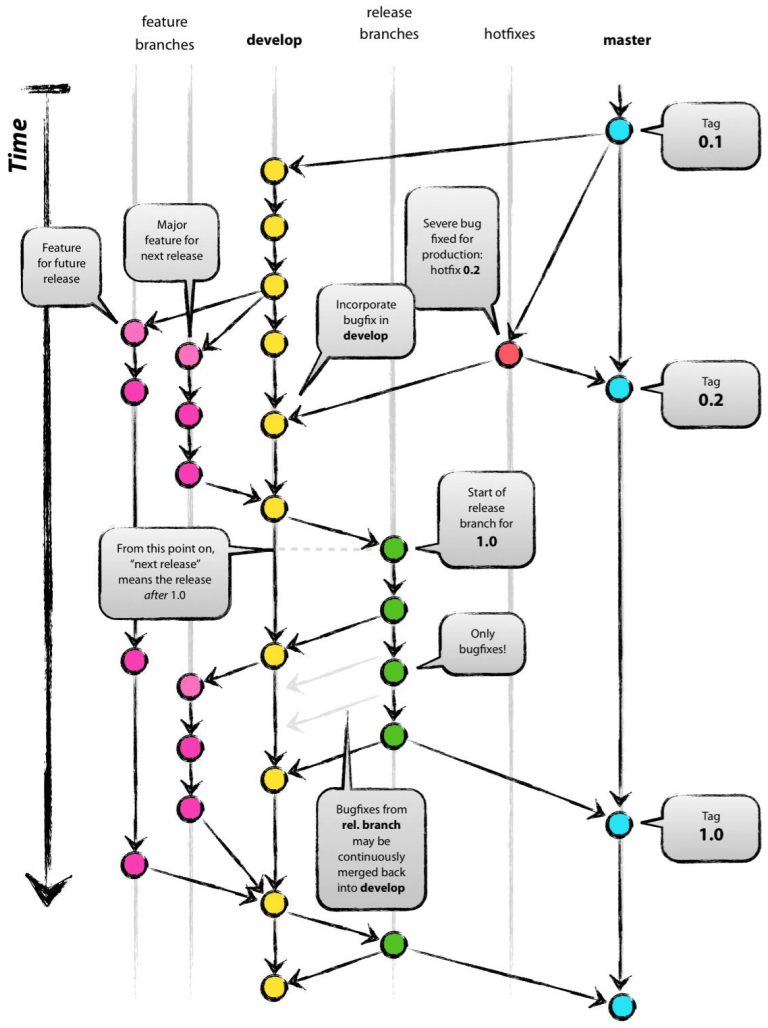
Dersom en utvikler ønsker å endre på en kodebase, vil han eller hun kopiere kodebasen over i sin egen branch. Dette refereres til ofte i versjonskontroll ved at utvikleren “sjekker ut” koden over i sin egen versjon. Utviklerens kode er nå separert fra kildekoden, og endringer som skjer mot kildekoden via andre utviklere øker forskjellene mellom utviklerens branch og kildekoden.

Jo lengre en branch er separert fra kildekoden, jo større problemer og risiko kan utvikleren møte dersom han vil integrere sin kode tilbake i hovedgrenen. Dersom forskjellene mellom utviklerens branch og hovedgrenen i kildekoden er blitt for store, vil utvikleren havne i en situasjon som ofte refereres til som “integration hell”. Utvikleren er nå i en situasjon hvor integrasjonen av endringene han har utført vil ta mer tid enn det tok å gjøre endringene i første omgang.

Kontinuerlig integrasjon forsøker å omgå dette problemet ved å integrere kode så tidlig og ofte som mulig, og dermed unngå å havne i en slik lås. Ofte integreres kode daglig eller etter oppretting av en ny funksjon. Ved å forhindre å havne i lås er målet å spare både kostnader og tid i utviklingen. I henhold til BDD skal enhetstester gjennomføres før utviklere kan publisere kode, og spesifikke integrasjonstester kjøres ofte automatisk på kodebasen etter at koden har blitt lagt til (Continuous integration, 2017).

9.1. Grenmodell

En grenmodell avgjør hvordan arbeidet med implementasjonen av prosjektet skal organiseres og struktureres.



(Vincent Driessen, 2010)

Figur 6 Grenmodell.

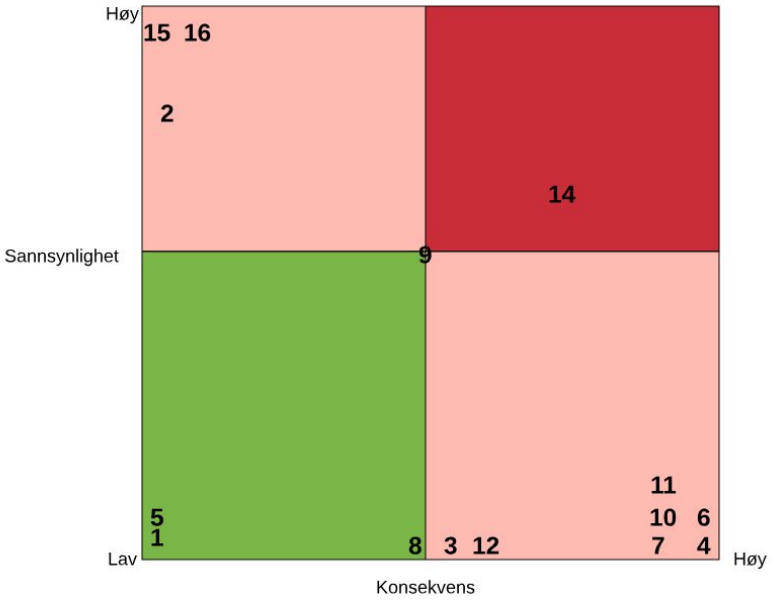
Vi ønsker å følge den kjente branchmodellen "A successful git branching model" for å organisere kodebasen vår.

Når nye funksjonaliteter skal implementeres skal det arbeidet enkapsuleres i en egen branch slik at det ikke påvirker resten av kodebasen. Når en funksjonalitet er implementert med godkjente tester og kodekontroll skal den integreres i "Development" branchen. Denne branchen skal til enhver tid reflekterte den nyeste versjonen av systemet. Hensikten med denne branchen er å integrere all kode som er implementert for så å kjøre ulike tester som potensielt kan avdekke bugs. Når development branchen når et stabilt punkt med ønsket funksjonalitet, skal den integreres i en release branch. Master er branchen hovedbranchen med kildekode som er i en produksjonsklar tilstand. Om bugs oppstår i denne hovedbranchen vil løsninger for slike bugs bli implementert i en egen hotfix branch som så vil integreres i master.

10. Risikoanalyse

Da vi har gjennomført et større prosjekt har det vært nødvendig at vi utførte risikovurderinger av flere interne og eksterne elementer som potensielt har kunnet påvirke prosjektet.
Vi utførte risikovurderinger for å kartlegge og vurdere alle farer og problemer ved å vurdere risikoen knyttet til et gitt element i form av å gradere konsekvens og sannsynlighet.
Da risiko ikke nødvendigvis bare omhandler umiddelbare og fysiske farer, har vi valgt å bruke arbeidstilsynets standard mal for risikovurdering *(Risikovurdering, 10.03.2018)*. Kjernen i denne risikovurderingen har gitt analysen vår svar på hva som kunne gått galt, hva som kunne blitt gjort for å forhindre dette og hva vi kunne gjort for å redusere konsekvenser dersom risikoen inntreffer.
Når konsekvens og sannsynlighet ble fastsatt, ble disse multiplisert til en total risikovurdering.
Resultatet av risikoanalysen har blitt ført opp i prosjektets risikomatrise.

Risikoanalyse oppsummert



Figur 7 - Risikoanalyse oppsummert
Figuren illustrerer hvor høy sannsynlighet og konsekvens alle elementene i risikomatrisen har. Elementene er representert som nummer, hvor de faktiske elementene har tilhørende nummer i risikomatrisen.

11. Involvering av oppdragsgiver

Et tiltak for å sørge for kontinuerlig forbedring av produksjonen, samt at systemet utvikles på en slik måte at det tilfredsstiller kunde og sluttbrukere, har vært å aktivt og kontinuerlig involvere produkteier og aktuelle sluttbrukere. Produkteier har blitt involvert ved enhver sentral avgjørelse, invitert til sprint planning og review møter, slik at hun har kunnet påvirke retningen systemet utvikles i, og eventuelt foreslå eller kreve endringer. I tillegg har det blitt rapportert til produkteier hver uke om aktivitet og progresjon opp mot brukerhistorier og milepæler, samt tidsforbruk og utfordringer.

12. Litteraturliste

11 proven practices for more effective, efficient peer code review. (2011, januar 25), hentet fra http://www.ibm.com/developerworks/rational/library/11-proven-practices-for-peer-review/

A successfull Git Branching model, Vincent Driessen (05.01.2010), hentet fra: http://nvie.com/posts/a-successful-git-branching-model/.

“Acceptance Testing.” Software Testing Fundamentals (blog), January 1, 2011. http://softwaretestingfundamentals.com/acceptance-testing/

Ad hoc testing, sist sett 01 Mai 2018), hentet fra: https://en.wikipedia.org/wiki/Ad_hoc_testing

Behavior-driven development. (2018, januar 30), hentet fra https://en.wikipedia.org/w/index.php?title=Behavior-driven_development&oldid=823200701

Continuous integration. (2017, desember 4). I Wikipedia. Hentet fra https://en.wikipedia.org/w/index.php?title=Continuous_integration&oldid=813690984

Defensive programming. (2018, February 28). Hentet 08. Mars 2018, fra https://en.wikipedia.org/wiki/Defensive_programming

Fagan inspection. (2017, september 12). I Wikipedia. Hentet fra https://en.wikipedia.org/w/index.php?title=Fagan_inspection&oldid=800256959

Integration testing. (2017, november 7). I *Wikipedia*. Hentet fra https://en.wikipedia.org/w/index.php?title=Integration_testing&oldid=809172659

Jest, sist sett 01/05/2018, hentet fra: https://facebook.github.io/jest/

Kolawa, A. (2007). Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press. Hentet fra http://www.wiley.com/WileyCDA/WileyTitle/productCd-0470042125.html

PEP 8 -- Style Guide for Python Code, sist sett 1. februar 2018, hentet fra https://www.python.org/dev/peps/pep-0008/

Regression testing. (2018, januar 3). I Wikipedia. Hentet fra https://en.wikipedia.org/w/index.php?title=Regression_testing&oldid=818451231

Risikovurdering, sist sett 1. februar 2018, hentet fra https://www.arbeidstilsynet.no/hms/risikovurdering/

The Hitchhiker’s Guide to Python! — The Hitchhiker’s Guide to Python, sist sett 1. februar 2018 hentet fra http://docs.python-guide.org/en/latest/

tutorialspoint.com. Software Testing - Types of Testing, sist sett: 1. februar 2018, hentet fra https://www.tutorialspoint.com/software_testing/software_testing_types.htm

Vincent Driessen (2010, Januar, 05), A successful Git Branching model, hentet fra: http://nvie.com/posts/a-successful-git-branching-model/.

Welcome to behave! — behave 1.2.5 documentation, sist sett 1. februar 2018, Hentet fra http://pythonhosted.org/behave/index.html

What Is Code Review? Hentet 1. februar 2018, fra https://smartbear.com/learn/code-review/what-is-code-review/

Mariani, D. (2016, Desember 25). The Art of Defensive Programming – Web Engineering Vox – Medium. Hentet fra https://medium.com/web-engineering-vox/the-art-of-defensive-programming-6789a9743ed4

WCAG 2.0, 11.12.2008, fra (n.d.). Retrieved April 09, 2018, from https://www.w3.org/TR/WCAG20/

ESLint - Pluggable JavaScript linter, sist sett 6 April, 2018, hentet , fra https://eslint.org/

Vedlegg 10. Risikomatrise

Nummer	Risiko	Konsekvens(1-4)	Sannsynlighet (1-4)	Risikovurdering	Tiltak for å forebygge	Tiltak redusere skade
1	Evry går konkurs	1	1	1		
2	Lettere sykdom	1	3	3		Holde seg hjemme til man er frisk, slik at man ikke smitter andre. Alt arbeid et individ har skal være tilgjengelig for alle, slik at andre enkelt kan overta. Dette innebærer også tilganger til verktøy som for eksempel versjonskontroll og vsts
3	Alvorlig Sykdom	2	1	2		Alt arbeid et individ har skal være tilgjengelig for alle, slik at andre enkelt kan overta. Dette innebærer også tilganger til verktøy som for eksempel versjonskontroll og vsts
4	Eksterne verktøy/servere blir slått av eller gjort utligjengelig for oss	4	1	4	Holde oss oppdatert om verktøyene og serverløsningene vi bruker	Backups og alternative løsninger I utgangspunktet benytte kjente/populære verktøy
5	Ødelagt arbeidspc	1	1	1	Sunn fornuft, varsomhet og godt nettvett	Reparerer pc, kjøp/lån pc
6	Manglende kompetanse i prosjekt	4	1	4	Bygge kompetanse	Bygge kompetanse
7	Splid i arbeidsgruppe	4	1	4	Teambuilding Respekt for hverandre Sprint retrospect Daily scrum	Aktuelle parter skværer opp med hjelp fra scrum-master eller emneansvarlig(Hallgeir) om splid med Hallgeir, skal Stig kontaktes
8	Ikke svar/oppfølging fra oppdragsgiver	2	1	2	Opprettholde kontinuerlig konktat med produkteier	Om kontakten brytes, vil vi forsøke å gjenopprette kontakten direkte med produkteier, og eventuelt få hjelp av instituttet

9	Manglende respons fra tredjeparter(Kommune, brukere)	2	2	4	Opprettholde kontakt med enhetsleder servicetorvet ved å være pågående på en slik måte som oppfattes naturlig og høflig	Gjennopprette kontakt ved å høflig mase
10	Datasett blir utilgjengelig	4	1	4		Genere datasett selv. Finne andre datasett
11	Uklart i overgang fra analyse/design til implementering	4	1	4	Hyppige leveranser som kvalitetssikres opp mot analyse og design	Feedback fra brukere og produkteier
12	Misforståelser mellom produkteier og team	3	1	3	Jevn kontakt med produkteier angående system/brukerkrav, design og implementering av applikasjon, avklare potentsielle misforståelser raskt	Klargjøre misforståelser med produkteier, eventuelt gjøre endringer i prosessen og utviklingen
13	Tilgang til arbeidslokaler endres	2	1	2		Bruke alternative lokaler som grupperom og lignende
14	Analyse/design modellerer/kartlegger ikke det ønskede systemet	2	3	6	Grundige/klare diskusjoner i teamet og med produkteier som forsikrer felles forståelse. Nøye utførelse av arbeid med forankring i vitenskapen	Klargjøre den aktuelle arbeidsoppgavens hensikt med team og produkteier iht. systemkrav
15	Uforutsette personlige hendelser (Tannlege, skade, legebesøk osv)	1	4	4	Planlegge private æfferer utenfor arbeidsstid så langt det lar seg gjøre.	Alt arbeid et individ har skal være tilgjengelig for alle, slik at andre enkelt kan overta. Dette innebærer også tilganger til verktøy som for eksempel versjonskontroll og vsts
16	Fulltidsansettelse under prosjektgjennomføring	1	4	4		Tilrettelegge for arbeid etter ordinær arbeidstid
17	Eksamensperiode IS-305	1	4	4	Ingen tiltak	Overføre arbeid til neste sprint som ikke ble ferdigstilt grunnet manglende tilstedeværelse under eksamensperioden

Vedlegg 11. FAT1 referat

Brukerhistorie	**Status**	**Kommentar**
1.	Godkjent	
2.	Ikke godkjent	Kjøring av “try it out” funksjonaliteten i dokumentasjonen fungerer ikke. Parameteret location er oppgitt som “required”, mens det i realiteten er valgfritt.
3.	Ikke godkjent	Legg til skjermbilder i maskinlæringsguide
4.	Godkjent	
5.	Godkjent	
6.	Godkjent	
7.	Ikke godkjent	- “Annerkjente design standarder” - Hva betyr dette? - Kommuner er ekstremt strenge i form av tilgjengelighet, wcag 2.0 er en standard som kan bidra til å løse dette. - “Send” knapp mangler. - Chatten burde være mulig å lukke.
8.	Ikke godkjent	- Bot må ikke være ufin/morsomheter. Ha standardsvar. - Pass på at apibot svarer på de spørsmål vi gir som eksempel. “Hjelp” funksjonen gir nå et svar som man ikke kan spørre om, fordi den ikke er trent optimalt på dette.
9.	Godkjent	
10.	Ikke godkjent	Implementert på en slik måte at den skaper feilaktig oppførsel i front-end når man ikke godkjenner stedsdeling.
11.	Ikke godkjent	Markørene må bedre styles

Vedlegg 12. Ukeplan

Sprint	Uke	Arbeidsområde	Notat	Milepæler	Resultat	Endring i ukeplan
Pre	2	Definere prosjekt Utvikle fremdriftsplan Oppstart Analyse	Planlegge	- Prosjektomfang definert - Fremdriftsplan utarbeidet - Ukeplan laget	Alle milepæler oppnådd i sprinten.	
1	3, 4	Analyse Valg av utviklingsverktøy Design Opplæring innen utviklingsverktøy	Utviklingsverktøy beskriver alle former for platformer, arbeidsverktøy og versjonskontroll	- Utført intervju - Brukerroller definert - Brukerhistorier med akseptansekriterier utarbeidet - Rike bilder laget - Funksjonsliste laget - Klassediagram laget - Hendelsestabell laget - Sekvensdiagram laget - Flowchart laget - Tilstandsdiagram laget - Funksjonsliste utarbeidet - Funksjonelle og ikke funksjonelle krav utarbeidet - Datasett valgt til trening - Utviklingsverktøy valgt	Alle milepæler oppnådd i sprinten, med unntak av brukerhistorier. Avventer møte med PO for aksept/revidering brukerhistorier.	Opplæring innen utviklingsverktøy startet 24.01 da backlog for sprint 1 er tom

2	5, 6	Risikovurdering Revidere brukerhistorier Etablere kvalitetsdokument/plan Opplæring innen utviklingsverktøy og Stavangers datasett Finne eksisterende relevant kode (“frameworks”) Utforme maskinlæringsguide for maskinlæringsansvarlige Oppsett API verktøy Styringskomiteemøte	Kvalitetsdokument tar for seg alle rutiner og prosesser som øker eller forsikre kvaliteten på utviklingsprosessen og systemet	- Valgt automatiseringsverktøy for testing - Aksept fra produkteier ang. brukerhistorier med akseptansekriterier - Gjennomført risikovurdering og risikomatrise - Rutiner for håndtering av nye risikoer utarbeidet - Testplan lagt - Plan for kontinuerlig integrasjon - Kodestandard utarbeidet - Maskinlæringsguide utarbeidet - API server satt opp - Struktur for API dokumentasjon utarbeidet - Planlagt styringskomittemøte - Rapportstruktur vedtatt	Alle milepæler oppnådd i sprinten, med unntak av aksept fra PO angående brukerhistorier.	Opsett API verktøy flyttet til sprint 2 fordi vi har prioritert analyse og design i sprint 1 29.01 - La til risikovurdering da dette er hensiktsmessig for å vurdere sannsynlighet og konsekvens på potensielle fallgruver 29.01 - La til styringskomiteemøter da dette ikke var tatt høyde for i planen 31.01 - La til revidering av brukerhistorier basert på tilbakemelding fra produkteier og Espen Limi 01.02 - Endret "Sette opp treningsrammeverk til ApiBot" Til "Utforme maskinlæringsguide for maskinlæringsansvarlige", fordi det er mer presist iht. intensjonen til arbeidsoppgaven.

3	7, 8	Sprint backlog	Utvikling av applikasjon	- Gjennomført første styringskomiteemøte - Intregrert Wit.ai og ApiBot - Opprette forbindelse med ApiBot - Gjøre spørring til ApiBot.no og motta svar - Stille konkret spørsmål via spørring til ApiBot.no og motta treffsikkerhetsprosent fra Wit.ai - Implementert struktur for å legge til kategorier - Lagt til to kategorier i API konstanter - Implementert brukerhistorie #1, 2 - Rammeverk for kontinuerlig integrering implementer - Rutiner for automatisering av tester iverksatt - Simpel prototype av det som er implementert i sprinten	Alle milepæler oppnådd i sprinten. Obs! Enkelte klasser relatert til brukerhistorie #1 vil refaktoreres og utvides senere i prosjektet når vi har utført flere tester med ulike dataer. Foreløpig er det implementert en "simpel" versjon i den forstand at ApiBot svarer relevant på spørsmålene den er godt trent på.	Tillegg: "Simpel prototype av det som er implementert i sprinten" - Brukes til review, for å hente aksept eller endringsforslag fra PO.
4	9, 10	Sprint backlog	Utvikling av applikasjon	- Trent ApiBot på to kategorier slik at den svarer intelligent på spørsmål relevant til den trente dataen. Med intelligent menes det at ApiBot svarer logisk på det den blir spurt om. Eksempelvis vil spørsmålet "Hvor kan jeg parkere på Lund" generere svar relatert til hvor du kan parkere på Lund. - ApiBot gir svaralternativer(kategorier) dersom treffsikkerheten på spørsmålet er for lav - Implemenetert brukerhistorier #3, 4, 5, 6, 7 - Alle Must Have og Should Have brukerhistorier implementert - Metodikk dokumentert i rapport. - Kvalitetssikring dokumert i rapport - Analyse dokumentert i rapport. - Web applikasjon for dokumentasjon og bruk av plugin. - ApiBot front-end chat implementert som plugin	Brukerhistorie 3,4 og 5 er ferdig implementert gjennom valgte rammeverk. Alle milepæler med unntak av kvalitetssirking dokumentasjon i rapport er oppnådd. De sistnevnte milepælene er under arbeid, og vil derfor bli flyttet til neste sprint dersom de ikke fullføres på tilfreddstillende måte innen sprinten.	Endring: Svaralternativer i form av kategorier. Endring: ApiBot front-end chat implementert" da disse funksjonalitetene dreier seg om brukerhistorie #7, som implementeres denne sprinten. Endring: Fjernet milepæl relatert til prototype fordi web applikasjonen som er implementert fungerer som en prototype.

5	12	Sprint backlog	Utvikling av applikasjon	- ApiBot front-end åpner chat med bruksguide til innbyggere - ApiBot informerer om at den er usikker når treffsikkerheten på svaret er under *TBD*%. - ApiBot svarer med vedlegg der det er logisk. - Implementert brukerhistorie #8, #9, #10 og #11 - Innbyggere kan dele sin GPS-posisjon med ApiBot. - ApiBot bruker innbyggeres GPS-posisjon i utregningen av svar, dersom det passer spørsmålet. - ApiBot åpner et kart i chatten med forslag til svar dersom det passer spørsmålet.	Alle milepæler med unntak av brukerhistorie 9 og 11 oppnådd.	Endring: Uke 11 utgår til fordel for IS-305 eksamen Endring: fjernet "- ApiBot har mulighet til å lære av innbyggere" pga. endring i prioritering av brukerhistorier i samråd med PO. Endring: Lagt til "Konfiguere CI build automatiske tester slik at den kjører tester implementert vha. behave modul", da ved å bruke behave rammeverket i Python må vi konfiguere build testing annerledes. Endring: Flyttet analyse og kvalitetssikring dokumentasjon i rapport hit. Da det ikke er ferdig. Endring: Flyttet mer arbeid til denne sprinten da sprint 6 inneholdt for mye arbeid vs. ressurser

6	13, 14	Ferdigstille kode Rapport		- Applikasjon ferdigstilt - Redegjørelse for valg i prosjektet formulert i rapport. - Ferdigsstille brukerhistorie #9 og #11. - Gjennomføre FAT møte med PO. - Analyse dokumentert i rapport - Kvalitetsikring dokumentert i rapport - Design dokumentert i rapport - Prosjektgjennomføring dokumentert i rapport	Applikasjonen er i stor grad ferdigstilt, men har feil og bugs på enkelte områder nevnt i FAT referat. Valg i prosjektet og prosjektgjennomføring har i stor grad blitt dokumentert i rapport, men da ikke alle valg er ferdig ønsker vi ikke å flagge denne milepælen som oppnådd. Brukerhistorie #9 ferdigsstilt. Brukerhistorie #11 er i stor grad ferdigstilt, men skal utbedres iht. FAT referat. FAT gjennomført. Analyse og design i stor grad dokumentert i rapport.	Endring: Flyttet "redegjørelse for sentrale valg formulert i rapport" til denne sprinten da det er enklere å fomulere når vi har gjort enda fler sentrale valg. Endring: Fjernet utviklingsoppgaver til fordel for rapportarbeid
7	15, 16	Kvalitetssikre system og rapport		- Gjennomført styringskomiteemøte - Utbedret applikasjon på samtlige punkter nevnt i FAT referat - Applikasjon ferdigstilt - Gjennomført FAT v2 - Prosjektgjennomføring dokumentert i rapport. - Gjennomført møter med Stig angående rapport	Samtlige milepæler oppnådd	
8	17, 18	Kvalitetssikre system og rapport		- Rapport ferdigstilt		

Vedlegg 13. Referat styringskomiteemøte

Styringskomiteemøte 1 - 15.02.2018

Deltakere:

Bachelorgruppe: Christian Moen, Erlend Thorvik, Erlend Wiklem, Sindre Grønstøl Haugeland, Martin Nenseth, Mathias Hartveit

Veileder: Stig Bjørnar Nordheim

Produkteier: Merethe Sjøberg

Gjennomføring:

13.00: **Styringskomitemøte starter**

- Presentasjon i henhold til agenda (se vedlegg)

13.30 **Presentasjon ferdig. Tilbakemeldinger og spørsmålsrunde**

13.50 **Styringskomite møte ferdig**

Referat tilbakemeldinger og spørsmålsrunde:

Produkteier : Snakket med Espen om at de vet lite om hva som er poenget i det daglige arbeid. Espen vet ikke om det er begynt å kode, brukerhistoriene var ikke ferdig før denne uken

Bachelorgruppe: Vi tar høyde for endringen, og de er aldri ferdig i den forstand at de er revidert. Endringene skjer når vi ser nye behov og blir revidert underveis, selv om selve brukerhistoriene var ferdig i sprint 1.

Produkteier: Det er viktig at jeg blir involvert i arbeidet med brukerhistorier. Vi har et satt produkt og jeg forventer at det ikke skal være noen endringer i brukerhistorier.

Hvis det blir endring i brukerhistorier som også vil si produktet , skal jeg (som produkteier) være med på denne diskusjonen og klar over dette.
Jeg og Espen må involveres mer, da vi er eiere og må ha mer status.
Vi har ikke fått innkalling til flere statusmøter etter jeg arrangerte det første statusmøtet.
Dere kan ikke velge selv hva som skal prioriteres, da dere ikke selv er produkteier.
Vi må også ha en oversikt over risikoer, vite hva som er kritisk og kan tippe lasset.
Ønsker demo og visning av funksjonalitet etterhvert.
Veileder(Stig**):** Dere må i samråd med produkteier avtale faste møter framover
Jeg har litt kalde føtter, da mye analyse og lite kode så langt er lite smidig.

Veileder: Finnes det lignende løsninger under utvikling i andre kommuner?
BachelorGruppe: Nei, men vi fant tilsvarende andre tjenester for andre ting

Veileder: Hvem har bidratt til brukerhistorier?
BachelorGruppe: Merethe, espen har fått de tilsendt og intervjuet av kommunen.

Veileder: Hva anser vi som største risiko:
BachelorGruppe: At offentlig data blir utilgjengelig

Veileder: Dette er studentene sin ide, hva er evry sin motivasjon til å bruke tid på prosjektet?
Produkteier: Vi har fokus på studenter og det å ha gode medarbeidere, og evry ansetter på bakgrunn av arbeid som utføres. Så Evry er ute etter nye arbeidstakere og man kan plukke fra de som får jobbe hos Evry.

Referat slutt.

Styringskomiteemøte 2 - 08.03.2018

Deltakere:

Erlend Thorvik, Sindre Haugeland, Mathias Hartveit, Martin Nenseth, Christian Moen
Espen Limi, Merethe Sjøberg, Stig Nordheim

Agenda:

1. Kommentarer til forrige referat

    - Gjennomført

2. Oppfølging av saker fra forrige møte

    - Tiltak fra forrige møte presentert

3. Status iht. plan

    - Status presentert

4. Demo

    - Demo av ny funksjonalitet gjennomført

5. Planen videre

    - Endringer i forhold til plan diskutert

    - Flytting av milepæler

6. Tilbakemeldinger / diskusjon

Besluttninger:

- Logge feil/error/bugs
- Gjennomgang av alle akseptansetester(F.A.T) med produkteiere etter påske
- Ukentlig møte med Stig etter påske relatert til rapport og eksamen

Tiltak:

Hva	**Hvem**
Loggføre feil/error/bugs	Christian
Utføre F.A.T	Alle
Ukentlig møte med Stig etter påske relatert til rapport og eksamen	Alle

Styringskomiteemøte 3 - 10.04.2018

Deltakere:

Erlend Thorvik, Sindre Haugeland, Mathias Hartveit, Martin Nenseth, Christian Moen
Espen Limi, Merethe Sjøberg, Stig Nordheim

Agenda:

1. Kommentarer til forrige referat
    - Kommentarer presentert
2. Oppfølging av saker fra forrige møte
    - Tiltak og tiltaks status fra forrige møte presentert
3. Fremgang i prosjektet
    - Fremgang presentert
4. Demo
    - Demo av ny funksjonalitet presentert
5. Planen videre
    - Planen videre presentert
6. Tilbakemeldinger / diskusjon

Besluttninger:

- Dersom fremtidige bugs oppstår skal det implementeres tester som fremprovoserer buggen.
- Rapporten skal evalueres av Stig gjennom tre iterasjoner. Første iterasjon tar for seg de første fem kapitlene. Andre iterasjon tar for seg resten av kapitlene. I tredje iterasjon skal helheten, med endringer fra de to første iterasjonene, evalueres.

Tiltak:

Hva	**Hvem**
Implementere tester for bugs	Alle
Evaluer rapport	Alle

Vedlegg 14. Maskinlæringsguide

Denne guiden inneholder definisjoner og instrukser relevant til de personene som skal trene språkgjenkjennings rammeverket som brukes i ApiBot.

Tilgang

For å aksessere maskinlæringsrammeverket går du til https://wit.ai/Cmoen11/ApiBot/entities og logger inn med ditt tildelte brukernavn og passord.

Intensjoner

En intensjon i sammenheng med maskinlæring, vil være det ønskede resultatet en sluttbruker vil ha tak i av en sammensatt setning.

Eksempler på hvordan intensjon skal innhentes fra spørsmål:
"Hei, jeg er i Kristiansand og lurer på hvor jeg kan finne forskjellige kunstmuseum i sentrum?"
Intensjon: get_museum
"Jeg er på ferie og trenger parkeringsplass nær tollbodgaten"
Intensjon: get_parking

Entiteter

En entitet vil være et nøkkelord eller kategori som blir tatt i bruk for å finne intensjonen i den sammensatte setningen.
"Hvor kan jeg parkere el-bilen min mellom 0800 og 1200 i sentrum, helst i parkeringshus"
Intensjon: get_parking
Entiteter: wit/datetime: 0800 - 1200, wit/location: sentrum,
wit/local_search_query: parkeringshus, el-bil(ladestasjon)

Bruk av forhåndsdefinerte intensjoner og entiteter:

Siden wit.ai allerede tilbyr en rekke forhåndsdefinerte entiteter og intensjoner skal disse brukes dersom det er passende. Fordelen med disse er at de trenes av den kollektive bruken av wit.ai

Retningslinjer for maskinglæringsansvarlig

For at fremtidige maskinglæringsansvarlige skal kunne trene ApiBot i fremtiden, må de følge visse retningslinjer. Retningslinjene skal følges i så stor grad som mulig, for å forsikre seg om

at treningen som ApiBot får holder høy kvalitet. Regler for maskinlæringsansvarlige gjelder også for maskinlæringsutviklere.

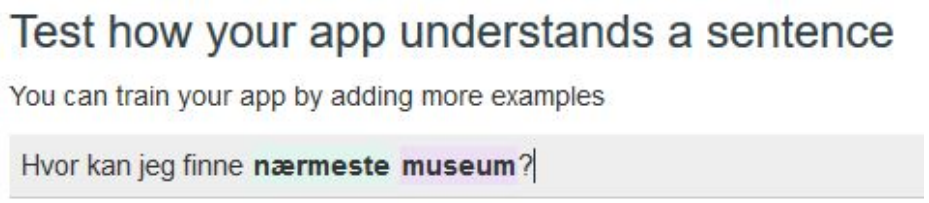
Dette innebærer:

- Tekst som inneholder former for diskriminering, støtende ord, rasisme, ærekrenking, bakvaskelse eller lignende, skal ikke bli godkjent som gyldig og derfor ikke trenes på.
- Tekst som ikke inneholder eksisterende kategorier skal ikke godkjennes. Maskinlæringsansvarlig kan derimot opplyse utviklere om henvendelsene om disse forekommer hyppig, dersom dette kan utbedres.
- Tekst som inneholder kategorier, men ellers ikke inneholder noen form mening skal ikke godkjennes. Feks: “ajsd inmopjisuf parkering anklsjdnadn”
- Tekst som inneholder skrivefeil kan godkjennes.
- Forøvrig gjelder norsk lov.

Intensjoner

Hvis det eksisterer data som kan anses som ‘nyttig’, kan dette bli omgjort til en intensjon i Wit. Dette kan være ting som vær, temperatur, veier og lignende. Skulle utviklere få tilgang til andre datasett, kan det derfor være nødvendig å opprette nye intensjoner.

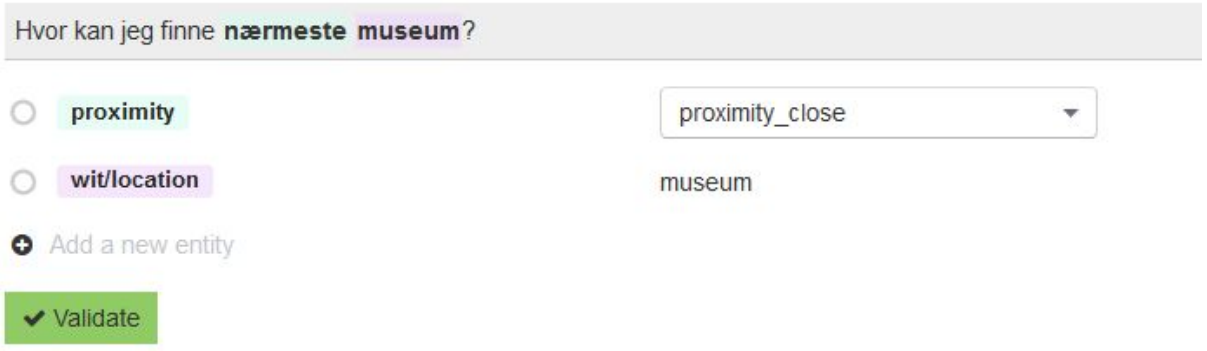
Nye intensjoner kan defineres på to måter. Først kan det defineres fra dashbordet til wit.ai.



Figur 7 - Maskinlærerammeverk tren med eksempel

Eksempel på ny intensjon

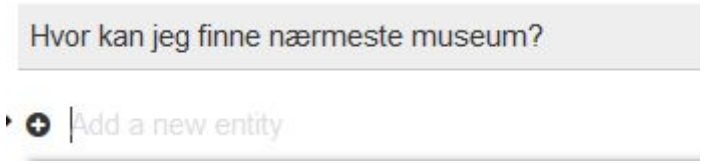
Dersom wit finner eksisterende entiteter i setningen vises de under input feltet.



Figur 8 - Maskinlærerammeverk entiteter

Dersom disse entitetene ikke stemmer må de fjernes ved å trykke på “X” tegnet til venstre for entiteten.

I eksempelet defineres det en intensjonen for å finne museer i nærområdet.
Intensjonen i setningen definerers ved å trykke på “+” tegnet under inputfeltet.



Alle intensjonene i applikasjonen ligger i en samlegruppe kalt “intent”.
Skriv “intent” og velg fra menyen.

Når intent entiteten er valgt kan det legges til nye intensjoner ved å trykke “Value” og spesifisere navnet på den nye intensjonen.

Ved å trykke på validate legges den nye intensjonen til i “intent” entiteten.
Wit vil deretter begynne å trene på den nye intensjonen. Treffsikkerheten øker ettersom flere forespørsler blir validert.

Den andre måten å legge til nye intensjoner på bruker spørringer som allerede er mottatt av wit ved hjelp av innboksen.

Innboksen inneholder alle spørringene som wit.ai applikasjonen har mottatt, og som ikke har blitt validert.

Følg stegene over med spørringen som inneholder den nye intensjonen.

Treffsikkerheten kan forbedres ved å legge til forskjellige spørringer med samme intensjon.

Entiteter

Om en entitet kan benyttes for å 'identifisere' en intensjon, vil dette være en kandidat for en ny entitet. I en setning som: "Hvor kan jeg finne nærmeste bensinstasjon?", kan både *nærmeste, bensinstasjon,* være potensielle entiteter, fordi de alle samme kan kobles til en intensjon om å få tak i en lokasjon.

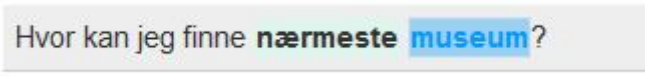
Konstanter

Hver konstant skal være en spørring knyttet opp mot de forskjellige datasettene brukt av ApiBot. Konstanter brukt i kall vil være basert på hvilke intensjoner og entiteter wit ai returnerer til ApiBot når en bruker stiller ApiBot et spørsmål. Navngivingen av konstanter vil være på engelsk, for å overholde normale konvensjoner.

Kategorier

For at wit skal kunne gjenkjenne spørringer som inneholder entiteter fra flere forskjellige intensjoner, brukes en egen gruppe for kategorisering av de forskjellige intensjonene. Dette gjøres ved å bruke et "Keyword search" på spørringen den mottar og treff legges til i returobjektet. Når det legges til en ny intent er det viktig at den også legges til i kategoriene. Det gjøres ved å markere et nøkkelord i spørringen og knytte det opp mot en entitet i "categories" entiteten.

For å legge til en ny kategori markeres nøkkelordet.



Den nye entiteten legges til som en "Value" i "categories" gruppen.

Når den nye kategorien valideres legges den til i “categories”.

Treffsikkerheten kan forbedres ved å legge til synonymer for de forskjellige kategoriene.

Vedlegg 15. Fremdriftsplan for Interessenter

Metode

Vi ser verdien av å respondere effektivt på endringer, prioritere fungerende programvare over tunge organisatoriske prosesser/kravspeccer, samt. involvere kunde og sluttbruker. Istedenfor å fokusere mye ressurser på dokumentasjon som kanskje er feilslått, i den forstand at den ikke matcher virkeligheten eller behovene til applikasjonen, ønsker vi å stille oss sterke til nødvendige og fordelaktige endringer underveis i utviklingen. I tillegg er vi i en situasjon hvor flere vesentlige aspekter ved applikasjonen ikke er kjent for oss, og vi er derfor avhengig av å klargjøre slikt underveis i utviklingen. Vi har derfor valgt den agile metodikken scrum.
Elementer som typisk inngår i scrum, som sprintplanlegging, sprinter, daily scrum møte, sprint gjennomgang, retrospect o.l. Skal i utgangspunktet anvendes for å skape en autentisk tilnærming til den agile metoden, slik at vi potensielt oppnår alle fordelene ved Scrum.

Roller:

Produkteier:

Merethe Sjøberg (EVRY)

Scrum-Master:

Martin Nenseth

Utviklings-Team:

Christian Moen, Erlend Thorvik, Erlend Wiklem, Sindre Grønstøl Haugeland, Mathias Hartveit

Arbeidstid

Estimert arbeidstid for 20 studiepoeng er 27 timer pr uke pr student. Det vil si at vi har 162 timer totalt per uke til rådighet.

I utgangspunktet ønsker vi å jobbe fra og med Mandag til og med Torsdag i tidsrommet08:00 - 15:00.

Vedlegg 16. Gruppekontrakt

Om gruppe kontrakten

Denne gruppekontrakt gjelder for medlemmer av gruppen Bitfarmers ved gjennomføring av bacheloroppgave ved Universitetet i Agder vår 2018. Gruppekontrakten vil bli brukt som en guide for å motivere og legge føringer for gruppearbeidet for å oppnå gruppens mål. Gruppekontrakten vil gjelde fram til bacheloroppgaven er levert.

Gruppe medlemmer

Gruppe kontrakten gjelder for følgende gruppemedlemmer:

Navn	Epost	Telefon
Christian Moen	moenc15@uia.no	95138942
Erlend Thorvik	erlend_thorvik@hotmail.com	99257554
Erlend Wiklem,	erlenw16@uia.no	99584131
Sindre Grønstøl Haugeland	sghaugeland@gmail.com	90018149
Martin Nenseth	martin.nenseth@gmail.com	95360153
Mathias Hartveit	mhartveit61@gmail.com	92446615

Krav til gruppen medlemmer

Gruppemedlemmer plikter å utføre sin del av gruppearbeidet, møte opp til avtalt tidspunkt og sette seg inn i prosjektet i sin helhet.

Gruppens mål

Gruppens mål er å gjennom prosjektet utvikle et produkt med høy kvalitet ved anvendelse av tekniske ferdigheter og metoder lært gjennom bachelorgraden. Innenfor dette målet inngår å teste kunnskapen som er opparbeidet gjennom utdanningen og sette disse ferdighetene ut i livet til et virkelig prosjekt. På denne måten kan gruppen opparbeide kompetanse på områder som er aktuelle i markedet, slik at gruppen får en

innholdsrik og relevant bacheloroppgave. Gruppen sikter derfor mot høyeste karakter innenfor Universitets standard.

Arbeidsmengde

Selv om det ikke er obligatorisk fremmøteplikt på universitetet, forutsettes det at gruppe medlemmer deltar i den organiserte undervisningen og tilegner seg en stor del av lære- og arbeidsstoffet på egen hånd. Normert arbeidsbelastning for 10 studiepoeng er 270 timer per semester. Dette tilsvarer 13 timers innsats per uke.

Arbeidsmetode

Gruppen vil legge opp til en norm ved å jobbe hver uke fra Mandag til Torsdag kl 08.00 - 15.00 på EVRY sine lokaler der prosjektet skal gjennomføres. Ved avvik fra den initielle planen blir dette på forhånd annonsert av Scrum-Master. Gruppemedlemmer står fritt til å møte opp til forelesninger på Universitetet i Agder under arbeidstiden, men dette må meldes til Scrum-Master.

Kommunikasjon

Hovedsakelig vil kommunikasjon i gruppen foregå under oppmøte på Evry. Ved deling av ressurser skal gruppen formidle dette gjennom Slack. Facebook og Discord kan også bli brukt for å raskt nå gruppemedlemmer. Hvis et gruppemedlem har behov for fri eller har fravær/forsentkomming skal dette avtales/rapporteres til Scrum-Master.

Konsekvenser

Da gruppemedlemmene kjenner hverandre godt fra før, og ikke er redd for noen dramatiske brudd på gruppekontrakten, vil brudd på kontrakt vil bli sanksjonert med advarsler og ved flere advarsler må gruppemedlemmet spandere øl på alle gruppemedlemmer.

Signaturer

17. Eksempel ukentlig rapport

Uke: 8						
Totalt antall timer igjen	**Timer brukt hittil**					
2274	**966**					
Dato	**Timer**	**Utførte arbeidsoppgaver**	**Relevant backloggjenstandnummer**	**Brukerhistorie**	**Utfordringer**	**Notat**
19.01.2018	42	Akseptansetester. Brukerkrav og risikoanalyse møte med PO Implementert brukerhistorie #2 Iverksatt implementering brukerhistorie #1 Research Python BDD rammeverk	# 1 102 # 2 122	#1, 2		Effektiv implementering av brukerhistorie #2 skyldes i stor grad grundig utredning i analysen av biblioteker.
20.01.2018	42	Implementert respons_generator, relevance, restpoint, filter_data, API constants, controller. Satt opp svært enkel prototype med GUI for å illustrere fremgang til PO på review	109-120 + 126, 128, 132	#1	Filtrere data - dataen kommer i ulike formater.	Simple versjoner av respons_generator og filter_data. Vil bli refaktorert, og utvidet senere for å gjøre ApiBot mer intelligent. Implementerer struktur for brukerhistorier 6 og 10, da disse funskjonalitetene må integreres i modulene respons_generator og filter_data.

21.01.2018	42	Refaktorert filter_data. Testing API docs Akseptansetester brukerhistorie #1 og #2.	136, 158	#1, 2		
22.01.2018	42	Refaktorering Kompetansebygging Review Retrospect				
Oppsummert						
Brukerhistorie 1 og 2 implementert. BDD ikke iverksatt fordi vi har testet og bygd kompetanse på dette rammeverket under denne sprinten. BDD med pythons behave rammeverk vil bli anvendt fra og med neste sprint.						

Vedlegg 18. Gruppereferansjon

Måten gruppen har jobbet på gjennom hele prosjektet kan beskrives som flytende, alle medlemmene deltok i varierende grad innenfor alle prosjektets oppgaver. Gruppemedlemmer med tidligere erfaring innen de forskjellige rammeverkene tok en ledende rolle og la til rette for at medlemmer med mindre kunnskap kunne komme seg opp på et akseptabelt nivå raskt, og begynne å bidra konstruktivt under utviklingen. Som det ble nevnt i rapporten ble det ikke fastsatt noen spesifikke roller innad i gruppen, forutenom Martin som tok på seg ansvaret som Scrum master.

Backenden i prosjektet er laget i Python. Python er et programmeringsspråk som flere gruppemedlemmer hadde tidligere kjennskap til. Under oppstartsfasen fokuserte vi på å få alle i gruppen opp på et akseptabelt nivå før den viktigste funksjonaliteten skulle implementeres. For å få dette til, brukte vi en uformell form for par programmering der gruppemedlemmer med mindre erfaring jobbet sammen med mer erfarne medlemmer for å dele kunnskap. Christian spilte en sentral rolle under opplæringen av andre utviklere i Python. Alle gruppemedlemmene har et godt grep rundt grunnprinsippene innenfor objekt orientert programmering, og med et relativt enkelt språk som Python fikk vi samtlige raskt opp på et dekkende kompetansenivå. Peer review av koden ble også et viktig verktøy for utveksling av kunnskap mellom forskjellige ferdighetsnivåene.

Frontend i prosjektet er implementert i React.js. Tre gruppemedlemmer hadde tidligere erfaring med dette biblioteket, hvorav Mathias og Erlend Wiklem var sentrale ressurspersoner som både delte sine kunnskaper i biblioteket godt med resten av gruppen og generelt tok ledende roller for denne delen av applikasjonen. I praksis har Mathias og Erlend Wiklem delt sin kompetanse ved å holde flere små foredrag om alt fra designprinsipper og testing i React.js, i tillegg har de bistått andre, mindre erfarne i React.js gjennom peer programming.

Prosjektets administrative arbeid har i stor grad blitt utført av Martin da han har vært Scrum master. Inn under dette faller arbeidsoppgaver som å kontinuerlig kommunisere med samtlige av prosjektets interessenter, håndheve agile prinsipper under utviklingen, i tillegg til å generelt ta en ledende rolle internt i gruppen og opp mot oppdragsgiver.

Maskinlæringsbiten av prosjektet ble i første omgang håndtert av Sindre Haugeland og Erlend Thorvik. Fra tidligere kurs hadde disse en grunnleggende erfaring med maskinlæring og prinsippene rundt naturlig språk prosessering. Selv om disse tok på seg hovedansvaret for dette tidlig i prosjektet, begynte flere andre gruppemedlemmer å bidra konstruktivt ettersom de opparbeidet seg en forståelse av prosessen og definisjonene innenfor dette feltet.

Ekstensiv testing av kode var en vesentlig utfordring, siden ingen i gruppen hadde tilstrekkelig kompetanse på dette fra tidligere. Det førte til at hele gruppen bidro i utformingen av en testplan, som inneholdt stegene vi så som nødvendig for å sikre den ønskede kvaliteten i produktet. Utførelsen og kontrollen av denne testplanen har gjennom hele prosjektet vært en delt oppgave blant alle gruppemedlemmene.

Utforming av den endelige rapporten ble gjennom hele prosjektet drøftet og diskutert av gruppen i plenum. Det var viktig for oss å komme til enighet om strukturen og skrivemåten for at rapporten skulle fremstå som et helhetlig dokument. Etter at produktet var levert og fokuset flyttet seg til rapporten bidro alle medlemmer under fastsatte økter med skriving. Dersom medlemmer ikke var tilgjengelig grunnet jobb eller lignende, hadde de selv ansvar for å bidra til rapporten etter arbeidstid.