Fredrik Ferdinand Waaler, Klaus Dyvik

Talegjenkjenning for låsesystem

Bacheloroppgave i Dataingeniør
Veileder: Saleh Abdel-Afou Alaliyat & Anniken Susanne T. Karlsen
Mai 2020

Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for IKT og realfag

NTNU
Kunnskap for en bedre verden

Fredrik Ferdinand Waaler, Klaus Dyvik

Talegjenkjenning for låsesystem

Bacheloroppgave i Dataingeniør
Veileder: Saleh Abdel-Afou Alaliyat & Anniken Susanne T. Karlsen
Mai 2020

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for IKT og realfag

NTNU
Kunnskap for en bedre verden

EGENERKLÆRING

Skjemaet under viser at begge studentene på gruppen har satt seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Begge studentene er bevisste på dere ansvar i forhold til oppgaven og klar over hvilke konsekvenser som kan medføre av fusk.

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

FORORD

Om Oss

Vi er to studenter som begge går dataingeniør-studiet på NTNU i Ålesund. Da vi på høsten 2019 skulle begynne å tenke på hva slags oppgave vi ville gjennomføre som bachelor våren 2020, var vi begge enige om at vi ønsket en oppgave som omhandlet maskinlæring og/eller kunstig intelligens. Vi var begge på utveksling det året og hadde egentlig planer om å ta maskinlæring og kunstig intelligens som valgfag i denne perioden. Grunnet fulle klasser og/eller manglende tilbud fra universitetene vi reiste til, ble dette desverre ikke mulig. Vi så derfor på bacheloroppgaven som en god mulighet til å kunne lære oss emnene. I tillegg til at bacheloroppgaven ville gi oss en mulighet til å lære oss et emne vi begge ønsket å se mer på, betød en slik oppgave at vi fikk muligheten til å utfordre oss selv ved å selv sette oss inn i et emne vi ikke kunne noe om fra før. Dette var en forlokkende tanke for oss begge, og vi bestemte oss derfor for å søke på oppgaver som relaterte seg til emnene på et vis eller annet.

Heldigvis for oss endte vi opp med nettopp en av de oppgavene vi søkte på. Bacheloroppgaven har derfor vært en fantastisk læringsmulighet for oss begge. Oppgaven er blitt gjennomført i et todelt løp - På den ene siden har vi måtte bruke mye tid på å sette oss inn i maskinlæring og kunstig intelligens, som var emner ingen av oss kunne noe om fra før. På den andre siden har vi vært så heldige at vi fått muligheten til å sette den nyervervede kunnskapen i bruk med en gang, og vi har kontinuerlig kunne bygget videre på det vi har laget etterhvert som vi har tilegnet oss enda mer kunnskap.

Om Oppdragsgiver

Oppdragsgiver for bachelorppgaven har vært Avento AS. Avento er en konsulentvirksomhet basert i Ålesund, med fokus på IT-rådgivning, systemutvikling og Buisiness Intelligence. De har et kontor på Moa i Ålesund, med omkring 60 ansatte. Flesteparten av de ansatte jobber aktivt ute hos kunder, og er derfor mye inn og ut av kontoret.

Vår hovedkontakt i Avento har vært Oskar Emil Skeide, som har gitt oss god hjelp med å kvalitetssikre produktet, sparre ideer og gi oss tilbakemelding underveis. Dette har hjulpet oss på god vei med å utvikle et produkt som faktisk kan tas i bruk av Avento og som forhåpentligvis møter de kravene og forventningene de hadde sett for seg på forhånd.

Andre bidragsytere

En spesiell takk går også til Saleh Abdel-Afou Alaliyat, Ibrahim Hameed og Anniken Karlsen ved NTNU i Ålesund, som gjennom hele prosjektet har gitt oss gode tilbakemeldinger og tips til forbedring, men ikke minst tro på at vi kan få det til.

Contents

Sammendrag

Vår 2019 hadde Avento AS et testprosjekt ved en annen bachelorgruppe, der de skulle implementere et låsesystem som brukte ansiktsgjenkjenning som verifiseringsmetode. Da dette kom med sine problemer i form av sikkerhet, ville Avento undersøke andre former for verifiseringsmetoder. Oppgaven de framla for oss var å heller prøve å implementere metoder for å bruke talegjenkjenning som verifiseringsmetode i et låsesystem. Denne oppgaven tok også høyde for å skape et fullstack-system rundt systemet for administrasjon og registrering.

Omfanget av rapporten beskriver hvordan gruppen har gått frem for å skape dette systemet. Her beskriver vi hvilket teoretisk grunnlag og hvilke metoder vi har måttet ta i betrakting for å løse problemet, og gjør rede for hvilke resultater vi har hatt i prosessen. Systemet har resultert i et fullt administrativt system som bygger maskinlæringsmodeller basert på hvilken tale brukere har registrert i systemet. I prosjektet har vi undersøkt og sammenlignet ulike typer arkitekturer innenfor maskinlæring for å finne ut av hvilken arkitektur som gir de beste resultatene for vårt problem.

Låsesystemet har resultert i at brukere kan spille inn lydopptak av seg selv som sier frasen "Åpne dør", før et tilbakevendene nevralt nettverk trenes opp på lydklippene. En kan så benytte seg av dette nettverket utenfor kontorarealene til bedriften ved å gjenta denne frasen. Her står det plassert en Raspberry Pi, som fanger opp om denne frasen blir sagt. Dersom noen sier "Åpne dør", vil denne så sende lyden til server for validering, før døren åpnes dersom talen var validert. Den endelige modellen er generelt god på å avgjøre hvorvidt en person skal verifiseres eller ikke. Det kan dog skje at den iblant ikke verifiser personer som skal være verifisert, da modellen av sikkerhetsgrunner har en tendens til å predikere lyddata som uverifisert. I disse tilfellene vil det være tilstrekkelig å gjenta frasen en eller to ganger, og modellen vil slippe deg inn dersom du er verifisert. Et sikkerhetsproblem i løsningen er at det er mulig å ta opp en tale fra en verifisert stemme og spille av lyden framfor systemet, noe som ikke gjør det aktuelt til bruk ved system ved høy sikkerhetsgrad.

Terminologi

Her beskriver vi de begreper og forkortelser leseren må kunne for å lese rapporten.

Begreper

Node - Enhet for beregning innenfor et nevralt nettverk som brukes i forbindelse med maskinlæring og kunstig intelligens. Disse inneholder matematiske formler som ut ifra inn-verdiene bestemmer ut-verdiene, og oppdateres ettersom.
Lag - Et sett av noder som mottar inndata samtidig kalles et lag. Noder i et lag jobber sammen for å videreføre informasjon og gi nye verdier til de neste lagene.
Vekter - De matematiske formelene inne i en node inneholder variabler som endres gjennom kjøretid. Disse variablene er det som bestemmer utfallet til en node basert på input, og kalles for vekter. Dersom man har trent en modell kan man bruke disse vektene for å gjenbruke modellen i samme tilstand den var da den var trent.
Epoch - Når man trener en maskinlæringsmodell gjøres dette gjerne i flere runder, slik at modellen lærer seg dataen bedre. Disse rundene kaller man for epoker, eller epochs fra engelsk.
Treningsdata - Data man bruker når man trener maskinlæringsmodeller kaller vi for treningsdata
Testdata - Data man bruker når man tester hvor gode modeller er kaller vi for testdata. Dette er et tilfeldig utvalg fra det opprinnelige datasettet som ikke blir med på treningen.
Verifisering - I dette prosjekt skal vi med maskinlæringsmodeller evaluere om en bruker skal kunne komme inn i kontorarealene til bedriften. En verifisert bruker er en bruker som skal klare å komme inn, mens en ikke-verifisert er en bruker som ikke skal klare det.
Prediksjon - En prediksjon er resultatet av klassifikasjonen fra en maskinlæringsmodell når man bruker modellen til å klassifisere data.
Samplingsrate - Fra engelsk *Sample Rate*, er samplingsraten antall datapunkter som representerer lyden per sekund i en lydfil. **Chunk** - En chunk er antall datapunkter en opptaksenhet tar opp i hver omgang. Disse kan senere settes sammen, som da utgjør hele lydfilen.

Forkortelser

I/O - Input/Output
API - Application Programming Interface
LSTM - Long Short Term Memory
VM - Virtuelle Maskiner
HTTP - Hypertext Transfer Protocol
URI - Uniform Resource Identifier. En addresse som peker til en spesifikk ressurs på en server

URL - Uniform Resource Locator er en spesifikk URI, som også inkluderer addressen til der ressursen ligger
GUI - Grafisk brukergrensesnitt (Eng. Graphical User Interface)

1 Innledning

I innledningen forteller vi litt om bakgrunnen for oppgaven og hvordan oppgaven er formulert fra oppdragsgiver sin side. Her ser vi hvilke behov oppdragsgiver har, og viser hvordan det ferdige systemet ser ut. Deretter forteller vi videre hvordan innholdet i rapporten er lagt opp.

1.1 Bakgrunn

Ansatte hos bedriften Avento AS bruker i dag en RFID-brikke som autentiseringsmetode for å komme inn på kontorarealene deres. Denne metoden fungerer, men krever at ansatte må ha brikken med seg til enhver tid de er på jobb. For å unngå det, ønsker Avento å finne andre metoder for autentisering for adgang til lokalene sine. I det siste har biometriske autentiseringsmetoder blitt mer og mer aktuelt, som er autentiseringsmetoder som benytter seg av unike karakteristikker hos enkeltmennesker for å identifisere personer. Metodene inkluderer gjenkjenning av fingeravtrykk, ansikt, stemme og øyne, med mer. Siden disse karakteristikkene er unike fra person til person, er autentisering et hypotetisk bruksområde for biometriske metoder. Biometri er dog fortsatt et felt i vekst, og prøving og feiling gjenstår for å avgjøre til hvilken grad, og med hvilke metoder, biometri kan brukes trygt i slike tilfeller [1]. Avento har de siste par årene dog hatt et ønske om å se på hvorvidt ulike biometriske metoder kan tas i bruke for å erstatte RFID-brikken de nå bruker for innlåsing til sine kontorlokaler. Våren 2019 hadde de derfor en gruppe studenter hos seg i forbindelse med en bacheloroppgave. Gruppens oppgave var implementere nettopp en slik biometrisk løsning, med ansiktsgjenkjenning som en verifiseringsmetode. Denne løsningen brukte eksterne skytjenester som Azure for å gjennomføre ansiktsgjenkjenningen. Med den ferdige løsningen kunne ansatte registrere seg med et bilde som ble lastet opp til en Raspberry Pi. Ved inngangsdøren til kontorlokalene ble det så plassert et kamera, slik at løsningen kunne sammenligne bildet som ble brukt for registrering med det kamera ved inngangsdøren fanget opp. Et stort problem med tjenesten var dog at den ikke skilte mellom en faktisk person som stod ved døren og en bilde av en verifisert person som ble holdt foran døren. Med andre ord, for å komme seg inn i kontorlokalene var det nok å ha et bilde av en som jobbet der. For å rette opp på dette laget teamet en iOS applikasjon for to-faktor autorisering, men dette gjorde i bunn og grunn verifiseringen mer komplisert enn det den tidligere hadde vært med RFID-brikke.

1.2 Formulering

Siden den tidligere implementerte løsningen med ansiktsgjenkjenning ikke fungerte som planlagt, ønsket Avento å se på andre mulige løsninger for å låse seg inn til kontorlokalene. Talegjenkjenning er en annen biometrisk metode som er veldig i vinden for tiden, og Avento ønsket derfor å se på muligheten for å utvikle et låsesystem ved hjelp av slik teknologi. De foreslo derfor for oss, at vi kunne forsøke å utvikle et slikt system som en bacheloroppgave, for å teste om det var

et godt nok alternativ som autentiseringsmetode for deres låsesystem. Opprinnelig ble vi gitt en åpen oppgave, der målet var å implementere en løsning som bruker talegjenkjenning for å regulere tilgangen til inngangsdøren ved Avento sine kontorlokaler, men etter dialog med bedriften fikk vi vite mer om hvilke krav de hadde. I tillegg til talegjenkjenning ønsket de at løsningen overordnet skulle være enkel å utvide, slik at man for eksempel kan bruke systemet for andre applikasjoner. En mulig utvidelse som ble nevnt i utredningen av oppgaven var tilkobling til bedriftens kaffemaskin, slik at brukere kunne igangsette trakting av kaffe ved bruk av stemmekommandoer. Avento ville også at en slik løsning skulle inkludere administrative systemer for løsningen, slik at en administrator innad i bedriften kunne overvåke bruken av systemet gjennom en logg, registrere nye brukere, slette eksisterende brukere, og lignende.

Helhetlig skulle altså det sammensatte systemet se omtrent slik ut:

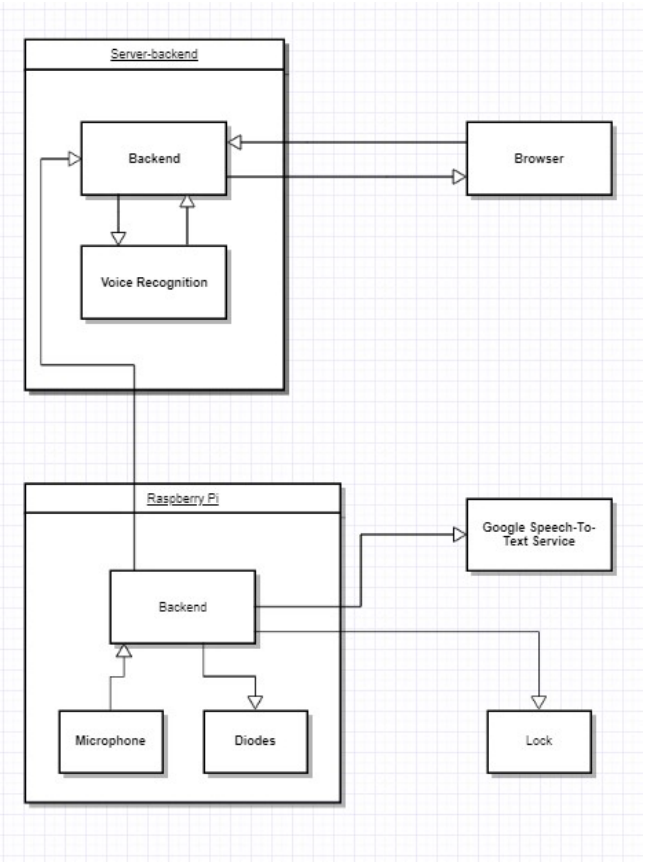


Figure 1.1: Distribusjonsskjema for prosjektet

En Raspberry Pi (3.1) tar opp stemmer ved inngangsdøren til Avento. Den bruker så tale-til-tekst-tjenesten Google Speech-To-Text (3.9.4) til å registrere hva som blir sagt. Dersom en spesifikk frase blir sagt, vil lydklippet sendes inn til en server med modellene for talegjenkjenning, som vurderer hvorvidt personen ved Aventos inngangsdør skal autoriseres. Et signal sendes tilbake til Raspberry Pi, som ved godkjenning sender et signal til låsen ved inngangsdøren slik at den åpnes opp. Parallelt kjøres det en webapplikasjon som snakker med serveren med maskinlæringsmodellene, som kan brukes til å hente ut loggføringer osv. om hvordan modellene brukes, i tillegg til å holde orden på brukere i systemet.

Siden Avento ved dette prosjektet vil finne ut om talegjenkjenning er en god metode for autentisering for deres låsesystem, kommer vi også gjennom rapporten til å ta for oss sikkerhetsperspektiver der det er hensiktsmessig. Både oppdragsgiver og gruppen er fullt klare over at det vil være krevende å få frem en løsning på talegjenkjenning som fungerer i alle tilfeller, men vil bruke kunnskapen som en læring av teknologiene. Da kan vi heller ta en vurdering om i hvilken grad vår implementasjon av talegjenkjenning er brukbar basert på resultatene. I systemet rundt var det et krav at det er muligheter for tilpasning, så det eventuelt viderearbeid enkelt kan implementeres.

1.3 Innhold

Denne rapporten vil videre dokumentere arbeidet som er gjort i forbindelse med problemstillingen som ble gitt fra Avento. Først vil rapporten ta for seg hvilket arbeid som ble gjort innledningsvis, med tanke på å videre utrede oppgaven, samt hente inn relevant informasjon om eksisterende løsninger og teorier som har vist seg nyttig i utviklingsprosessen, og som den ferdige løsningen baserer seg på. Når det teoretiske grunnlaget er på plass, vil rapporten betrakte de prosedyrer, organisatoriske metoder og kommunikasjonsmetoder som er brukt for å utvikle, kvalitetssikre og styre prosjektet. Deretter vil rapporten ta for seg de tekniske verktøyene (programmeringsspråk, eksterne biblioteker, osv.) som er tatt i bruk for å utvikle systemet og hvordan disse spiller inn på det endelige resultatet. Når all bakgrunnsinformasjon som så er nødvendig for å forstå den utviklede løsningen er lagt frem, vil rapporten vise til resultatene som er kommet fra arbeidet - hva har inngått i arbeidsprosessen og hva står vi igjen med etter endt arbeid. Det vil fremkomme en detaljert beskrivelse av de ulike komponentene i løsningen, både fra talegjenkjenningssystemet og fra det administrative systemet, og det vil drøftes hvorfor løsningen ble utviklet på den måten den ble. Etter fremlagte resultater vil rapporten ta for seg de begrensningene som er relevante for prosjektet, samt de endringene/avvikene som oppstod under arbeidet med løsningen i henhold til den opprinnelige planen og hvordan det generelle prosjektarbeid har gått for seg. Det vil også drøftes hvorvidt den endelige løsningen tilfredsstiller kravspesifikasjonen, eventuelle mangler og foreslåtte områder for videreutvikling. Rapporten avsluttes med en konklusjon som oppsumerer prosjektets resultater og de erfaringene som er blitt til under prosjektets arbeid. Til slutt følger en kildeliste og vedlegg.

2 Teoretisk Grunnlag

I dette kapittelet tar vi for det teoretiske grunnlaget av teknikker vi har brukt når vi har gjennomført prosjektet.

2.1 Eksisterende løsninger

Det ble opprinnelig foreslått fra oppdragsgiver at vi kunne prøve å implementere systemet for talegjenkjenning gjennom bruk av eksisterende API'er eller lignende. Avento hadde på forhånd notert seg at Amazon og Azure hadde noen API'er for talegjenkjenning [85] [86]. På oppfordring fra Avento ble gruppen derfor instruert om å se på hvorvidt disse API'ene allerede hadde løsninger for talegjenkjenning som kunne utvides til det aktuelle formålet.

Etter det første møte med Avento, der beskrevet oppgavebeskrivelse ble framlagt, ble første prioritering umiddelbart å se på de foreslåtte eksisterende løsningene, Amazon og Azure sin teknologi for talegjenkjenning. Sammen konkluderte vi med at de to løsningene ikke hadde tilstrekkelig funksjonalitet til at det ville vært hensiktsmessig å tilpasse disse til oppdragsgivers behov. Det var i hovedsak to store problemer med den eksisterende teknologien. For det første kunne Amazon og Azure sine tjenester kun brukes til å gjenkjenne et smalt sett av spesifikke fraser. Dette byr på utfordringer, hovedsakelig fordi det ikke vil være veldig brukervennlig å tvinge brukere til å huske ikke-intuitive fraser for å låse seg inn. En eksempelfrase fra Azure sin løsning var "I am going to make you an offer you simply cannot refuse". Det ville vært mer ønskelig dersom brukere av løsningen f.eks. kunne si noe mer intuitivt, slik som "Åpne dør", da det er mer relevant for bruken til Avento. For det andre, krevde eksisterende løsninger (Amazon og Azure) at man sendte inn bruker-id'en til personen man ønsket å verifisere, sammen med lydopptaket som ble forsøkt verifisert. Dette komplisierte ting ytterligere, fordi man ved verifisering da ville være nødt til å finne ut av hvem det var som prøvde å verifisere seg - her ville det være mere ønskelig at både identifisering og verifisering kun baserte seg på innsending av taleopptak. Andre eksisterende løsninger som tilbydde annen funksjonalitet for talegjenkjenning hadde også samme, om ikke fler, begrensinger som Amazon og Azure sine implementasjoner.

Etter videre utredning i samarbeid med veiledere, så vi deretter på muligheten for å utvikle egne modeller for talegjenkjenning (ved hjelp av kunstig intelligens og maskinlæring). Grunnen til dette var at vi da ville kunne lage en løsning som var mer tilpasset til oppdragsgiver. F.eks. ved å lage en løsning som ville kunne gjenkjenne en person basert på stemmen, uavhengig av hva denne personen sier, uten å instruere maskinen om hvilken person man prøver å verifisere. På denne måten ville vi omgå begrensingene som ble pålagt av eksisterende teknologi. Oppdragsgiver synes det var et positivt forslag, og bestemte med dem at vi skulle forsøke på dette, slik at de fikk en løsning som var mest tilpasset dem. Det overordnede målet er fortsatt at disse modellene skal kunne brukes for å

regulere tilgang til bedriftens inngangsdør, og samtidig kunne utvides til å ta i bruk andre steder. Den reviderte versjonen av oppgaven inneholdt fortsatt også implementasjon av annen funksjonalitet som ligger til grunn for at løsningen skal kunne brukes og utvides. Dette inkluderer som nevnt administrative systemer for brukerbehandling og loggføring, samt annen støttefunksjonalitet som databaser for datalagring, og lignende.

2.2 Kunstig Intelligens

For å løse denne oppgaven har vi valgt å undersøke en rekke forskjellige metoder innenfor kunstig intelligens for å klassifisere om en person skal være autorisert eller ikke.

2.2.1 Konvolusjonelle Nevrale Nettverk

Ett type nevralt nettverk er konvolusjonelle nevrale nettverk. Dette er nevrale nettverk som ser på bilder og forholdet mellom pikslene i bildet. Vi refererer til konseptet ved *CNN*, fra det engelske *Convolutional Neural Network*. Hovedsakelig tar modellen en konvolusjon - en delmengde av et bilde - og generaliserer ut fra informasjonen delmengden inneholder. Man gjør dette for å hente ut de mest signifikante karakteristikkene fra et bilde, slik at man lettere får bestemt hva som er i bildet. Til slutt bruker man verdiene man sitter igjen med i vanlige nevrale lag for å klassifisere input. Prosessen er illustrert i figur 2.1.

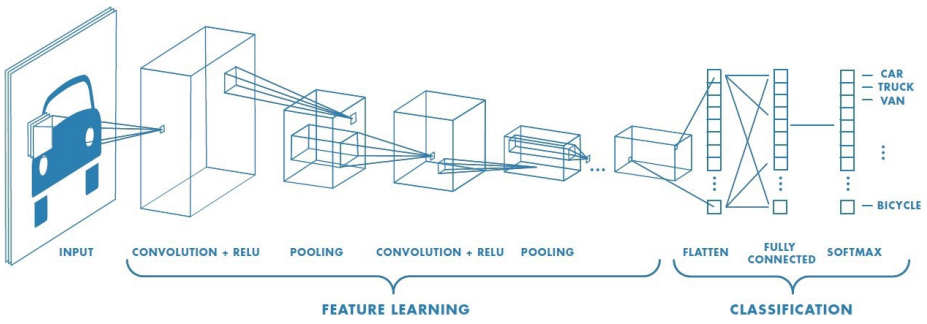


Figure 2.1: Illustrasjon av CNN-arkitekturen [87]

Konvolusjonelle nevrale nettverk har vist å gi gode resultater i eksisterende prosjekter som tar for seg relaterte problemstillinger, f.eks. for å bestemme hvem som snakker i en gitt gruppe med mennesker [53], eller for å bestemme om den som snakker er den han/hun utgir seg for å være blant et spesifikt sett med personer [54]. Dermed var det naturlig for oss å se på slike nevrale nettverk i forbindelse med vår oppgave. Da vi i vårt prosjekt blant annet også representerer taledata som MFCC-verdier (2.5.6), som igjen kan representeres som todimensjonale bilder, har dette gitt oss videre motiv for å teste ut kon-

volusjonelle nevrale nettverk - da dette som nevnt er nettverk som jobber mot bildedata.

2.2.2 Tilbakevendene Nevrale Nettverk

Tale er en tidsserie. Innenfor nevrale nettverk er det et konsept som heter tilbakevendene nevrale nettverk. Disse nettverkene ser på tidsserier og hvordan signaler i tidsseriene er i forhold til hverandre. Derfor kan slike type nettverk være en god løsning. Også her, som i likhet med CNN, har tilbakevendende nevrale nettverk vist seg å fungere godt for relaterte problemstillinger for talegjenkjenning - mer spesifikt for å bestemme hvem som prater på et lydklipp fra et spesifikt sett med personer [55]. Vi refererer til konseptet som *RNN* fra det engelske *Recurrent Neural Network*. Måten et slikt nettverk håndterer data på, er at output fra RNN-lagene sendes tilbake til de gjemte nodene, slik at de kan brukes på nytt, vist i figur 2.2.

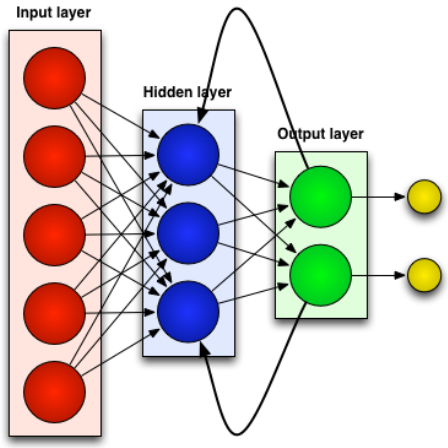


Figure 2.2: Illustrasjon av RNN-arkitekturen [90]

2.2.2.1 LSTM

RNN-arkitekturen har sine fordeler ved at informasjon videreføres til den neste cellen. Likevel er dette problematisk over lengre tid, da nettverkene begynner å glemme konteksten til talen i sin helhet. Dette er på grunn av at i tradisjonelle RNN opptar hver node kun informasjon fra den forrige noden, og ikke i tillegg ser på talen i sin helhet. Dette er kalt *The Vanishing Gradient Problem*[58]. For å løse dette innfører vi LSTM (Long Short Term Memory), som bedre ivaretar kjent informasjon gjennom å lagre data over tid[59]. Dette gjør den ved å ha en komponent som også tar med tidligere input inn i den neste noden.

2.3 Talegjenkjenning

For å kunne skille mellom et individ basert på stemme må vi ta i bruk talegjenkjenning. Talegjenkjenning er en biometrisk metode for å kjenne igjen unike karakteristikker fra stemmen til et individ. Tanken er så at man skal kunne identifisere hvem som snakker ved å se på disse karakteristikkene. Hvordan man finner hvilke karakteristikker som identifiserer en tale er gjennom en rekke matematiske prosesser man utfører på lyd-dataen. Forsvarets Forskningsinstitutt beskriver gjennom rapporten "Signalrepresententasjoner for automatisk talegjenkjenning"[96] ulike metoder for hvordan man kan han hente ut karakteristikker fra talen til bruk for talegjenkjenning. Hvilken metode vi har brukt skriver vi mer om i 2.5.

2.3.1 Sikkerhet ved bruk av talegjenkjenning

For å finne ut om talegjenkjenning er en optimal løsning for oppdragsgiver, må vi undersøke eventuelle trusler i systemet vi skal utvikle, slik at oppdragsgiver kan ta det i betrakting når de skal vurdere om de vil bruke løsningen. Ved bruk av talegjenkjenning kommer det som ved alle andre verifiseringsmetoder sikkerhetsspørsmål relatert til hvor lett det er å finne metoder for å misbruke systemet - f.eks. ved å låse seg inn på en utilsiktet måte. Det er to perspektiver vi har tatt for oss i prosjektet.

Siden vi i dette prosjekt benytter oss av teknikker for maskinlæring, er sannsynligheten for at modellene tar en feil vurdering et kritisk punkt. Det vil si at en modell ikke skal gi en person som ikke er en registrert bruker tilgang til systemet. Denne feil kan også oppstå når personer som er ukjente for systemet prøver å verifisere seg selv. I en maskinlæringsmodell er det viktig å teste med data som er ukjent for maskinen. I klassifikasjonsproblem man gjerne bruker maskinlæring for å løse, holder man av en bit av dataen som skal brukes som et testsett for modellene, som man kan bruke til å gi en indikator på hvor gode modellene er. Et eksempel kan være å bruke en database av bilder av dyr, og prøve å klassifisere hvilket dyr som er på ethvert bilde. Her er alle bilder forskjellig, og man kan lett sette av en delmengde bilder for testing av modellen. I vårt tilfelle derimot vil dette testsettet være augumenterte filer av de samme stemmene vi har trent modellen på, og vil derfor ikke representere alle stemmer som finnes. Siden disse lydene ikke er med i test- eller treningssettet, er det mulig at lydene likevel blir klassifisert som verifisert. Vi må derfor sørge for at modellen har en bias mot å klassifisere stemmer den ikke kjenner til som uverifisert, ved å holde av lydfiler før de er lagt til i datasettet til bruk testing av ekstern data.

Det andre perspektivet omhandler i hvilken grad det er mulig å lure systemet til å tro at du er en bruker av systemet ved å stjele det biometriske avtrykket til en faktisk bruker. For at en biometrisk metode skal være sikker, må det ikke være mulig å gjenskape verdiene. Som skrevet i bakgrunnen av oppgaven

([**?**]), kunne man under ansiktsgjenkjenningssystemet til den forrige gruppen holde opp et bilde av en person for å så komme inn, og var derfor ikke en god nok verifiseringsmetode. Vi ser for oss et scenario der en uavhengig part tar opp lyden fra et godkjent forsøk i et talegjenkjenningssystem, og senere bruker opptaket for å bli verifisert for det samme systemet. Derfor må vi finne ut om dette er en fare ved bruk av systemet.

2.4 Normalisering av lydnivå

Lyddata vi bruker i prosjektet er hentet fra ulike kilder, som vi skriver om i detalj i 3.12. Det gjør at datasettet er basert på lyd tatt opp må mange forskjellige enheter, i ulike forhold. Derfor vil lydnivået variere mellom enhver lydfil. Siden disse lydnivåene varierer, vil vi normalisere opptak slik at alle opptak har samme lydnivå - dette gjøres for at lydnivå ikke skal ha noe å si for talegjenkjenningen, da dette kan være en faktor som f.eks. avhenger av mikrofonen som er brukt for opptak. Det er ikke relevant for karakteristikkene som brukes for å identifisere stemmen på lydklippet. En måleenhet for lydnivå er dbFS (*Decibels relative to full scale*) [51], der man ser på volum i forhold til den høyeste mulige lydnivå man kan generere digitalt. I oppgaven omformer vi alle lydfiler til -20dBFS før prossesering, som betyr at lydnivået ligger på 20 decibel under denne maksimumverdien.

2.5 Lydprossesering

I oppgaven skal vi skille mellom stemmene til enkeltpersoner. Det gjør at vi må hente ut visse egenskaper som er unike til enhver person. Dette er det vi kaller *fingeravtrykket* til en stemme. For å hente ut egenskapene til en stemme, må en rekke matematiske skritt til. Etter skrittene beskrevet under sitter man igjen med Mel-frekvens cepstralkoeffessienter (MFCC), som er en måte å representere disse unike egenskapene på.

2.5.1 Diskret Fourier-Transformasjon

Når vi begynner lydprosessering må vi først finne ut av hvilke frekvenser som er tilstede i signalet. Et signal kan uttrykes ved å sette sammen en rekke sinusfunksjoner [**?**], som vi kaller frekvenskomponenter. Dette gjør vi ved å utføre diskret Fourier-transformasjon på dataen [2].

En diskret Fourier-transformasjon bruker en viss mengde (N) diskrete punktprøver fra lydsignalet over tid til å evaluere en summering som gir en frekvenskoeffisient for hver punktprøve i lydsignalet, gjennom formel 1 under.

$$X_k = \sum_{n=1}^{N-1} x_n * e\frac{-i2kn\pi}{N} \tag{1}$$

Her representerer k en spesifikk punktprøve, og N representerer det totale antallet punktprøver. Regner man ut dette for alle punktprøvene får man en matrise som uttrykker signalet over tid.

2.5.2 Korttids Fourier-Transformasjon

Frekvensene vil endre seg over tid og frekvensene vil variere. Derfor kan det være fordelaktig å se på forskjellige deler av lyden når vi utfører Fourier-transformasjonen og heller sette de sammen senere. Disse delene er kalt vinduer. Siden lyden ikke endrer seg så raskt, kan vi anta at frekvensene i lyden er lik i et veldig lite vindu [49]. Derfor utfører vi teknikken korttids fourier-transformasjon (STFT) ved å ta diskrete fourier-transformasjoner over korte vinduer [3]. Dette gir en mer nøyaktig beskrivelse av frekvensspekteret i lydprøven.

2.5.2.1 STFT-Spektrogram

Etter å ha utført STFT på en lydfil, kan vi lage et spektogram for verdiene som kommer frem av transformasjonen. For å illustrere dette har vi vist hvordan lydfilen i figur 2.3 ser ut når den er gjort om til et STFT-Spektrogram i figur 2.4.

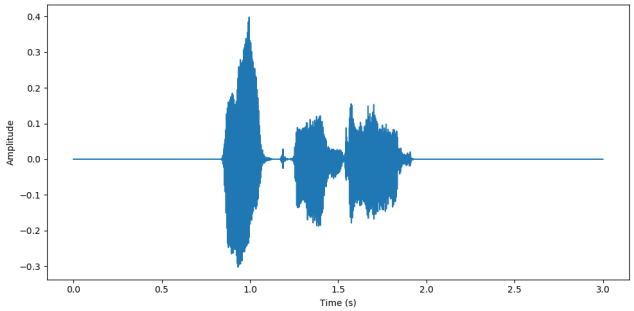


Figure 2.3: Lydfil

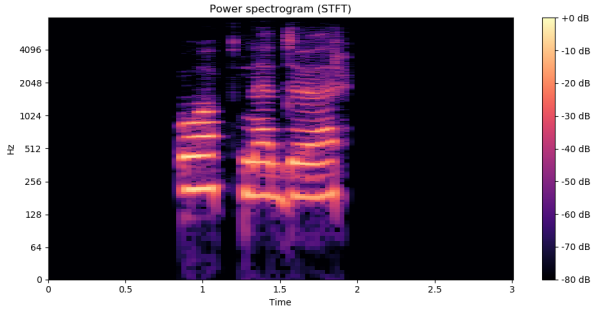


Figure 2.4: STFT-spektrogram

2.5.3 Hanning-vindu

Et Hanning- eller Hann-vindu brukes for å få jevnere data når vi utfører Korttids Fourier-Transformasjon på dataen. En av grunnene til dette er at transformasjonen antar at signalet fortsetter uendelig, så vi vil slake ut kurven der vi skal utføre den. [50]. Det finnes mange forskjellige metoder for å utføre slik begrensning av kurven, men Hanning er et av de mest brukte. Formel 2 viser hvordan man regner ut Hanning-vinduet.

$$w(n) = 0.5 - 0.5cos(2\pi n)/(M-1), 0 \leq n \leq M-1 \tag{2}$$

For eksempel vil et vindu av *M=512* prøver se slik ut

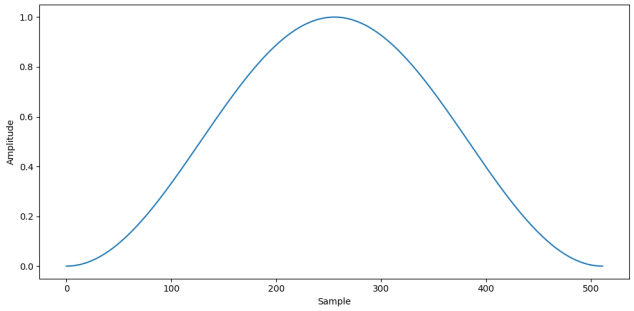


Figure 2.5: Hanning-vindu med *M=512*

2.5.4 Mel-skala

Lyd er logaritmisk, og det kan derfor være vanskelig for mennesket å skille mellom toner i en lyd. Mer spesifikt er det vanskelig å vite hvor stor avstand det er mellom to toner, både fordi forholdet endrer seg raskere jo høyere lyden er, og fordi hjernen vil oppfatte lyd mer lineært enn det den faktisk er. For å kompensere for denne menneskelige feilslutningen, oppfant Stevens, Volkmann og Newman [93] mel-skalaen, som mennesket lettere kunne bruke som referanse til å skille lyd, ved å bruke formel 3.

$$m = 2596 * log_{10}(1 + f/700) \tag{3}$$

Denne gjør at lyd på forskjellige frekvensnivåer kan representeres lineært og ikke logaritmisk. Disse verdiene kaller vi for melverdier. Det finnes noen forskjellige formler for å konvertere til mel-skalaen. I prosjektet har vi brukt formelen i 3, fordi det er den mest utbredte, i tillegg til at Python-pakken *librosa* vi bruker for lydprossessering tar i bruk den. For alle mulige frekvensverdier som mennesker kan oppfatte (20Hz til 20.000Hz) er forholdet slik:

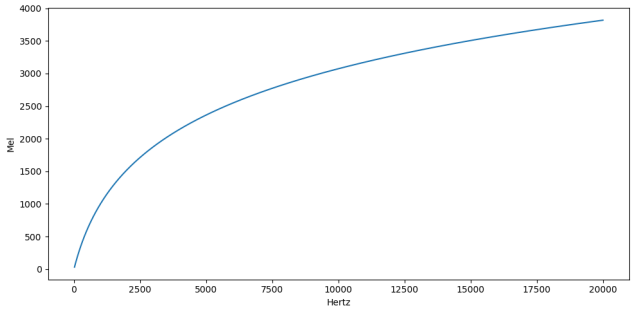


Figure 2.6: Forholdet mellom hertz og mel-verdier

2.5.4.1 Melspektrogram

For å vise hvordan omgjøring til melverdier påvirker representasjonen av lyden kan vi plotte et spektrogram av melverdiene. Dette kalles et melspektrogram. Lydfilen i figur 2.3 er gjort om til et slikt spektrogram i figur 2.7. Vi ser her at talen blir enda mer distinkt enn da vi kun utførte STFT på lydfilene (Fig. 2.4).

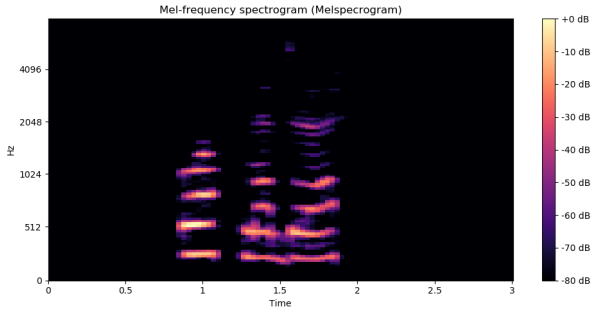


Figure 2.7: Melspektrogram

2.5.5 Mel-filterbanker

Dersom man gjør om lyden til melverdier (2.5.4), blir dette representert som lineært og ikke som logaritmisk. Derfor kan vi enklere se på lyd slik som mennesker oppfatter den. Fra melverdiene kan man så legge til et filter av en viss delmengde frekvenser, slik at man kan representere hvor signifikante frekvensene er totalt sett i lyden. Disse verdiene utgjør mel-filterbanker [49].

2.5.6 Mel-frekvens Cepstralkoeffisienter

Mel-frekvens cepstralkoeffesienter (MFCC) danner et mel-frekvens cepstrum, som er det kortsiktige kraftspektrumet til en lyd. Ordet "cepstrum" kommer fra en omrokering av bokstavene i ordet "spectrum". Det består av koeffesientene funnet ved å ta diskret cosinustransformasjon av Mel-filterene[37]. Vi kartlegger verdiene i en to-dimensjonell liste for å representere verdiene over tid. Når verdiene er representert på denne måten får vi et bilde vi kan bruke som inndata i konvolusjonelle nevrale nettverk. Siden x-aksen representerer tid, vil det også være mulig å bruke dataen som input i et tilbakevendene nevral nettverk.

2.5.7 MFCC-Spektrogram

MFCC er en rekke koeffesienter over tid, så vi kan også gjøre om lyden i figur 2.3 til et MFCC-spektrogram slik som i figur 2.8.

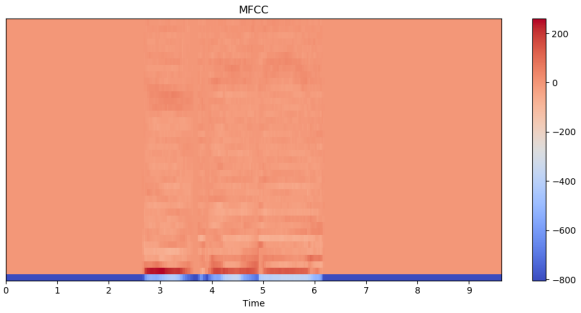


Figure 2.8: MFCC-Spektrogram

2.6 Mål på ytelse av modeller

I maskinlæringsverdenen har man mål på hvor gode modeller er. Man vil hovedsakelig ha mål på hvor mye av dataen i testsettet som blir gitt en riktig etikett under predikering. Når vi i vårt problem skiller mellom to forskjellige kategorier (autorisert og ikke-autorisert), kaller vi det binær klassifikasjon. Under binær klassifikasjon ser man på fire kategorier:

- **Sann Negativ (TN)** - Når data klassifiseres som negativ og skal være negativ.
- **Falsk Negativ (FN)** - Når data klassifiseres som negativ, men skal være positiv.
- **Sann Positiv (TP)** - Når data klassifiseres som positiv og skal være positiv.

- **Falsk Positiv (FP)** - Når data klassifiseres som positiv, men skal være negativ.

Hver av disse kategoriene beskriver prosentandelen av data som blir klassifisert som positiv og negativt i henhold til antall data i den faktiske klassen.

2.6.1 Forvirringsmatrise

Et av de viktigste målene er en forvirringsmatrise, som er en representasjon av forholdet mellom den virkelige etiketten data har og hvilken etikett dataen ble klassifisert som av maskinlæringsmodellen. Hvis man har målene for punktene ovenfor, kan man opprette en forvirringsmatrise på denne måten:

		Predicted	
		Positive	Negative
Actual	Positive	TP	FN
	Negative	FP	TN

Table 1: Eksempel på forvirringsmatrise

Fra tabellen ser vi at faktisk etikett er langs y-aksen, og den predikerte etiketten er langs x-aksen.

2.6.2 Nøyaktighet

Et mål for å sjekke hvordan modeller yter for alle tilfeller, er nøyaktighet. For å regne nøyaktigheten til en maskinlæringsmodell bruker man formel 4, med de ulike prosentandelene til verdiene beskrevet i innledningen til 2.6.

$$ACC = \frac{TP + TN}{TP + FP + TN + FN} \tag{4}$$

2.7 Agile metoder

Før vi begynte med utviklingsarbeidet for dette prosjektet, så vi på flere ulike metodikker for software-utvikling, som f.eks. fossefallsmetoder og spiralmodellen. Vi fant dog ut av at ingen av disse var helt velegnet til oppgaven vår, da vi visste at oppgaven sannsynligvis ville undergå endringer underveis og at feedback fra oppdragsgiver og veileder ville dra prosjektet i retninger vi ikke kunne forutse fra starten av. Basert på erfaring og tidligere kunnskap valgte vi derfor å gå for utviklingsmetodikken agile metoder.

Agile metoder er et samlebegrep for en rekke utviklingsteknikker innenfor software-utvikling. Metodene baserer seg i hovedsak på å jobbe kjapt og smidig og å være tilpasningsdyktige under utvikling. Det agile manifestet nedenfor summerer hovedpunktene innenfor agile metoder:

1. Individer og interaksjoner over prosesser og redskap
2. Fungerende software over omfattende dokumentasjon
3. Samarbeid med kunden over kontraktsforhandlinger
4. Respondere til endringer over å følge en plan

Ved å jobbe agilt under utvikling, har vi som utviklere enklere kunne tilpasset oss til en oppgave med usikre komponenter der vi fra start ikke har kunne si med sikkerhet hvordan det ferdige produktet skal se ut (f.eks. Azure API (2.1) mot egenlagde modeller for maskinlæring). I tillegg har det agile fokuset på fungerende software gjort at vi ved bi-ukentlige møter med veiledere og under møter med oppdragsgiver har hatt muligheten til å vise frem noe konkret, slik at vi kontinuerlig har kunne fått tilbakemelding. Mer spesifikt for vår oppgave, har vi brukt den agile metoden Scrum. Denne vil bli ytterligere forklart senere i rapporten (3.6).

2.8 Cohesion og Coupling

Cohesion og coupling er to designprinsipper innenfor programvareutvikling som ofte blir brukt sammen.

Coupling refererer til hvor avhengige ulike moduler og klasser i en programvare er av hverandre. Som et generelt designprinsipp sikter man mot lav coupling - altså lav sammenheng mellom modulene i programvaren. Lav sammenheng mellom de ulike delene av software, sikrer at man enkelt kan endre implementasjonen og de underliggende mekanismene som utgjør en enkelt modul eller klasse uten å måtte gjøre endringer i andre deler av programmet som tar bruk av denne modulen eller klassen.

Cohesion referer til hvorvidt alle elementer i programvaren som rettes mot å utføre en spesifikk enkelt oppgave, er samlet i den samme klassen eller modulen. Som et generelt designprinsipp sikter man mot høy cohesion - altså høy logisk/funksjonell selvstendighet for de ulike modulene og klassene som utgjør en programvare. Høy cohesion er med på å gjøre software-komponenter mer gjenbrukbare og enklere å vedlikeholde, da funksjonaliteten de leverer vil være selvstendig og logiske endringer ikke vil påvirke andre deler av programvaren.

Lav coupling og høy cohesion har vært to viktige designprinsipper under utvikling av vårt produkt. Da vi har testet ut mange forskjellige metoder for maskinlæring, har det f.eks. vært svært fordelaktig at vi kan endre på den interne strukturen til et nevralt nettverk, uten å samtidig måtte endre på hvordan datasettet til maskinlæringsmodellen preprosesseres og behandles - og vice versa.

2.9 Nettverksprotokoller

På internettet er det en rekke nettverksprotokoller, som er et sett av regler som brukes ved kommunikasjon mellom enheter som kommuniserer over nettet. Protokollene er standardiserte gjennom en formell enighet gjennom internasjonale forum, slik som W3C og ISO. Nedenfor beskriver vi de mest essensielle protokollene brukt i prosjektet for å oppnå det endelige resultatet.

2.9.1 Hypertext Transfer Protocol

For å kommunisere mellom serveren vår og andre delsystemer har vi brukt Hypertext Transfer Protocol (HTTP). HTTP er en protokoll for overføring av hypermedia på internettet. Protokollen følger en struktur der en klient snakker med en server som deretter svarer. Prosessen er som følger:

1. Klient sender en forespørsel (request) til server.
2. Server mottar forespørselen og gjør det den blir bedt om.
3. Server sender tilbake en respons (response) til klienten.
4. Klienten mottar responsen og kan benytte seg av informasjonen.

2.9.1.1 Forespørsler

Det finnes forskjellige forespørsler en klient kan gjøre mot en server. Ut ifra hvilken forespørsel som blir sendt er det forskjellige regler. Her har vi tatt for oss de forskjellige typer forespørsler vi bruker i oppgaven

- **GET** - GET er en forespørsel som sendes dersom klienten skal be om noe fra serveren. Et godt eksempel på dette er når man skal hente en nettside fra serveren. Da sender man en GET-forespørsel til en webserver og ber om alle dokumentene som trengs for å generere nettsiden (HTML-, CSS- og JavaScript-dokumenter). På serveren er det da definert hvilke dokumenter som sendes tilbake.
- **POST** - En POST-forespørsel gjøres når klienten ønsker å sende inn informasjon til serveren. Det kan for eksempel være å sende inn data fra et spørreskjema til lagring på serveren. Etter en POST er det vanlig å også få tilbake informasjon slik som i GET, men gjerne som en bekreftelse for klienten på at handlingen har blitt utført.

2.9.1.2 Responser

Når en server har mottatt en forespørsel sender den tilbake en respons til klienten. Responsen inkluderer en kode, som sier noe om hvordan serveren har håndtert forespørselen. Nedenfor beskriver vi de vanligste kodene brukt.

- **200 OK** - Koden 200 OK betyr at alt har gått som det skal. Da har serveren innfridd ønsket til klienten. I meldingsdelen av denne responsen legger serveren med data, slik at klienten får den forespurte informasjonen.

- **400 Bad Request** - Koden 400 Bad Request blir returnert av serveren dersom den ikke forsto forespørselen.
- **404 Not Found** - Koden 404 Not Found dukker opp dersom serveren ikke har definert et utfall for forespørselen. Det kan være at man prøver å gå inn på en lenke som ikke eksisterer i systemet.
- **500 Internal Server Error** - Dersom serveren får en feil under prosessering av en forespørsel vil koden 500 Internal Server Error vises.

2.9.2 Representational State Transfer API

Representational State Transfer API (REST API) er en stil for distribusjon av hypermedia mellom servere og klienter som kommuniserer gjennom HTTP. I vårt prosjekt bruker vi dette prinsippet til å levere nettsider til klienter fra serveren, samt kommunisere over nett innad i systemet (f.eks. mellom Raspberry Pi og server for å gjøre prediksjoner). API-et ble designet av Dr. Roy Fielding i år 2000, der han definerte seks hovedprinsipper. [63]

1. **Uniformt Grensesnitt** - All informasjon i systemet skal være separert fra hverandre og kun kunne mottas ved én URI.
2. **Klient-server-forhold** - Ved bruk av REST skal det være et klart klient-server-forhold. Med dette betyr det at det er klienten som velger hvilke operasjoner som gjøres på serveren og ikke omvendt
3. **Tilstandsløs** - Forespørsler til serveren skal ikke ha sammenheng med andre forespørsler. Hver forespørsel skal behandles som ukjent for serveren.
4. **Bufferbar** - All data som sendes fra serveren skal kunne lagres i en buffer, slik at data ikke trengs å lastes inn to ganger. Dersom serveren har tjenester over større skala vil dette også være fordelaktig dersom data må sendes over lengre strekninger.
5. **Støtte for lagdelt system** - At serveren har mulighet for å ha et lagdelt system gjør at det er ukjent for klienten hvor data er lagret.
6. **Sending av kjørbar kode** - Serveren skal ha mulighet til å sende kode som er kjørbar på klienten. Dette siste prinsippet er valgfritt å implementere derimot, da man i mange applikasjoner ikke har behov for det.

,

2.9.3 Internet Protocol

For å kunne levere data fra serveren, benytter vi av oss Internet Protocol (IP) for å "fortelle" andre enheter hvor de kan nå serveren. IP er en protokoll for å sette en adresse på alle enheter som er koblet opp på internett[69]. Det er viktig å ha en slik protokoll, siden internettet består av mange undernettverk

som utgjør et større nettverk. En adresse innenfor IP kalles en IP-adresse, som består av en rekke tall som en ruter bruker for å velge hvor data, kalt en pakke, skal sendes til. For å kommunisere med en server for å spørre om data gjennom HTTP 2.9.1, angir man IP-adressen til denne serveren.

2.9.4 Transport Layer Security

En reell fare ved bruk av internett er at brukere kan tyvlytte på data som sendes mellom klient og server dersom de er koblet til internettet til en av partene. Dette er spesielt problematisk dersom f.eks. passord sendes til en server fra en klient. For at slik tyvlytting ikke skal være mulig på kommunikasjon til vår webserver bruker vi sikkerhetsprotokollen Transport Layer Security (TLS). TLS krypterer pakker før de sendes, for å legge til et ekstra sikkerhetsledd innenfor internettet [70]. Ved implementasjon av TLS, bruker man HTTPS istedenfor HTTP 2.9.1 for å indikere at siden benytter seg av denne protokollen. S-en står for Secure.

2.9.5 File Transfer Protocol

File Transfer Protocol (FTP) er en protokoll for overføring av filer fra en server til en klient, eller motsatt[71]. I prosjektet vårt har vi brukt FTP for å overføre filer mellom servere og lokal datamaskin. For eksempel måtte vi laste over lydfilene fra server etter at lydopptak ble spilt inn via spørreskjemaet vårt, som vi enkelt kunne gjøre ved bruk av FTP.

2.9.6 Secure Shell

Da vi ikke har fysisk tilgang til serverene vi har brukt i prosjektet vårt, bruker vi Secure Shell-protokollen (SSH) for kommunisere med disse. SSH er en protokoll som krypterer alle handlinger gjort med serveren for å forhindre angrep. [72] Det gjør at man har mulighet til å logge seg på og hente ut/modifisere data uten at andre har mulighet til å se hvilke kommandoer som blir sendt over nettet.

2.9.6.1 SFTP

For å gjøre overføring ved bruk av FTP 2.9.5 mindre sårbart mot dataangrep, benytter vi oss av SFTP (Secure File Transfer Protocol). Det har samme prinsipp som FTP, men er istedenfor bygget på toppen av SSH for å gi et ekstra sikkerhetsledd ved å kryptere data som sendes [77]. Spesielt viktig er bruken av SFTP når vi sender inn lydfiler fra Raspberry Pi til server for prediktering av talen, slik at det er vanskeligere å stjele lyden.

2.10 Datalagring

Lagring av data har vært et viktig aspekt for prosjektet vårt. Taledata må f.eks. lagres på serveren over tid, slik at maskinlæringsmodeller kan trenes opp på nytt dersom det gjøres endringer i deres interne struktur, eller dersom det

legges til eller fjernes data som modellen skal trene på. I tillegg må taledata kobles opp mot annen brukerinformasjon, slik som innloggingsinformasjon for administratorer, osv.

Hvordan vi skal lagre data på server og de ulike delkomponentene som utgjør systemet som en helhet, var tidlig oppe for vurdering i prosjektfasen. Vi kunne f.eks. ha lagret all informasjon direkte på filsystemet til serveren, men dette byr på flere ulike problemer. Ved å gjøre dette risikerer vi uoverensstemmelser mellom data i ulike filer som på et vis eller annet hører sammen, manipulering og uthenting av data vil være vanskelig og det vil være utfordrende dersom flere delkomponenter (eks. Raspberry Pi og server) trenger tilgang til samme data. Med tanke på sikkerhet ville denne framgangsmåten også være problematisk. Vi landet derfor på å bruke databaser for datalagring.

En database er en samling med data som er organisert slik at den er enkel å lese, skrive til og organisere. Det finnes flere ulike typer databaser (hierarkiske, objekt-orienterte, NoSQL, etc.) som alle er gode til sitt bruk, men etter diskusjon med oppdragsgiver valgte vi for dette prosjektet å bruke relasjonelle databaser.

Relasjonelle databaser organiserer data i tabeller som linkes til hverandre gjennom relasjoner, basert på felles data mellom tabellene. Dette gjør det enkelt å hente ut sammenhørende data fra forskjellige tabeller med en enkelt spørring mot database, samt å holde oversikt over hvilken data som jobber/hører sammen [76]. Da vi lagrer data som er logisk å lagre som forskjellige enheter, men som allikevel hører sammen (eks. opptak for registrering og generell informasjon om bruker, tokens for sikkerhet som knyttes opp mot en bruker (4.6.6), etc.) var relasjonelle databaser derfor et naturlig valg. I tillegg har vi fått fordypning i relasjonelle databaser gjennom fag på NTNU, og vi har begge god erfaring med databasetypen fra tidligere prosjekter. Relasjonelle databaser er også den klart mest brukte formen for databaser, slik at ved eventuell videreutvikling vil dette sannsynligvis være den formen for database det er enklest for utviklere å forholde seg til [4].

2.11 Datasikkerhet

I forbindelse med at vi har utviklet en administrativ nettside for prosjektet, har data- og websikkerhet vært et viktig konsept å forholde seg til. Vi lagrer innloggingsinformasjon, brukerdata, osv., for ikke å nevne at hele systemet relaterer seg til et så sikkerhetskritisk aspekt som et låsesystem. Det har derfor vært viktig å sikre det utviklede produktet fra utilsiktet bruk, samt sikre trygg lagring av data og trygg bruk av nettsiden. Under følger beskrivelser av konsepter innenfor datasikkerhet som har vært viktig å tenke på når vi har utviklet systemet, samt hvordan de ulike konseptene relaterer til vårt produkt.

2.11.1 Proxy

En proxy-server fungerer som et mellomledd mellom den som bruker den og internett. Når en bruker ruter trafikken sin gjennom en proxy-server, flyter all trafikk som var ment til den spesifiserte nettadressen (f.eks. www.vg.no) gjennom proxy-serveren på vei til den spesifiserte adressen. Når serveren for den spesifiserte adressen så svarer på forespørselen, vil denne dataen blir sendt tilbake til proxy-serveren før den videresendes til brukeren. [13] Selv om en proxy i seg selv kan være et nyttig verktøy for bl.a. webutvikling, er en proxy-server er et vanlig verktøy for hackere og andre brukere med onde hensikter på internett. En angriper kan nemlig bruke en proxy-server til å fange opp forespørselen han/hun sender, etter at den har passert sikkerhetssjekker som er plassert på klientsiden med f.eks. JavaScript. For oss, og andre som utvikler nettsider, er det derfor imperativt at de sjekkene som gjennomføres av JavaScript eller lignende på klientsiden kun er ment for å gjøre siden mer brukervennlig, i tillegg til at all innkommende data til serveren blir validert på serversiden før den prosesseres.

2.11.2 SQL-Injeksjon

SQL er en fellesbetegnelse for spørrespråk som benyttes for å kjøre operasjoner på relasjonelle databasesystemer (2.10). En SQL-Injeksjon er en teknikk som brukes for å angripe webapplikasjoner ved å endre innholdet på SQL-spørringer som kjøres på serversiden, og dermed muligens data som returneres til angriperen via nettsiden eller annen logikk som tillater en angriper å utnytte applikasjonen. Sårbarheter for SQL-Injeksjoner oppstår som regel ved at webapplikasjonen inkluderer inn-data fra brukeren direkte i SQL-spørringer som kjøres på serversiden. F.eks. vil en webapplikasjon ofte logge inn brukere ved å slå opp brukernavn og passord i databasen etter input fra brukeren. Dette vil normalt sett manifestere seg i en SQL-spørring som ser omtrent slik ut:
"$SELECT * FROM\ users\ WHERE\ username =' foo'\ AND\ password =' foo\_pass'$"

Der "foo" er levert brukernavn og "foo_pass" er levert passord fra brukeren. Dersom applikasjonen inkluderer levert data fra brukeren rett i SQL-spørringen uten noen form for sikkerhetssjekk eller formatering av dataen, kan en angriper for eksempel logge inn med en hvilken som helst bruker dersom han/hun leverer følgende passord "$'foo'\ OR\ 1 = 1$". Den endelige SQL-spørringen vil da bli seendes slik ut:

"$SELECT * FROM\ users\ WHERE\ username =' foo'\ AND\ password =' foo'\ OR\ 1 = 1$"

Denne spørringen vil alltid returnere relasjonen for brukeren med brukernavn "foo", fordi $1 = 1$ alltid er sant [14]. Sikkerhetsaspektet med SQL-Injeksjoner har vært viktig å vurdere for oss i forbindelse med utvikling i den administrative nettsiden, da vi har vært nødt til å inkludere bruker-levert data i SQL-spørringer

ved flere anledninger - f.eks. ved innlogging, tilbakestilling av passord, token-validering, osv.

2.11.3 CSRF

Cross-Site Request Forgery (CSRF) er en annen teknikk som ofte brukes for å angripe webapplikasjoner. CSRF utnytter brukerenes tillit til webapplikasjonen, ved å tvinge brukerene til å gjennomføre forespørsler som de ikke var ment å gjennomføre[15]. Dette kan manifestere seg på forskjellige måter, men under følger et typisk eksempel:

En angriper kan ha klart å laste opp data (f.eks. et brukernavn) til applikasjonen som blir gjengitt som HTML når det vises på nettsiden. Dersom applikasjonen f.eks. tillater
" $< /p ><$ *img src* = "*https* : *//www.app.com/changepassword?new_pass* = *foo*" $>$
som et gyldig brukernavn, og brukernavnet blir gjengitt i en paragraf på websiden uten at det gjøres ordentlig sikker formatering av data som fremvises, vil det gjengitte brukernavnet manifestere seg i form av et bilde som inkluderes i HTML-koden. Dersom en administrator så f.eks. laster inn oversikten over brukere, vil nettleseren hans/hennes gjøre en automatisk HTTP-GET til
https : *//www.app.com/changepassword?new_pass* = *foo*.
Hvis applikasjonen så tillater at endring av passord gjøres gjennom HTTP-GET, vil angriperen ha lykkes med å endre passordet til administratoren til "foo".

Dette har også vært et relevant sikkerhetsproblem for oss, da eksempelet over viser en eksakt problemstilling vi har vært nødt til å håndtere i utviklingen av vår applikasjon. Dette fordi en administrator ofte vil fremvises data som er levert av brukere, i form av navn, eposter, osv.

2.11.4 XSS

Cross-site Scripting (XSS) er en angrepsteknikk som er nært beslektet med CSRF (2.11.3). Her utnytter angriperen webapplikasjonens tillit til brukerne ved å laste opp ondsinnet kode som blir kjørt i nettleseren til andre brukere, som regel i form av JavaScript. Et vanlig eksempel vil være at angriperen laster opp en kommentar som ikke formateres riktig før den inkluderes i kommentarfeltet på nettsiden som angripes. Kommentaren kan være skrevet slik at den da manifesterer seg i form av JavaScript i nettleseren, istedenfor ren tekst i kommentarfeltet. Når andre brukere senere går til kommentarfeltet, vil JavaScript-koden som er lastet opp av angriperen kjøre i brukerens nettleser. En vanlig angrepsteknikk vil være å få JavaScript-koden til å hente ut cookies (f.eks. session-tokens) som hører til offeret, og sende disse til angriperen ved hjelp av XmlHttpRequest i JavaScript [73]. Den administrative websiden inkluderer i flere tilfeller data i HTML, som opprinnelig er levert av brukere. Av denne grunnen har vi også vært nødt til å vurdere sikkerhetstrusler som XSS når vi har utviklet den administrative nettsiden.

2.11.5 Brute Force

Brute-Force angrep er en annen vanlig angrepsvektor for hackere og andre brukere med onde hensikter. Angrepet går i hovedsak ut på å automatisere utsending av mange forespørsler til en server med varierende verdier, i håp om at noen av forespørselene vil gi verdifulle resultater. Et vanlig eksempel er Brute-Force angrep mot en portal for innlogging, der en angriper gjerne bruker en stor liste over vanlige passord til å sende inn store mengder med innloggingsforsøk for en bruker. Ved å overvåke responsene fra serveren kan angriperen avgjøre hvorvidt et innloggingsforsøk var gyldig, og dermed hvorvidt han/hun lyktes i å "gjette" brukerens passord. Med tanke på at vår webapplikasjon for administratorer også implementerer bl.a. en portal for innlogging, har dette også vært et sikkerhetsaspekt vi har vært nødt til å vurdere hvordan vi skal forsvare oss mot.

2.11.6 POLP

Principle of least privilege (POLP) er et sikkerhetsprinsipp for utvikling som i hovedsak går ut på at hver modul eller enhet for et datamiljø kun skal gis tilgang til de ressursene som er nødvendig for at enheten eller modulen skal kunne gjennomføre den jobben den er satt til å gjøre. Vi har gjennomgående brukt dette prinsippet i applikasjonen for å heve sikkerheten. Det har f.eks. vært viktig å sikre at brukeren som brukes til å gjøre interaksjon mot databasen kun har tilgang til de nødvendige tabellene, og at dataobjekter som brukes for å prosessere lydfiler fra filsystemet kun har lese-rettigheter til mappene som inneholder lyddata.

3 Materialer og Metode

Dette kapittelet omhandler hvordan prosjektet for oppgaven er satt opp. Dette innebærer alt fra det organisatoriske og organisasjonsstrukturen til utviklingsmiljøet for de underliggende delsystemene. Her beskriver vi også hvilke språk og verktøy som er brukt i oppgaven.

3.1 Organisering

Oppgaven er delt opp tre forskjellige organisatoriske parter. Her beskriver vi hvilke roller de forskjellige partene i oppgaven har.

3.1.1 Prosjektgruppe

Prosjektgruppen består av to studenter ved NTNU i Ålesund, Fredrik F. Waaler og Klaus Dyvik. Studentene utgjør oppdragstaker innenfor prosjektet, og er de som skal utrede selve oppdraget.

3.1.2 Oppdragsgiver

Oppdragsgiver i prosjektet er Avento AS [119]. Det er denne bedriften som stiller krav til prosjektgruppen om hvordan det endelige systemet skal foreligge. Avento forvalter en kontaktperson, Oskar Emil Skeide, som er bindeleddet mellom prosjektgruppen og bedriften. Videre informasjon om oppdragsgiver finnes i forordet til oppgaven.

Under startfasen av prosjektet holdt vi et møte med oppdragsgiver for å diskutere hvordan vi skal kommunisere gjennom prosjektets gang. Prosjektgruppen påtok seg ansvar for å invitere til møter ved de tidspunkt der det var naturlig å involvere oppdragsgiver (ved veiskiller, etter endt milepæl, osv.), eller der det ellers var nødvendig med input fra oppdragsgiver. I starten var hovedfokuset for møtene planlegging av prosjektet, der vi har diskutert hvordan de vil at oppgaven skal løses. Senere møter har hatt større fokus på kvalitetsikring og input fra oppdragsgiver - er f.eks. det vi har laget så langt slik oppdragsgiver hadde tenkt seg? Det har ligget til grunn at vi har vist progresjon i det endelige produktet, som har vært mer relevant i de siste fasene av prosjektet der de mer visuelle funksjonalitetene har funnet sted (f.eks. grensesnitt for administratornettsiden). Vi har ellers oppdatert oppdragsgiver kontinuerlig med arbeid i oppgaven, slik at de har hatt mulighet til å ta del i alle beslutninger og ivareta sin interesse som oppdragsgiver.

Kontaktperson ved Avento jobber til dels hos bedriften og til dels på Campus Ålesund, så møtetidspunkt er planlagt ut fra hans agenda og ettersom hvor han har befunnet seg. Da medlemmene i prosjektgruppen er fleksible har dette ikke vært et problem. Sammen har vi laget et rom i kommunikasjonsplatformen Microsoft Teams, der vi kan stille spørsmål og ivareta kontakten. Da landet ble

stengt ned for å hindre smittefaren for Covid-19, ble all kontakt flyttet over til Microsoft Teams, og fysiske møter ble erstattet med videomøter der.

3.1.3 Veiledere

I tillegg til de foreliggende partene i prosjektet er også prosjektgruppen bistått med veildere fra universitetet. Rollen til veilederne er å sørge for at oppgaven går fremover og hjelpe prosjektgruppen med problemløsing ettersom det er nødvendig. Fordelen med å ha en gruppe veiledere er at man får utvidet prosjektgruppen til å være nærmere slik det er i arbeidslivet, der man ofte drøfter problem ovenfor en større gruppe.

Som vi har skrevet om i 3.6, har vi benyttet oss av den agile metodikken Scrum i oppgaven. Sprintene våre har vært på lengde av to arbeidsuker, som har gjort at det har vært naturlig å ha møter med veiledere etter at disse er endt. Her gjennomgår vi status på oppgaven slik den står på det avtalte møtepunktet, og ser etter løsninger på eventuelle problemer. Ofte har prosjektgruppen sendt materiale til veilederene på forhånd for gjennomsyn, slik at veilederen får gitt best mulig kritikk under møtet. Som oftest har møtene blitt arrangert klokken 11.30 annenhver torsdag da dette har passet for både veiledere og gruppemedlemmer. Møtene tok i utgangspunktet sted på et kontor ved universitetet, men ble senere flyttet til Skype for Business som følge av Covid-19.

3.2 Planlegging av prosjekt

Før vi begynte på oppgaven hadde vi gjennom et annet emne ved universitetet et obligatorisk arbeid som dekket prosessen for forarbeid i oppgaven. Oppgaven resulterte i en forprosjektrapport, som er lagt ved som vedlegg 7.1. Rapporten er basert på en mal forvaltet av foreleser i emnet, som tar for seg en rekke punkter som kan være greie å vurdere under planlegging av et prosjekt. I begynnelsen definerte vi selve prosjektorganisasjonen slik som i 3.1. Deretter definerte vi ulike avtaler og normer vi hadde innad i prosjektgruppen for å sikre at prosjektgruppen arbeider konsistent med oppgaven.

Det neste kapittelet i rapporten hadde oppdragsgiver i større grad innvirkning på. Dette er selve prosjektbeskrivelsen og hva som var planlagt å gjøres. Da vi fikk tildelt denne oppgaven lå det ved en liten beskrivelse av prosjektet, som var et utgangspunkt for hva oppgaven innebar. Ut fra beskrivelsen kunne vi drøfte tanker om hvilke muligheter vi har ut ifra tidsomfang og kompetanse innenfor fagfeltet. Etter å ha åpnet for dette, planla vi et møte med oppdragsgiver for å diskutere hvilke konkrete krav de stilte til oppgaven. Ut ifra dette møtet kunne vi utrede en prosjektbeskrivelse der vi detaljert beskrev hva som inngikk i oppgaven. Denne ble igjen diskutert med oppdragsgiver, slik at begge sider var enige om utfallet. I tillegg til prosjektbeskrivelsen utredet vi forskjellige diagram som skulle bidra med at oppdragsgiver kunne forstå hva vi hadde tenkt. Et deployment-diagram viste hvordan det endelige systemet

skulle være, og et UML-diagram viste hvordan brukeropplevelsen skulle være. En kravspesifikasjon var også utredet for å dekke de funksjonelle kravene. Disse diagrammene er også lagt ved som vedlegg for rapporten i vedlegg 7.5.

I forprosjektrapporten er det videre beskrevet planen for møter, både med veiledere/oppdragsgiver og innad i gruppen. Under avviksbehandling ble det definert hva som skulle skje dersom uforutsette hendelser skulle oppstå under prosjektet. Avslutningsvis framla vi en liste over utstyrsbehov for å gjennomføre oppgaven.

3.3 Kvalitetssikring

I forprosjektrapporten (Vedlegg 7.1), ble ulike aspekter ved prosjektet definert for å sikre at kvaliteten ivaretas gjennom hele prosjektet. I seksjon 3.1 om prosjektgruppen beskrev vi hvilke roller gruppemedlemmene hadde innad i prosjektet. Siden vi kun er to personer som jobber med oppgaven, ble organisatorisk ansvar ikke fordelt ut ifra enkeltmedlemmer, men definert som et felles ansvar. Dette for å legge til rette for at vi tar felles beslutninger som igjen sikrer god kommunikasjon. Kapittel 4 - Avtaler dekket hvilke avtaler vi har i gruppen for arbeidstider og gruppenormer. Det beskriver at vi har gode holdninger opp mot prosjektet, og skal samarbeide for å løse oppgaven mest mulig.

Her har vi definert at vi skal bruke arbeidsmetodikken Scrum, beskrevet i dypere detalj i 3.6. En viktig grunn for det er at det gjør at vi kontinuerlig diskuterer status på oppgaven under daglig scrum-møte. Det gjør at begge gruppemedlemmer til en hver tid er oppdatert på hverandres arbeid og er med på å sikre at begge gruppemedlemmer har en helhetlig forståelse av alle aspekter ved prosjektet. Ved endt sprint skriver vi også rapport om perioden og tar en større vurdering på hva som er viktig til neste periode. Etter sprintslutt er også et møte med veiledere satt opp for å få deres perspektiv på status, som gjør at vi sikrer at vi har god tid til å gjennomføre prosjektet. Sprint-rapporter er lagt ved som vedlegg 7.6.

3.4 Avtalte kriterier for fullført oppgave

I kravspesifikasjonen funnet i vedlegg 7.2 ligger diverse krav vi har utredet for oppdragsgiver som skal dekke hvilke funksjonalitet som skal inngå i det ferdige systemet. Disse er utarbeidet innad i prosjektgruppen på bestilling fra oppdragsgiver. I tillegg til selve systemet har vi i utgangspunktet avtalt med bedriften at systemet skal settes opp utenfor døren til kontorarealet deres slik at det er klart til bruk.

Siden det har oppstått komplikasjoner som følge av Covid-19 endret planen seg, ettersom gruppemedlemmene ikke får tilgang til deres kontor så lenge pandemien vedvarer. Derfor har vi etter dialog med oppdragsgiver blitt enige at vi istedenfor setter opp et testlokale hjemme hos et av gruppemedlemmene, som skal simulere inngangsdøren til bedriften. Videre sender vi det ferdige produk-

tet over post til bedriften med en installeringsmanual slik at de kan ta i bruk denne.

3.5 Forventet dokumentasjon

Innenfor prosjektet er det forventet at følgende dokumentasjon fremlegges:

- Kravspesifikasjon - For enighet om funksjonalitet som prosjektet inkluderer
- UI Mockups - For design av brukergrensesnitt
- UML Diagram - For visning av hvordan systemet i sin helhet fungerer
- Deployment diagram – Oversikt over de ulike komponentene i prosjektet
- Versjonsliste - Liste over hvilke versjoner av all programvare og utvidelser brukt

Dokumentasjonen som er forventet er også beskrevet i forprosjektrapporten. På grunn av at installering av det ferdige systemet ikke er mulig under pandemien, er det også forventet at et dokument for oppsett av systemet utarbeides.

3.6 Utviklingsmetodikk - Scrum

Arbeidsmetodikken vi har brukt i denne oppgaven er kalt for Scrum. Det er en av de mer populære agile arbeidsmetodikkene (2.7). Metodikken er veldig fleksibel, men har et sett av regler som må overholdes for at det skal bli sett på som Scrum.

3.6.1 Sprint

Innenfor Scrum holder man fokus på en gitt tidsperiode. Denne perioden kan variere stort, men det er vanlig å sette denne innenfor to til fire uker for å opprettholde det agile konseptet. Disse tidsperiodene kaller man for en sprint. Før en sprint starter blir man enige innad i prosjektgruppen om hva som skal fullføres innen sprinten er ferdig. Et prosjekt har gjerne en backlog over arbeidsoppgaver som skal utføres i prosjektet, og ut ifra disse velger man ett sett oppgaver som dekker arbeidsmengden for en sprint når den startes. I sprinten skal man så for det meste kun ha fokus på disse oppgavene. Det er mulig at arbeidsmengden enten er for stor eller for liten, så laget må diskutere hvorden sprinten har vært. Derfor har man i ettertid av enhver sprint et retrospektivmøte der man diskuterer hvordan det har gått i den foregående sprinten slik at man kan tilpasse seg til den videre prosessen.

3.6.2 Daily Standup

Selv om man vet hva som skal gjøres i en Sprint, og man gjerne har definerte arbeidsoppgaver er det fortsatt viktig med kommunikasjon innad i prosjektgruppen. Derfor arrangeres det et kort møte på morgenen der alle i teamet forteller litt om hva man har jobbet med siden sist, om det har oppstått noen problemer, og hva planen til vedkommende er ut dagen. Det gjør at alle har mulighet for å drøfte om sin egen innsats samt få andre perspektiver på problemer man sliter med. I tillegg er det greit for andre medlemmer i laget å få oppdatert status på hvordan man ligger an i forhold til å overholde fristen på Sprinten, og det hjelper hele gruppen å opprettholde et helhetlig perspektiv på oppgaven.

3.6.3 Backlog

En backlog er en oversikt over alle arbeidsoppgavene som må gjøres for at prosjektet skal kunne sies å være ferdige. Hver av disse oppgavene har et endelig mål som definerer når oppgaven ansess som ferdig. I tillegg tilegnes oppgavene prioritering, tidsestimater og eventuelle kommentarer som skal hjelpe den som gjennomfører arbeidsoppgaven. Backlog skal kontinuerlig oppdateres av gruppemedlemmene for å gjenspeile hvilke arbeidsoppgaver som har oppstått, hvilke det jobbes på og hvilke som er ferdige.

3.6.4 Roller

Når man bruker Scrum er det noen ganger veldig åpent hvilke oppgaver folk holder på med. Det gjør at man trenger litt overordnet ansvar. Derfor er det et par spesifiserte roller som hvert Scrum-lag har tildelt medlemmene.

3.6.4.1 Produkteier

En produkteier skal representere eieren av produktet. Eieren skal stille krav til Scrum-laget for å sikre at de viktigste oppgavene blir gjort. Rollen inngår også i å ta kritiske beslutninger om hvilke funksjoner som skal bli prioritert, slik at det endelige systemet blir tilpasset slik han/hun ønsker det. Selv om produkteier representerer eieren, trenger ikke nødvendigvis denne personen å være den personen som bestilte oppgaven. Grunner til dette er at i mange utviklingsprosjekt er oppdrag gitt av folk uten erfaring innenfor systemutvikling. Derfor må produkteieren klare å representere oppdragsgivers meninger, men på en måte som lar seg oversette til tekniske krav og funksjonalitet som utviklerene kan implementere.

3.6.4.2 Scrum Master

Scrum i seg selv har ikke en "sjef", men det er fortsatt ofte en person som blir tilegnet tittelen Scrum Master. Denne personen har et overordnet ansvar for at Scrum-prosessen blir gjennomført slik den skal. Vedkommende har også ansvar for at backloggen er ryddig, og holder et øye på progresjonen i laget og sørger for at gruppen overholder de tidsfrister som er satt.

3.6.5 Våre retningslinjer

Siden vi i dette prosjektet bruker scrum som metodikk har vi satt noen retningslinjer for hvordan det skal utføres. Som en pekepinne har vi satt sprintlengden til å være på to arbeidsuker, som etter enighet virket som den mest naturlige tidsperioden å jobbe med oppgaven etter tidligere erfaring, siden vi kontinuerlig kan revurdere status på oppgaven.

Hver morgen har gruppen stand-up møte der medlemmer forteller hva de gjorde den foregående dagen, om de har hatt noen problemer, og hva som medlemmet skal jobbe med den inneværende dagen. Dette gjør at begge medlemmer alltid vet hva den andre driver på med, slik at god kommunikasjon er ivaretatt.

På slutten av hver sprint blir en sprint-rapport utredet, som inneholder hva gruppen har drevet med den siste sprinten. Dette er for å oppdatere medlemmene om hvor mye som ble gjort i perioden, både for motivasjon og tidsestimering av den neste sprinten. I tillegg blir rapporten sendt til veiledere for å oppdatere dem på status før møtet.

3.7 Programmeringsspråk

For å få på plass den endelige løsningen slik den er forespurt av Avento, måtte vi velge oss ut en knippe programmeringsspråk og lignende teknologier som skulle brukes til å skrive løsningen.

3.7.1 Python

I oppgaven har vi i hovedsak brukt programmeringsspråket Python. Python er et programmeringsspråk med støtte for mange av programmeringsparadigmene, inkludert prosedyrisk, objektorientert og funksjonell programmering. Det gjør at vi effektivt kan bruke Python både for utvikling i forbindelse med administratorsystemet og for talegjenkjenningen. Språket er et høynivå språk, som gjør at det er veldig fleksibelt og som bidrar til at man enkelt kan gjøre mindre endringer uten store konsekvenser, med at man f.eks. ikke må rekompilere hele applikasjonen. Språket har i de siste årene vært veldig utbredt, som gjør at det eksisterer en rekke pakker laget av andre personer som kan være behjelpelig i prosjektet [78]. I tillegg er Python er utbredt språk for vitenskapelige prosjekter som f.eks. omhandler maskinlæring osv. [79]. Dette gjør at det allerede finnes mange bibliotek med eksisterende funksjonalitet for maskinlæring laget til Python. Da begge gruppemedlemmene føler seg komfortable med Python, og også har tidligere erfaring med utvikling av nettsider i dette språket, var Python er naturlig valg når det kom til programmeringsspråk.

3.7.2 HTML

Under prosjektet har vi hatt behov for å lage diverse nettsider. For å lage innholdet for disse har vi brukt *Hypertext Markup Language*, som er et mark-

eringsspråk som er standard for nettlesere i dag. Det brukes for å strukturere hvor på en nettside informasjon befinner seg. Informasjon blir plassert inne i elementer som kan flyttes rundt på ettersom hvor informasjon skal ligge i forhold til hverandre.

3.7.3 CSS

For at nettsidene våre både skal være brukervennlige og se bra ut visuelt sett, har vi brukt *Cascading Style Sheets*. Dette er en standardisert måte å designe innhold på, og er designet til bruk av bl.a. HTML. CSS er mest brukt i webutvikling, men kan også brukes i andre språk. Ved et sett av regler kan man definere egenskapene til et HTML-element, slik at man får ønsket størrelse, farge, plassering, eller lignende. Med tanke på at administratornettsiden har et universelt design på tvers av de ulike sidene, kommer også CSS til god bruk ved at vi slipper å spesifisere det samme designet på nytt for hver av sidene.

3.7.4 JavaScript

Vi har også hatt bruk for diverse dynamisk funksjonalitet på tvers av administratornettsiden. Siden HTML/CSS kun har støtte for markering og design, har vi tatt i bruk JavaScript til å innføre den funksjonaliteten vi trenger på klientsiden - da dette er et standardisert språk som prosjektgruppen fra før av har god erfaring med. JavaScript er et språk som også har støtte for flere av programmeringsparadigmene. Det støtter blant annet objektorientert og funksjonell programmering, som gjør at det er veldig fleksibelt. Språket er standardisert gjennom ECMAScript som er en spesifikasjon for interaksjonsspråk i webutvikling [97]. JavaScript har historisk vært brukt til webutvikling for å få støtte for interaksjon på en nettside, men har i senere tid også blitt brukt mer og mer til lokale applikasjoner.

3.8 Teknologier

I denne delen beskriver vi ulike teknologier brukt i oppgaven for å ellers hjelpe oss med å løse problemstillingen. Slike verktøy er f.eks. brukt for å sikre versjonskontroll av koden vår eller datalagring.

3.8.1 Git

Siden vi jobber på prosjektet sammen har vi hatt behov for å utvikle på samme prosjekt. For å løse dette har vi valgt å bruke Git grunnet på tidligere erfaring med teknologien. Git er et versjonskontrollsystem som brukes under systemutvikling [41]. Endringer i programkode publiseres i en *commit*, som har en beskrivelse av hvilke endringer som blir gjort. På denne måten kan man gå tilbake til tidligere løsninger dersom noe skulle gå galt. Ofte legger man filene på en server slik at mange mennesker kan jobbe sammen på samme prosjekt. Det finnes alternativer til Git, som f.eks. SVN og Mercurial, men da begge gruppemedlemmer fra før av har erfaring med Git, og dette er den klart mest bruke

formen for versjonskontroll, så vi ikke et behov for å vurdere andre løsninger [80].

3.8.1.1 GitHub

En av skytjenestene som lagrer filene på server er GitHub[42], som vi har brukt i vårt prosjekt for å kunne utvikle sammen. I tillegg til å kunne jobbe på det samme prosjektet åpner GitHub for at andre mennesker kan laste ned prosjektet. Det åpner for at veileder lett kan få tilgang til kildekode.

3.8.2 AJAX

En annen måte vi har kommunisert informasjon er ved bruk av AJAX. AJAX er en fremgangsmåte som tillater asynkron utveksling av data fra nettsider ved å utveksle data mellom nettleser og serveren i bakgrunnen. Dette gjør det mulig å oppdatere deler av en nettside eller gjøre innsendinger til en server uten å måtte laste inn en nettside på nytt. AJAX er ikke en teknologi i seg selv, men heller en samling av teknologier som muliggjør asynkron datautveksling. [9]

3.8.3 PostgreSQL

I systemet har vi trengt en database for å lagre informasjon om brukere, og valget falt på PostgreSQL. Da vi fra tidligere prosjekter har god erfaring med PostgreSQL, og vet at det finnes gode biblioteker som gjør det enkelt å bruke PostgreSQL med Python, var dette et naturlig valg. I tillegg hadde ikke oppdragsgiver noen spesifikk preferanse for databasesystem, slik at det var åpent for at vi kunne velge dette selv. PostgreSQL er et gratis databasesystem for relasjonelle databaser (2.10), med åpen kildekode. Interaksjon med en PostgreSQL-database skjer ved hjelp av Postgres sitt eget spørrespråk, som er en variant av tradisjonell SQL.

3.9 Eksterne Biblioteker og lignende

Vi har brukt en rekke eksterne biblioteker og utvidbare moduler for å utvide funksjonaliteten til Python og JavaScript, slik at vi kan imøtekomme kravene som stillles til vårt prosjekt. Eksempler på slike biblioteker og moduler er TensorFlow og Keras for maskinlæring, Flask og Jinja for utvikling av webapplikasjonen, samt SciPy og Numpy for databehandling og lignende. Under følger en kort beskrivelse av de ulike bibliotekene/modulene vi har innlemmet i prosjektet, og en liten forklaring hvilken rolle de spiller i prosjektet som helhet.

3.9.1 TensorFlow

Ved utredning av maskinlæringsmodeller har vi brukt rammeverket Tensorflow [34]. TensorFlow er et open-source bibliotek for maskinlæring som er utviklet av Google. Det åpner for at man kan enkelt kan sette opp sine egne modeller av nevrale nettverk. Biblioteket er laget for å trene data raskest mulig ved hjelp

av *Tensors*, som er et matematisk objekt av mange dimensjoner, eller variabler. Vi har også vurdert andre alternative rammerverk for maskinlæring, som f.eks. PyTorch, men landet på TensorFlow da dette er et mer modent rammeverk med mer viden støtte for utplassering på server, noe som var nødvendig for vårt prosjekt [81].

3.9.2 Keras

Når man bruker TensorFlow har man behov for å definere modeller og lag. Keras [35] er et av *deep learning*-rammeverk som er kompatible med TensorFlow, og også det mest prefererte. Rammeverket har en rekke ulike lag man kan bruke i sine nevrale nettverk. I tillegg inneholder det aktivasjonsfunksjoner og optimaliserere for å kunne justere data. Keras har også funksjoner for preprosessering av lydfiler, som kan være nyttig i noen sammenhenger.

3.9.3 TQDM

Opplæring og testing av maskinlæringsmodeller tar ofte lang tid og det kan derfor være greit å få tilbakemelding underveis, slik at vi vet hvordan prosessen ligger an. Biblioteket TQDM har her gitt oss god hjelp. Biblioteket visualiserer en fremdriftslinje for ulike innebygde kontrollstrømmer i Python, som for- og while-loops. Biblioteket gir også en prediksjon på gjenstående dette. Dette har vært med på å effektivisere arbeidet med maskinlæringsmodeller, fordi vi da uforstyrret kan drive med andre ting imens modellene trener, uten å måtte sjekke kontinuerlig hvorvidt opplæring og/eller testing av modellen er ferdig.

3.9.4 Google Cloud

Google Cloud er en platform fra Google som er et samlepunkt for skytjenester levert av firmaet. Platformen inneholder en rekke ulike tjenester, som for eksempel verktøy for dataanalyse, utviklingsverktøy og kunstig intelligens. Tilgang for tjenester er ofte gratis i begynnelsen, slik at det er mulig å teste ut diverse API i forkant.

3.9.4.1 Speech-To-Text

For å finne ut av om en spesifikk frase er sagt i en lydfil bruker vi tjenesten Speech-To-Text fra Google Cloud. Dette et API som brukes til å omdanne tekst til tale[46]. Man kan bruke API-et ved å enten sende inn en lydfil, eller sende inn en strøm med lyd for å få direkte tilbakemelding. Tjenesten er kompatibel med flere språk, inkludert norsk.

3.9.5 Librosa

Når vi skal hente ut karakteristikker fra en stemme bruker vi pakken Librosa. Librosa er en Python-pakke for analyse av musikk og lyd[43]. Pakken inneholder metoder for å hente ut informasjon fra en lydfil, og endre en lydfil til ønskede

faktorer. Det er mulig å hente ut en rekke egenskaper som viser ulike karakteristikker av lyden, og utfører diverse matematiske omforminger for brukeren. Vi bruker Librosa til å få omgjøre lydfiler til en felles samplingsrate, og for å hente ut MFCC-verdier fra filene.

3.9.6 PyAudio

For å kunne ta opp lyd i Python har vi brukt PyAudio. PyAudio er et bindeledd mellom Python og I/O på datamaskinen[44], som gjør at vi gjennom Python kan ta bruk av mikrofonen på den lokale enheten der koden kjøres.

3.9.7 PyDub

For å omforme lyden til et felles dBFS-nivå, som er en måleenhet for lydvolum, har vi brukt pakken PyDub. Denne bruker en klasse AudioSegment, som inneholder ulike metoder for å endre på lyden til en .wav-fil [39].

3.9.8 PyLoudNorm

Selv om vi har brukt PyDub (3.9.6) til å omforme lydfiler til et felles dBFS-nivå, har det også vært nødvendig å kunne måle lydnivået på enkeltstående lydfiler i forbindelse med manipulering av lyd. F.eks. har det vært nødvendig å måle lydnivået på to filer før de slås sammen, som beskrevet i 4.3.2.1, for å sikre at ingen av filene overdøyver den andre. Til dette har vi brukt Python-biblioteket PyLoudNorm, som tilbyr akkurat denne funksjonaliteten til utviklere [91].

3.9.9 SciPy

I prosjektet håndterer vi store mengder data i forbindelse med maskinlæringsmodellene. SciPy er et økosystem for diverse matematiske og vitenskapelige metoder i Python, som gjør det enkelt å håndtere slik vitenskapelig data på flere ulike måter [52]. Det er *open-source* og ansees for å være standard-biblioteket innenfor slike anvendelser. Innenfor SciPy er det en rekke underpakker som har ansvar for forskjellige typer datahåndtering:

3.9.9.1 Numpy

Under prosessering av data trenger vi datatyper som gjør det lettere å holde orden på, og utføre matematiske operasjoner. Det bruker vi NumPy til. NumPy er et bibliotek som har fokus på utregning med matriser og lister [56]. Spesielt viktig er metoder for å holde orden på hvilke datatyper matrisen holder på og for å sjekke formen (størrelsen) på matrisen. Siden SciPy og dets underpakker er utbredt, gjør NumPy det enkelt å jobbe med store mengder data mellom flere biblioteker. TensorFlow og Keras for eksempel støtte har for å bruke NumPy som input.

3.9.9.2 Matplotlib

Matplotlib er et utbredt rammeverk til Python for å visualisere data [57]. Det er en pakke som er kompatibel med NumPy-objekter. Det har en rekke underbibliotek for forskjellige anvendelser. Et av disse er kalt PyPlot, som assimilerer graftegning i MATLAB innenfor Python. Å få visualisert data gjør at man kan analysere og sammenligne forskjellige data. Dette er kommet godt til nytte i prosjektet vårt, f.eks. fordi vi har behov for å tegne matriser som sier noe om ytelsen til ulike maskinlæringsmodeller, slik at vi senere kan sammenligne de ulike modellene.

3.9.9.3 Wavfile

Ved de tilfellene der vi manipulerer lyddata, i form av Numpy-matriser eller lignende, som skal brukes for å trene maskinlæringsmodellene, må vi lagre dataen som .wav filer før de brukes for trening. Til dette bruker vi Wavfile, som er en pakke som hører til SciPy [29]. Wavfile gjør det mulig å laste inn og skrive til disk, filer av typen ".wav".

3.9.10 Pandas

Pandas er i likhet med Matplotlib (3.9.9.2) et Python-bibliotek som er laget for å analysere data, bl.a. i form av Numpy-objekter [92]. Dette biblioteket har også vist seg nyttig i løpet av prosjektet i forbindelse med analyse av prediksjons-data fra modellene for maskinlæring som brukes til talegjenkjenningssystemet.

3.9.11 os

Ved håndtering av lyddata bruker vi mappestrukturen innad i prosjektet. os er en modul innebygget i Python som gjør det mulig å kommunisere med operativsystemet, og derfor gjøre dette [30]. Fordelen med denne modulen er at man lett kan gå gjennom filer på datamaskinen, slik at man kan anvende dataen i systemet.

3.9.12 random

Når vi trener en modell er det viktig å få variasjon i datasettet. Derfor stokker vi om på datasettet før vi mater inn data til modellene våre. For å gjøre dette genererer vi tilfeldige tall gjennom den innebygde modulen random [31]. Modulen åpner for å generere pseudo-tilfeldige tall, slik at man får tilnærmet tilfeldig data. Det er også mulig å generere tilfeldige tall basert på diverse distribusjoner. Random-modulen har også blitt brukt for å sikre tilstrekkelig tilfeldighet i den faktiske dataen som brukes for å lære opp modellene - f.eks. for å gi manipulerte lydfiler tilfeldighet i start og sluttpunkt, slik at slike uviktige karakteristikker ikke blir interpolert av maskinlæringsmodellene.

3.9.13 secrets

Det er også viktig at vi i applikasjonen kan generere tilfeldige "tokens" - unike, ikke-gjettbare og krypterte strengverdier som inkluderes i HTTP-forespørslene til brukere av administratornettsiden. For å få til dette har vi brukt Python sitt egne, anbefalte bibliotek for å generere slike tokens, *secrets* [68]. Vi kan i applikasjonen kan bruke disse som engangsnøkler til beskyttede sider og ressurser, og er nyttige for oss da vi f.eks. ønsker å sikre at kun inviterte brukere skal ha tilgang til siden for registrering.

3.9.14 pickle

Under utredning av modellene bruker vi forskjellige data for å gi forskjellige resultat. Før vi trener en modell, blir dataen omformet til ulike verdier i et objekt av klassen *AudioPreparer*. For å kunne lagre dette objektet på datamaskinen og slippe å omforme dataen hver gang vi skal trene en modell, gjør vi objektet om til en byte-streng gjennom modulen pickle i Python. Pickle lagrer altså tilstanden til objekter. På denne måten kan man hente opp objektet senere, slik at man slipper å generere informasjonen på nytt [32].

3.9.15 Flask

Da vi i systemet har valgt å bruke Python som vårt hovedspråk, har vi sett på rammeverk som tilbyr webtjening innenfor språket. Flask er et rammeverk for Python for å utvikle nettsider og omkringliggende logikk. Rammeverket lar deg enkelt sammenkoble backend-logikk i Python, med front-end logikk i HTML og Javascript. I tillegg har også Flask innebygd teknologi for "routing" på nettsider, autorisering, autorisasjon og session management, samt brukerhåndtering [5]. Det finnes også mange andre utbredte rammeverk som tilbyr lignende funksjonalitet i Python, f.eks. Django. Flask er dog ikke fullt så komplekst som Django, og dermed enklere å forholde seg til [6]. Da vi allerede var kjent med Flask, og oppdragsgiver ikke hadde noen spesiell preferanse for valg av rammeverk, valgte vi derfor Flask, ettersom den administrative websiden i seg selv ikke er et stort og komplekst websystem.

3.9.15.1 FlaskLogin

Da vi for den administrative nettsiden har måttet håndtere bl.a. innlogging og utlogging av brukere, samt autorisering i forbindelse med nettsider som kun skal være tilgjengelige for visse personer, har Flask Login kommet godt med. Flask Login er en undermodul til Flask (3.9.15) som utvider funksjonaliteten til Flask ved å tilby mekanismer for bruker- og sesjonshåndtering, samt autentisering og autorisering. En webutvikler kan enkelt ta i bruk modulen for å bl.a. sikre at kun brukere som er innlogget får tilgang til spesifikke sider, holde oversikt over hvilke brukere som er logget inn, logge ut brukere eller beskytte brukeres HTTP-cookies [83].

3.9.15.2 Flask Flashing

For den administrative nettsiden, som for de fleste nettsider, har tilbakemelding til brukeren vært et viktig aspekt. For å gjøre varsling av tilbakemelding til brukeren enkelt og intuitivt for oss som utviklere, har vi brukt Flask Flashing. Dette er en undermodul til Flask, som gjør at tilbakemeldinger som sendes ut vedvarer over flere HTTP-forespørseler. På denne måten kan vi som utviklere sende ut en tilbakemelding til brukeren som svar til en HTTP-forespørsel, og vise tilbakemeldingen på f.eks. neste HTTP-forespørsel, eller når det skulle passe seg. Dette gjøres ved at tilbakemeldingene knyttes til cookies som tilhører brukerenes sesjoner [84]. I tillegg gjør Flask Flashing at vi kan gi tilbakemeldinger til brukeren på en konsekvent og uniform måte.

3.9.16 Jinja2

For å få kommunisert data fra backend i Flask til frontend i HTML, har vi brukt Jinja2. Jinja2 er en templet-motor for Python. Jinja2 blir brukt av Flask for å kompilere HTML-filer, slik at data fra Python kan innlemmes i HTML under kjøretid. [8]

3.9.17 Psycopg2

Den administrative nettsiden må også ta høyde for lagring av data, f.eks. innloggingsinformasjon eller loggføringer for talegjenkjenningstjenesten. Vi har som nevnt (3.8.3) brukt en Postgres-database for å håndtere denne lagringen. Vi trengte dog også en måte for å kommunisere med denne databasen gjennom Python, f.eks. for å hente ut data som skal innlemmes i nettsidene, eller for å tillate dynamisk lagring av data (f.eks. ved registrering av nye brukere) gjennom nettsiden. For å gjøre kommunikasjon mellom Python og Postgres enkelt, har vi tatt i bruk Python-biblioteket *Psycopg2*, som er et database-adapter for Python til Postgres. Det vil si at biblioteket enkelt åpner en kommunikasjonskanal, slik at man enkelt kan gjøre transaksjoner mot databasen fra Python. Det finnes alternativer til psycopg2, som f.eks. txpostgres eller asyncpg [82], men psycopg2 er det mest utbredte alternativet [12] og begge gruppemedlemmer har i tillegg god erfaring med biblioteket fra tidligere. Dermed ble psycopg2 et naturlig valg.

3.9.18 re

I forbindelse med den administrative nettsiden må applikasjonen også håndtere en del innsendt data, f.eks. passord ved registrering. For å sjekke at de innsendte passordene og annen innsendt data møter kravene som stilles, og ikke er endret med en proxy eller under overføring (f.eks. skal et passord være mellom 8-20 bokstaver, inneholde små og store bokstaver, samt et tall), brukes biblioteket "re" i Python. Dette gjør det enkelt å sammenligne en hvilken som helst streng med et regulært utrykk, og dermed å sikre at all data som håndters på serversiden er gyldig [33].

3.9.19 bcrypt

I tillegg til at den administrative nettsiden må ta i mot innsendte passord og validere disse, er applikasjonen også nødt til å håndtere lagring av passord etter endt registrering. Naturligvis må disse passordene krypteres, slik at de ikke er lesbare selv om en angriper skulle få tilgang til databasen som lagrer de. Vi har brukt biblioteket *flask-bcrypt* for å håndtere kryptering av disse passordene for vår nettside. Flask-bcrypt gjør det enkelt å kryptere alle passord med et salt gjennom bcrypt-algoritmen [**?**] (en ikke-reversibel hashing som er spesielt motstandsdyktig mot brute-forcing (2.11.5)) før de lagres i databasen, samt sammenligne passord i klartekst med krypterte passord ved innlogging [**?**].

3.9.20 RecorderJS

Et viktig apsekt med den administrative nettsiden er også at den skal gjøre det enkelt å registrere nye brukere. Brukere som registrerer seg til systemet for talegjenkjenning må ta opptak av seg selv og sende de inn over nett. For å gjøre det mulig å ta opptak lokalt på egen enhet har vi bruk JavaScript-biblioteket RecorderJS [17]. Biblioteket innlemmer funksjonalitet som gir JavaScript tilgang til enhetens lokale opptaksenhet, på tvers av hardware, operativsystem og nettleser.

3.9.21 smtplib

Nettsiden for administratorer må ikke bare håndtere data inn, men også data ut. Når en administrator registrerer en ny bruker, sendes det ut en epost med invitasjonslink til den nye brukeren. For å få til utsending av epost ved hjelp av Python har vi tatt i bruk biblioteket *smptlib*. Biblioteket lager en SMPT-klient som kan brukes til å sende epost til en hvilken som helst SMPT- eller ESMTP-lytter [11].

3.9.22 ConfigParser

Gjennomgående for den administrative nettsiden, er det også behov for å vedvare data som ikke kan hentes fra en database, men som det også er behov for å lagre på en sikker måte, slik at angripere ikke skulle få tilgang til denne dataen dersom det f.eks. skulle lekke kildekode. Et eksempel på dette er konfigurasjonsdata som brukes for å opprette forbindelsen til databasen. For å løse dette har vi tatt bruk av Python-biblioteket *ConfigParser* [67]. Biblioteket gjør det enkelt å skrive til og hente data fra konfigurasjonsfiler som ligger plassert utenfor den opprinnelige kildekoden. Dette tillater oss å bruke en slik konfigurasjonsfil som et slags alternativ til sikker lagring - innhold kan kun leses fra filen dersom man har superbruker-tilgang til operativsystemet.

3.9.23 RPi.GPIO

I dette prosjektet bruker vi en Raspberry Pi (3.1) som den fysiske enheten. På denne er det en rekke pins som kalles GPIO, der man kan sende ut signaler ettersom de skal være på eller av. For å sende ut disse signalen gjennom Python har Raspberry Pi en pakke kalt RPi.GPIO som har funksjonalitet til dette.

3.9.24 Apache HTTP Server 2

Apache 2 er en open-source HTTP-server, som gjør det mulig å tjene websider og andre tjenester over HTTP fra en servermaskin [18]. Vi har hatt bruk for Apache i flere tilfeller gjennom vårt prosjekt, f.eks. i forbindelse med distribuering av den administrative nettsiden på nett, slik at den er mulig å nå fra de ansatte i Avento gjennom en webtjener. Det finnes utallige alternativer til Apache, som f.eks. nginx og Microsoft ISS [19], men vi har i samarbeid med oppdragsgiver funnet ut at Apache er den mest gunstig webtjeneren for vårt prosjekt.

3.9.25 Mod_wsgi

Mod_wsgi er en servermodul som muliggjør tjening av Python-applikasjoner gjennom Apache [**?**]. Enkelt beskrevet bruker modulen Web Server Gateway Interface (WSGI) [21] til å videresende innkommende HTTP-forespørsler fra Apache til webapplikasjoner i Python. Da vi har laget webapplikasjoner i Python, har denne modulen tillatt oss å "koble" disse på nett gjennom Apache.

3.9.26 Certbot

Certbot er et open-source verktøy for webservere som automatiserer prosessen for å installere TLS-sertifikater og aktivere HTTPS-kommunikasjon mot websider på serveren [22]. Det har vært viktig for oss å få på plass HTTPS, ikke minst fra et sikkerhetsperspektiv, men også fordi brukere skal kunne ha mulighet til å gjøre opptak direkte i nettleser på noen av nettsidene vi har implementert - i bl.a. Chrome og Firefox er ikke dette mulig over ren HTTP [24].

3.9.27 Cron

Cron er en type daemon (bakgrunnsprosess) som kan kjøres på en rekke operativsystemer for å kjøre tidsbaserte oppgaver [23]. På Ubuntu kan man bruke kommandolinjeverktøyet *crontab* for å legge til slike jobber - f.eks. annenhver dag, hver søndag kl 10, eller omtrent hver eneste kombinasjon av måneder, dager, timer, minutter og sekunder man kan tenke seg. Cron har vært et viktig hjelpemiddel for oss, da vi bl.a. har hatt behov for å legge inn tidsbaserte oppgaver på serveren for administratornettsiden (f.eks. for å automatisk slette utgåtte brukere etter endt døgn).

3.10 Utviklingsverktøy

Ved utvikling har vi brukt ulike verktøy. Vi forklarer hvilke verktøy og hvorfor vi har brukt disse verktøyene i denne delen.

3.10.1 JetBrains PyCharm

Under utvikling av prosjektet har vi jobbet i det integrerte utviklingsmiljøet PyCharm [64]. Det er utgitt av firmaet JetBrains. Prosjekter i PyCharm inneholder en mappe kalt *.idea* som er med på å spesifisere hvilke innstillinger som er med i prosjektet, som er fordelaktig dersom andre skal åpne prosjektet eller prosjektet skal overfører til andre enheter. Det finnes en rekke utvidelser til programmet for å gjøre det mer komfortabelt å programmere, som for eksempel autofullføring av kode. Man kan kjøre kode innad i programmet, som også har støtte for debugging og vitenskapelig visning av variabler. PyCharm er også kompatibelt med Git, slik at man enkelt kan se hva status på filer er og hvilke endringer som er gjort i fargekoder, som gjør det enklere å jobbe flere på et prosjekt. En god funksjon i programmet er støtte for liveoppdatering når man utvikler nettapplikasjoner for å slippe å starte opp applikasjonen på nytt hver gang man har gjort endringer. Begge gruppemedlemmer hadde fra før av god erfaring både for utvikling i Python og for webutvikling, og fortsatt bruk av dette utviklingsmiljøet var dermed naturlig for dette prosjektet.

3.10.2 FileZilla

For å lettere ordne mappestrukturer og overføre filer til serverene vi har brukt i prosjektet, har vi tatt i bruk FileZilla [65]. Denne applikasjonen kobler til en server på en port ved bruk av de ønskede protokoller. Vanligvis er FTP eller SFTP (2.9.5) brukt for å gjøre dette.

3.10.3 PuTTY

PuTTY er en klient for å koble til en server ved SSH 2.9.6. Fordelen ved å bruke et program som PuTTY til å gjøre dette er at man kan lagre alle eventuelle individuelle innstillinger som trengs ved bruk av SSH opp mot en server. Vi har brukt SSH opp mot serverene vi har brukt i prosjektet for å installere nødvendige pakker, databaser, programmeringsspråk og lignende som kreves for å kjøre serverene med de applikasjonene vi har utviklet (eks. den administrative websiden).

3.11 Utviklingsmiljø

Under utvikling har vi ulike utviklingsmiljø som er ansvarlige for de underliggende systemene. Under er det beskrevet hva vi har utviklet i de forskjellige miljøene og dets struktur.

3.11.1 Hovedmiljø

I hovedmiljøet har det meste av utvikling foregått. Det er her vi har laget administrasjonssiden og utviklet maskinlæringsmodellene våre. I den øverste mappen har vi plassert administrasjonssystemet som opererer som en server i det ferdige systemet. Under denne mappen er det opprettet en egen mappe for talegjenkjenning, siden serveren skal ha ansvar for å kommunisere med denne tjenesten. Under utvikling har vi hatt to separate mapper for talegjenkjenningssystemet. Grunnen for at vi har to mapper for dette er at vi trenger et miljø for å utrede og teste modellene, og et for å holde på det endelige produktet som faktisk skal være med i det endelige systemet.

3.11.2 Miljø for Innspillingsenhet

Innspillingsenheten vår skal kommunisere med hovedmiljøet i det endelige produktet. Derfor har vi opprettet et eget miljø for denne tjenesten, slik at vi kan teste systemet. Fordelen ved å gjøre det er også at vi da enkelt kan overføre filene til operativsystemet for innspillingsenheten uten å senere slette dataen her.

3.11.3 Miljø for system av innsamling av taleopptak

Når vi skulle lage en måte å hente inn data av folk som sa *Åpne dør*, opprettet vi et separat system. Grunnen til dette var at dette skulle bli brukt som innsamling og ikke var en opprinnelig del av systemet. I tillegg var det lettere å skille hva som skulle bli lastet opp på serveren som skulle tjene siden. Dette er et system bygget opp med Flask, som vi beskriver mer om i detalj i 4.7.3.

3.12 Data

Det er blitt brukt flere ulike metoder for å samle inn data iløpet av dette prosjektet. Vi har både samlet inn data selv med mikrofon og egen opptaksenhet, gjennom en egenutviklet spørreundersøkelse på nett, og også innhentet taledata fra nettet. I prosjektet bør all lyd være i likt format. Ettersom lyd er lagret i ulike lydformater, har vi bestemt oss for å bruke lydfiler i WAV-format, slik at vi kan bruke de samme bibliotekene på alle filer. Grunnen for at WAV ble valgt kontra andre lydformater er at lyden ikke er komprimert for å spare plass, som gjør at lydfilene blir av høyere kvalitet. All lyd er også omgjort til frekvensen 16kHz, som betyr at hvert sekund i lydfilen består av 16 tusen datapunkter. Uavhengig av det opprinnelige formatet til filene, blir de konvertert over til disse spesifikasjonene.

3.12.1 Oversikt

Nedenfor er en oversikt over alle datasett vi har brukt med en kort beskrivelse. Det blir skrevet mer om hvert datasett i de påfølgende seksjonene.

1. **Opptak av medstudenter** - Data tatt opp av medstudenter, der hver student har en egen frase på lengde med en avisoverskrift.
2. **Opptak via applikasjon på web** - Data tatt opp ved et spørreskjema på nett, av personer som sier frasen "Åpne dør".
3. **Common Voice** - Datasett administrert av Mozilla med lydopptak på engelsk fra folk over hele verden.
4. **MS-SNSD** - Datasett fra Microsoft med støy fra hverdagslige situasjoner (Trafikk, AirCondition, osv.).
5. **ImageNet** - Datasett som inneholder bilder som er koblet opp mot et synset (en type ordgruppe som har lik betydning).

3.12.2 Opptak av medstudenter

Den første og enkleste metoden for datainnsamling vi har brukt, er egen innsamling av stemmedata ved å fysisk gå rundt fra person til person med mikrofon og opptaksenhet. I all hovedsak ble disse lydfilene samlet inn fra medstudenter på forskjellige steder på campus. Vi har brukt en MacBook som opptaksenhet og tatt opp tale gjennom QuickTime[89]. Opptak i QuickTime blir opprinnelig lagret i ".m4a"-format, slik at vi dermed ble nødt til å konvertere disse til ".wav"-format før de ble brukt til opplæring og validering av maskinlæringsmodellene. Denne konverteringen ble enkelt gjort ved bruk av "AudioSegment" fra pydub-biblioteket ([39]) for Python.

Format	Samplingsrate	Antall Personer	Antall Klipp
.m4a	44kHz	33	102

Table 2: Datasett for opptak av medstudenter

3.12.3 Opptak via applikasjon på web

Senere i prosjektet har vi gått over til å samle inn opptak av folk som sier frasen *Åpne dør*. En måte å samle inn denne dataen var å ta opptak slik vi gjorde av medstudenter, men for å øke antall lydprøver gjorde vi dette gjennom en egenutviklet nettapplikasjon utformet som et spørreskjema. Under utfylling av skjemaet måtte brukeren registrere kjønn og alder, samt ta tre opptak der han/hun sier frasen "åpne dør". Opptak på forskjellige opptaksenheter vil gi forskjellig lyd. Fordelen ved at applikasjonen var på nett er at man også får opptak av en delmengde mikrofoner, som muligens kan bidra til at modellene blir bedre, siden modellene vil ta høyde for dette.

3.12.4 Common Voice

I tillegg til stemmedata vi har samlet inn selv, har vi også brukt Mozilla sitt *Common Voice*-datasett[40] for maskinlæring. I utgangspunktet er dette

Format	Samplingsrate	Antall Personer	Antall Klipp
.wav	22kHz	70	210

Table 3: Datasett fra opptak via spørreskjema på nett

datasettet ment å bruke for å lære maskinlæringsmodeller hvordan folk snakker. For hver lydfil i datasettet er det informasjon om kjønn, alder og aksent, som kan brukes til andre applikasjoner. Vi har dog brukt dette datasettet for å utvide settet med ikke-verifiserte stemmer. Dette for å gjøre maskinlæringsmodellene mer tilbøyelig til å klassifisere stemmer den ikke er sikre på, som ikke-verifiserte. Det finnes ikke et datasett av stemmer på norsk, så vi har brukt det engelske datasettet. For å kartlegge fingeravtrykket i stemmen kan det fortsatt hende at dette hjelper. Dette datasettet er stort, så totalt sett har vi tatt for oss 1000 tilfeldige opptak fra datasettet, slik at modellene våre fortsatt har fokus på dataen vi skal validere opp mot.

Format	Samplingsrate	Antall Personer	Antall Klipp
.mp3	44kHz	1000	1000

Table 4: Oversikt over data brukt fra Common Voice

3.12.5 MS-SNSD

Microsoft Scalable Noisy Speech Dataset (MS-SNSD) er et offentlig tilgjengelig datasett fra Microsoft som inneholder store mengder med miljøstøy i .wav format, med en samplingsrate på 16 kHz. Datasettet er ment å brukes for å trene opp dype nevrale nettverk til å undertrykke bakgrunnsstøy [36]. MS-SNSD har kommet til god nytte for oss i forbindelse med manipulering av datsett, da vi har brukt støydata til å utvide representativiteten til lyddata vi har samlet inn, mot ulik lyddata som våre modeller kan komme til å måtte predikere etter installasjon hos kunde.

Format	Samplingsrate	Antall Klipp
.wav	16kHz	130

Table 5: Oversikt over data brukt fra MS-SNSD

3.12.6 ImageNet

ImageNet er en database av bilder som er basert på ord i WordNet-databasen som skal fungere som et samlested for problemer relatert til bildeklassifisering[60]. WordNet-databasen er en database som samler ord som er relatert til hverandre i grupper. For eksempel vil dette innebære at synonymer er samlet som de kaller for synsets. Det er over 100 000 synsets i databasen. ImageNet prøver å ha 1000 bilder som beskriver hvert av disse synsettene.

Denne databasen har tidligere vært brukt til å trene en arkitektur kalt ResNet50, som er et nevralt nettverk med 50 forskjellige lag. Vi kan derfor bruke vektene til hvert lag i den ferdigtrente modellen for å initialisere ett nettverk. Denne arkitekturen blir skrevet om mer i detalj under ResNet50 i seksjon 4.3.1.2.

Antall Bilder	Antall Synsets
14.197.122	21841

Table 6: Oversikt over ImageNet-databasen

3.13 Utstyr

Her beskriver vi hvilke fysisk utstyr vi har trengt i prosjektet. Dette er utstyr som er komponenter i opptaksenheten i systemet.

3.13.1 Raspberry Pi 2B

Under prosjektet har vi brukt en Raspberry Pi 2B som opptaksenhet. Det er en liten maskin som inneholder mange av de samme komponentene som en vanlig datamaskin. Enheten har ulike porter man kan gjennom signaler sende strøm gjennom, til bruk for for eksempel å skru på dioder eller låse opp en dør. På denne er det installert 32-bit-versjonen av Ubuntu Server 18.04.4 LTS som operativsystem.

Grunnlaget for valg av enhet er at samme typen enhet var koblet opp mot låsesystemet i det forrige bachelorprosjektet hos Avento. Derfor hadde oppdragsgiver allerede en Raspberry Pi 2B. Fordelen ved dette er at vi enkelt kan overføre systemet vårt ved å bytte ut lagringskortet til deres enhet med det vi bruker for å teste. Grunnet utbruddet av Covid-19 har vi ikke hatt tilgang på oppdragsgiver sin enhet. Likevel hadde et av gruppemedlemmene er lik enhet hjemme, som vi har brukt under testing av løsningen.

Figure 3.1: Raspberry Pi 2B[61]

3.13.2 SanDisk microSDHC Ultra 32GB

For lagring på Raspberry Pi har vi kjøpt inn et microSD-kort fra SanDisk. Med tanke på størrelse fant vi ut at 32GB ville være nok for vårt tilfelle. Det åpner for at vi har mer enn nok lagringsplass etter å installere operativsystemet. Minnekortet har en hastighet på opptil 98MB/s, som er nok til å ha en god brukeropplevelse under utvikling.

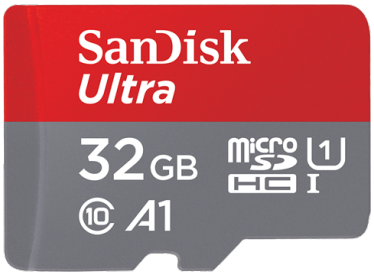


Figure 3.2: SandDisk microSDHC Ultra 32GB[62]

3.13.3 Plexgear M-4 Mikrofon

Når vi har tar opp lyd fra opptaksenheten bruker vi mikrofonen Plexgear M-4. Med denne kan man ta opp lyd mellom 20Hz og 20kHz, som gjør at den er kompatibel med samplingsraten på 16kHz vi har brukt som standard i prosjektet. Dette gjør den gjennom en AUX-kabel. På mikrofonen er det også en fysisk bryter som kan brukes til å blokkere at lyd blir tatt opp. Dette gir et ekstra sikkerhetsledd framfor oppdragsgiver dersom de ikke ønsker at tale skal sendes inn i systemet.

Figure 3.3: Plexgear M4 [117]

3.13.4 USB sound card USC-100

På grunn av at vi i dette prosjekt bruker en Raspberry Pi 2B som opptaksenhet, og denne ikke har en AUX-inngang til bruk av mikrofon, har vi hatt behov for et lydkort for å kunne ta opp lyd. Lydkortet vi har brukt kalles USB sound card USC-100. Lydkortet kobles til Raspberry Pi, og har støtte både for input og output av lyd gjennom AUX-innganger. Det gjør at den både kan kobles til mikrofon og høytaler.

Figure 3.4: USB sound card USC-10 [118]

4 Resultater

I dette kapittelet forteller vi hvilke resultater vi har oppnådd i prosjektet. Vi forklarer først hvordan systemet i sin helhet foreligger, med tilhørende diagrammer for å vise funksjonalitet, før vi forklarer mer i detalj hvordan vi har utviklet de ulike delene.

4.1 Systemet i sin helhet

Når vi har sammensatt alle delsystemene i det ferdige produktet ser sammenhengen mellom dem slik som i figur 4.1.

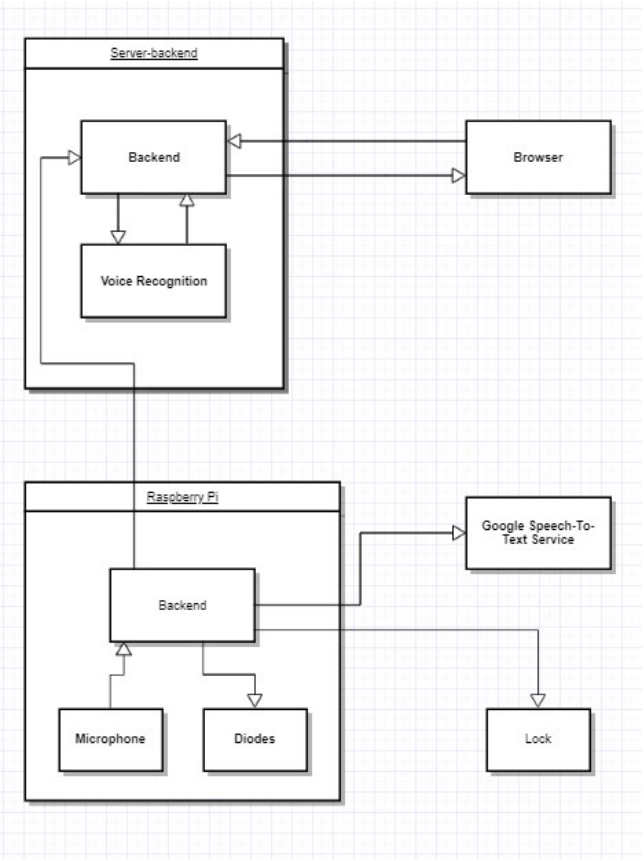


Figure 4.1: Diagram av systemet i sin helhet

Opptaksmiljøet er en Raspberry Pi som har programvare til å ta opp lyd gjennom en mikrofon. Den sender lyd den hører til Google sin tale-til-tekst-tjeneste for detektere hvilken frase som er sagt. Dersom frasen indikerer at noen ønsker å låse seg inn til kontorlokalene, sender den så tale til serveren for autorisasjon av

opptaket. Dersom serveren sender en respons tilbake om at talen var verifisert, vil Raspberry Pi sende et signal til låsesystemet til oppdragsgiver for å låse opp inngangsdøren.

Serveren tar statistikk av alle forsøk og lagrer de i en database. Denne databasen kan administrator få adgang til gjennom en nettapplikasjon som serveren også tjener. I tillegg til informasjon om serveren har denne webapplikasjonen støtte for å registrere nye brukere til systemet, mm.

4.2 Use Case

Figur 4.2 viser et Use Case-diagram som beskriver hva personer med ulike roller kan gjøre i systemet. Ved å bruke et Use Case-diagram, kan vi vise ovenfor oppdragsgiver hva som kan gjøres med systemet fra et bruker- og administratorperspektiv. Brukere er personer som skal bruke selve låsesystemet, mens administratorer skal ha overordnet oversikt og utføre en rekke operasjoner mot systemet. En administrator har også mulighet for å være en bruker.

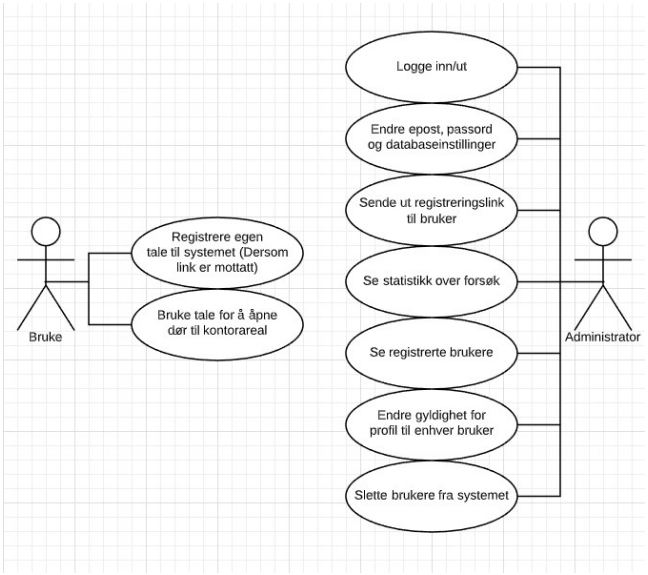


Figure 4.2: Use Case-diagram av systemet

4.3 Prosessen om utredning av modeller for talegjenkjenning

I denne seksjonen tar vi for oss de ulike metodene brukt for å utrede maskinlæringsmodeller. Med begrensede forkunnskaper om emnet tar prosessen for seg hvordan vi har gått frem for å teste ut ulike løsninger for å komme frem til det

endelige produktet. I starten beskriver vi selve testoppsettet, før vi beskriver ulike teknikker brukt for å få ønsket resultat. Deretter går vi gjennom prosessen for utredning, før vi konkluderer med det endelige resultatet.

4.3.1 Testoppsett

Når man arbeider med talegjenkjenning jobber man med mange forskjellige teknikker. Det gjør at vi gjerne har implementert disse lokalt for utprøvelse, før de blir en del av prosjektet. Derfor, for å holde arbeidet systematisk, har vi definert noen klasser som utgjør et skjelett under testingen. I hovedsak er klassene beskrevet innenfor denne seksjonen ansvarlige for den generelle logikken rundt maskinlæringsmodellene, mens en rekke preprosesseringsskritt ofte tas gjennom andre filer i hovedmappen. Disse skrittene beskrives senere under seksjoner for andre hjelpeverktøy (4.3.2). En oversikt over de ulike klassene og hjelpemodulene/verktøyene som er brukt i forbindelse med utredning av maskinlæringsmodeller og hvordan disse henger sammen, finnes i figur 4.3.

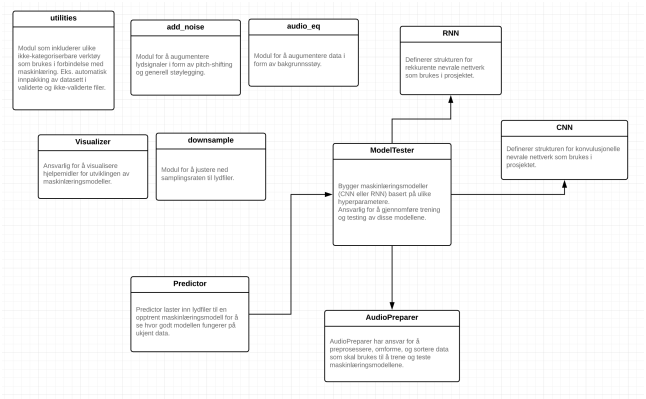


Figure 4.3: UML-Diagram av testoppsettet

4.3.1.1 AudioPreparer

AudioPreparer (4.3) er en klasse som skal holde på dataen som skal kjøres i modellene. Denne klassen er satt opp slik at man kan laste inn data og omforme disse til et datasett med tilsvarende etiketter. Klassen har hjelpefunksjoner for å normalisere og stokke om på datasettet. Hovedgrunnen til at vi har valgt å lagre data her, er på grunn av at det ofte tar lengre tid å omforme alle lydfilene til verdiene som modellen skal trenes opp på. Istedenfor å laste inn all data på nytt hver gang man skal bruke datasettet, kan vi heller lagre objektet på datamaskinen og heller laste inn objektet når datasettet skal brukes.

Innenfor AudioPreparer er det en rekke viktige metoder som er der for å gi

struktur til klassen. For å laste inn data til datasettet har vi en metode *load* (Fig. 4.4) som gjør om en lydfil til data som kan brukes for trening av modellene, og legger til disse i datasettet. Her definerer du også hvilken etikett denne dataen skal ha i datasettet. I tilfellet under gjør vi om lyd til MFCC-verdier (2.5.6), der framgangsmåte blir skrevet mer om i 4.3.3.3.

```
def load(self, path, label):
    """
    Load an audio file, convert it to MFCC, and add it to the dataset
    :param path: Path to the audio file
    :param label: True label of the audio
    """
    if self.logging:
        print(str(label) + ' - ' + path.split('/')[-1])
    y, sr = librosa.load(path, sr=16000, mono=True, duration=self.dur)
    y = librosa.util.fix_length(y, sr*self.dur)
    mfcc = librosa.feature.mfcc(y, n_mfcc=40, hop_length=160, win_length=400, n_fft=512)
    self.dataset.append(mfcc)
    self.labels.append(label)
```

Figure 4.4: Metoden *load*

Etter at dataen er kommet inn i datasettet må vi sikre at dataen struktureres i tilfeldig rekkefølge, slik at maskinlæringsmodellene ikke tror det er avhengighet mellom lydfilene. Derfor har vi implementert metoden *shuffle* (Fig. 4.5), som setter sammen listene for etiketter og data, og endrer rekkefølge på datasettene ved å bruke funksjonen med samme navn, *shuffle*, fra modulen *random* (3.9.12).

```
def shuffle(self):
    """
    Shuffles the dataset
    """
    c = list(zip(self.dataset, self.labels))
    random.shuffle(c)
    self.dataset, self.labels = zip(*c)
```

Figure 4.5: Metoden *shuffle*

Når man skal bruke dataen fra datasettet holder man av en viss delmengde av datasettet til bruk for testing. Dette gjør man for å sjekke om modellene har klart å lære seg datasettet etter trening. Derfor har vi laget en metode *get_test_train_data* (Fig. 4.6) for å dele opp det opprinnelige datasettet i to delsett, ett for testing og ett for trening . Testsettet skal være et tilfeldig utvalg, så det er viktig å bruke *shuffle*-metoden før man henter ut datasettet.

```python
def get_test_train_data(self, ratio):
    """
    Splits the dataset in two based on a ratio, so one part is reserved for testing, and one for training
    :param ratio: The percentage of data in training set.
    :return: Data for training, Labels for training, Data for testing, Labels for testing
    """
    cut = int(len(self.dataset) * ratio)
    return np.array(self.dataset[:cut]), np.array(self.labels[:cut]), np.array(self.dataset[cut:]), np.array(
        self.labels[cut:])
```

Figure 4.6: Metoden *get_test_train_data*

4.3.1.2 Ulike modeller

I prosjektet tester vi ut ulike former for nevrale nettverk (2.2). For å holde orden på disse er arkitekturen til hver type lagt inn i hver sin fil. Hver fil har en funskjon *build_xxx_model*, der *xxx* står for type modell. Under viser vi hvilke modeller og parametere vi bruker under testing av de forskjellige typene.

CNN
For CNN (2.2.1) har vi kommet frem til å bruke følgende nettverk i figur 4.7 for testing av denne type nevralt nettverk.

```python
def build_cnn_model(input_shape):
    """
    Builds a CNN model, compiles it and returns the model.
    :param input_shape: The shape of the dataset.
    :return: The CNN Model
    """
    model = Sequential([
        Conv2D(128, (3, 3), activation='relu', input_shape=input_shape, strides=(1,1)),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(64, (3, 3), activation='relu', strides=(1,1)),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(32, 3),
        Flatten(),
        Dropout(0.2),
        Dense(256, activation='relu'),
        Dropout(0.2),
        Dense(units=2, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

Figure 4.7: Implementasjon av CNN

Her konvolerer vi først MFCC-verdiene til en mindre dimensjon til 128. Deretter samler man verdier fra denne dataen for å gjøre settet enda mindre igjen. Til slutt sitter vi igjen med en dimensjon av 32, som skal gi hovedtrekkene ved input. Disse verdiene er så flatet ut og sendt til ett sett av 256 gjemte noder, før modellen bestemmer om verdiene skal klassifiseres som verifisert eller ikke.

De konvolusjonelle lagene er definert ved ulike hyperparametere. I en artikkel fra Towards Data Science [98] beskriver Shashank Ramesh hvordan man velger

hyperparametere innenfor et CNN. Vi velger filtere på 3x3 (representert som '(3,3)' i koden) for å få mer lokal informasjon om verdiene enn hva vi hadde fått dersom vi brukte et større filter. En annen parameter er *strides*, som er antall piksler man flytter seg for enhver konvolusjon. Vi ønsker å sjekke alle verdiene i dataen, da det er en tidsserie, så vi setter denne verdien som 1x1. Pooling-lagene har filtere på 2x2, som var anbefalt ved bruk av små bilder, eller matriser av data.

Dersom man heller vil prøve å klassifisere enkeltstemmer endrer man antall *units* i det siste laget til å representere antall personer. Gjennom nettverket har vi lagt inn et par *Dropout*-lag, som tar bort andelen data spesifisert for resten av treningen. Jason Brownlee [99] forklarer at å innføre reduksjon av settet vil gi større variasjon under kjøretid, og redusere sannsynligheten for å overtrene modellene, da datasettet vil variere mellom hver epoch. Det er verdt å nevne at dataen ikke blir med på klassifiseringen, men fortsatt er med på å trene modellen frem til de fjernes.

RNN

I motsetning til CNN, har RNN (2.2) en enklere struktur. Det er på grunn av at man ikke må redusere dataen ettersom at modellen ser på hvordan datapunktene i verdien ligger i forhold til hverandre. Kodesnutten under viser hvordan vi har bygget opp dette.

```
def build_rnn_model(input_shape):
    """
    Builds a RNN model, compiles it and returns the model.
    :param input_shape: The shape of the dataset.
    :return: The RNN model
    """

    model = Sequential([
        LSTM(256, input_shape=input_shape, return_sequences=True),
        Dropout(0.2),
        LSTM(128),
        Dropout(0.2),
        Dense(units=2, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

Figure 4.8: Implementasjon av RNN

Nettverket kjører gjennom ett sett av 256 LSTM-celler, før output herifra går videre til en annen LSTM-celle med et redusert antall på 128. Denne dataen går så inn i et lag for å klassifisere dataen. Her har vi også brukt et *Droput*-lag

for å variere dataen.

ResNet50
Under bruk av den ferdigtrente dataen fra ImageNet (3.12.6) har et ResNet50-nettverk vært nødvendig.

```
def build_resnet_model():
    """
    Builds a ResNet model, compiles it and returns the model.
    :return: The ResNet model
    """

    model = Sequential()
    model.add(ResNet50(include_top=False, weight='imagenet', pooling='avg'))
    model.add(Dense(units=2, activation='softmax'))

    model.layers[0].trainable = False

    model.compile(optimizer='sgd',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

Figure 4.9: Implementasjon av ResNet50

I figur 4.9 bruker vi ResNet50-arkitekturen og bruker vektene fra den ferdigtrente modellen som vekter. Deretter går resultatene fra dette inn i output-noder, som gir vårt resultat. Under dette fryser vi de ferdigtrente lagene ved å sette *trainable=False*, slik at vi ikke endrer på vektene til modellen. Dersom man derimot vil kunne endre på vektene fjerner man denne linjen.

4.3.1.3 ModelTester

ModelTester (4.3) er modulen der modellene blir bygget opp. Det har én funksjon *build_model* som bygger en modell basert på de ulike parameterne. Først velger man en modelltype som skal ligge til grunn for modellen, som deretter blir bygget. Så velger man en AudioPreparer som inneholder den ønskede dataen modellen skal trene på. Til slutt spesifiserer man antall epochs modellene skal kjøre gjennom under treningen. Etter at disse verdiene er valgt henter funksjonen datasettet ut fra den valgte AudioPreparer, og bygger modellen som er valgt med dette settet. Først vil den omforme datasettet til å passe input-typen til ønsket modell, for å så bygge en modell av den spesifiserte typen. Denne prosessen kan man se i figur 4.10.

```
model_type = model_type.lower()
# Get dataset from an AudioPreparer
x_train, y_train, x_test, y_test = audio_preparer.get_test_train_data(0.7)
model = None
# Shapes the data based on expected input of models, as well as building the models
if model_type == 'cnn':
    x_train = x_train.reshape(-1, x_train.shape[1], x_train.shape[2], 1)
    x_test = x_test.reshape(-1, x_test.shape[1], x_test.shape[2], 1)
    model = build_cnn_model(x_train.shape[1:])
elif model_type == 'rnn':
    model = build_rnn_model(x_train.shape[1:])
elif model_type == 'resnet':
    x_train = np.stack((x_train,)*3, axis=-1)
    x_test = np.stack((x_test,)*3, axis=-1)
    model = build_resnet_model()
else:
    return "Not valid input type"
```

Figure 4.10: Omforming av datasett til ønsket modell

Når modellen er hentet ut kan man så trene og teste dataen. Dette gjør man gjennom TensorFlow-funksjonene *model.fit* og *model.evaluate*, vist i figur 4.11.

```
# Fits the data based on the fetched training data
model.fit(x_train, y_train, epochs=epochs)
# Evaluates the model based on the fetched testing data
model.evaluate(x_test, y_test)
```

Figure 4.11: Trening og testing av modellene

Etter trening og testing av modellen vil en forvirringsmatrise av resultatene vises, slik at man kan vurdere hvor gode modellene er. Modellen vil også lagres på datamaskinen, slik at man kan bruke den for prediksjon på nye lydfiler.

4.3.1.4 Predictor

Siden vi i dette prosjektet skal lage et låsesystem, må vi teste modellene våre på data som er ukjente for modellene. Det vil både si data av mennesker som ikke har lydfiler i datasettet, og nye opptak av mennesker som er i datasettet. Derfor har vi laget en modul, Predictor (4.3), som man kan gjøre dette i. Her spesifiserer vi mappen til ukjente lydfiler og hvilken modell man vil teste på. Hver lydfil i mappen vil bli gjort om til MFCC-verdier, som så bruker funksjonen *model.predict* fra TensorFlow for å se hvilken etikett den tror lyden har. Dette gjøres enkeltvis for hver lydfil, slik at vi enklere kan se spesifikt hvilken etikett hver lydfil får. Til slutt vil en forvirringsmatrise basert på disse verdiene tegnes opp.

4.3.1.5 Andre ressurser

Ut over klassene og modulene beskrevet over, har vi også spesifisert hvor alle ressurser er lagret, da det jobbes med en stor mengde filer. Dette gjøres i undermappen *resources*, og er strukturert slik:

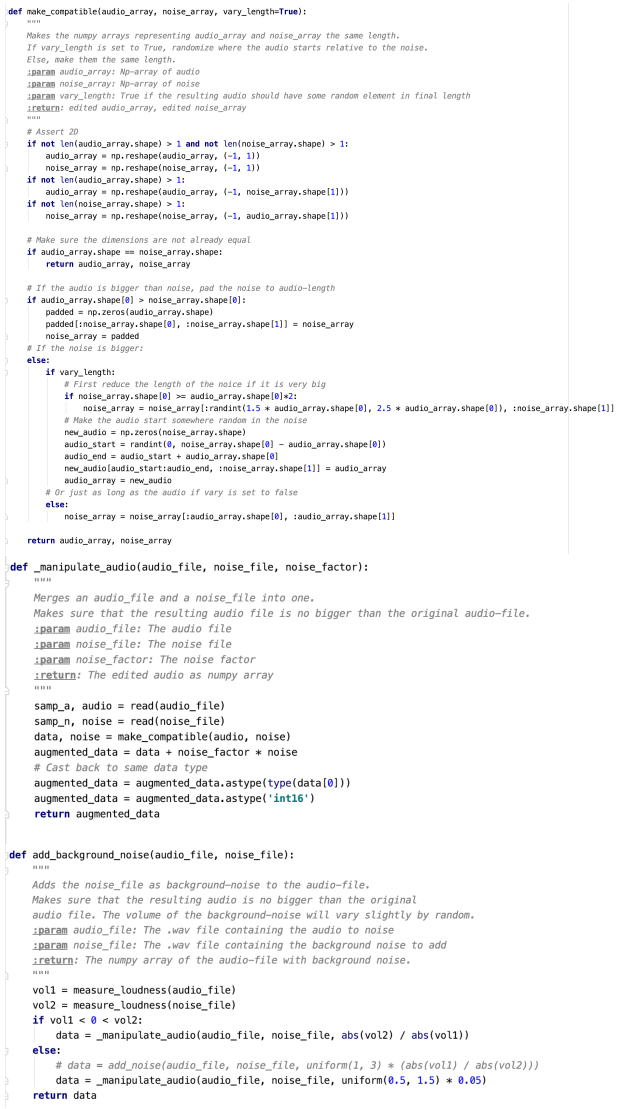
- **AudioFiles** - Denne mappen holder på alle lydfiler i prosjektet. Inne i denne er det flere undermapper ettersom modeller prøves ut.
- **AudioPreparers** - I denne mappen lagres alle objekter av AudioPreparer når de genereres.
- **models** - I mappen *models* lagres alle modeller som blir generert av klassen ModelTester.
- Andre ressurser som eventuelt trengs under utførelse lagres også her.

4.3.2 Andre hjelpeverktøy

I tilegg til hovedskjelettet for talegjenkjenning (4.3.1), som gjør det enkelt å teste, trene og gjenbruke maskinlæringsmodeller for prediksjon, har vi også måtte laget en del hjelpeverktøy underveis i prosjektet. Dette er i hovedsak verktøy som har hjulpet oss å forbedre datasettene våre, ellers forenkle arbeide med talegjenkjenning, eller gi innsikt i hvordan ulike modeller fungerer slik at vi enkelt kan sammenligne forskjellige fremgangsmåter.

4.3.2.1 Generering av bakgrunnsstøy

For at en maskinlæringsmodell skal kunne gjøre konsekvente gode prediksjoner på hvorvidt en person skal verifiseres eller ikke-verifiseres, er det imperativt at modellene trenes opp på datasett som er representative for alle stemmeprøver modellen kan kunne komme til å måtte gjøre prediksjoner for, som drøftet av Borovicka, Tomas, et. al [100]. Et slikt datasett, som tar høyde for alle de individuelle forskjellene mellom stemmene i befolkning, for ikke å nevne ulikhetene i individers stemme over tid, er naturligvis ekstremt vanskelig å samle inn. For å til dels kompensere for de manglene som måtte finnes i datasettene vi selv har samlet inn (3.12), har vi laget en Python-modul som utvider datasettene vi allerede har ved å lage forskjellige redigerte versjoner av eksisterende lyddata. Dette gjøres i hovedsak ved å slå sammen eksisterende lyddata med tilfeldige lydklipp som inneholder støy, slik at det originale lydklippet får tillagt bakgrunnstøy og andre små endringer.



```
def make_compatible(audio_array, noise_array, vary_length=True):
    """
    Makes the numpy arrays representing audio_array and noise_array the same length.
    If vary_length is set to True, randomize where the audio starts relative to the noise.
    Else, make them the same length.
    :param audio_array: Np-array of audio
    :param noise_array: Np-array of noise
    :param vary_length: True if the resulting audio should have some random element in final length
    :return: edited audio_array, edited noise_array
    """
    # Assert 2D
    if not len(audio_array.shape) > 1 and not len(noise_array.shape) > 1:
        audio_array = np.reshape(audio_array, (-1, 1))
        noise_array = np.reshape(noise_array, (-1, 1))
    if not len(audio_array.shape) > 1:
        audio_array = np.reshape(audio_array, (-1, noise_array.shape[1]))
    if not len(noise_array.shape) > 1:
        noise_array = np.reshape(noise_array, (-1, audio_array.shape[1]))

    # Make sure the dimensions are not already equal
    if audio_array.shape == noise_array.shape:
        return audio_array, noise_array

    # If the audio is bigger than noise, pad the noise to audio-length
    if audio_array.shape[0] > noise_array.shape[0]:
        padded = np.zeros(audio_array.shape)
        padded[:noise_array.shape[0], :noise_array.shape[1]] = noise_array
        noise_array = padded
    # If the noise is bigger:
    else:
        if vary_length:
            # First reduce the length of the noice if it is very big
            if noise_array.shape[0] >= audio_array.shape[0]*2:
                noise_array = noise_array[:randint(1.5 * audio_array.shape[0], 2.5 * audio_array.shape[0]), :noise_array.shape[1]]
            # Make the audio start somewhere random in the noise
            new_audio = np.zeros(noise_array.shape)
            audio_start = randint(0, noise_array.shape[0] - audio_array.shape[0])
            audio_end = audio_start + audio_array.shape[0]
            new_audio[audio_start:audio_end, :noise_array.shape[1]] = audio_array
            audio_array = new_audio
        # Or just as long as the audio if vary is set to false
        else:
            noise_array = noise_array[:audio_array.shape[0], :audio_array.shape[1]]

    return audio_array, noise_array
```

```
def _manipulate_audio(audio_file, noise_file, noise_factor):
    """
    Merges an audio_file and a noise_file into one.
    Makes sure that the resulting audio file is no bigger than the original audio-file.
    :param audio_file: The audio file
    :param noise_file: The noise file
    :param noise_factor: The noise factor
    :return: The edited audio as numpy array
    """
    samp_a, audio = read(audio_file)
    samp_n, noise = read(noise_file)
    data, noise = make_compatible(audio, noise)
    augmented_data = data + noise_factor * noise
    # Cast back to same data type
    augmented_data = augmented_data.astype(type(data[0]))
    augmented_data = augmented_data.astype('int16')
    return augmented_data


def add_background_noise(audio_file, noise_file):
    """
    Adds the noise_file as background-noise to the audio-file.
    Makes sure that the resulting audio is no bigger than the original
    audio file. The volume of the background-noise will vary slightly by random.
    :param audio_file: The .wav file containing the audio to noise
    :param noise_file: The .wav file containing the background noise to add
    :return: The numpy array of the audio-file with background noise.
    """
    vol1 = measure_loudness(audio_file)
    vol2 = measure_loudness(noise_file)
    if vol1 < 0 < vol2:
        data = _manipulate_audio(audio_file, noise_file, abs(vol2) / abs(vol1))
    else:
        # data = add_noise(audio_file, noise_file, uniform(1, 3) * (abs(vol1) / abs(vol2)))
        data = _manipulate_audio(audio_file, noise_file, uniform(0.5, 1.5) * 0.05)
    return data
```

Figure 4.12: Funksjoner i modulen audio_eq

Bildene ovenfor (4.12) viser hvordan dette gjøres i praksis, gjennom modulen audio_eq (4.3). Audio-filen som skal endres og støy-filen som skal legges til som bakgrunnsstøy, gjøres begge om til Numpy-matriser (3.9.9.1). Matrisene gjøres så om til samme størrelse (ved å korte ned eller utsette starten for lydklipp eller bakgrunnsstøy) med funksjonen "make_compatible", slik at det skal være mulig å slå sammen de to matrisene. For at det resulterende lydklippet ikke skal ha gjennomgåene mønster (som f.eks. at bakgrunnsstøy alltid starter likt relativt til det originale lydklippet), bruker vi tilfeldighetesmodulen til Python

(3.9.12) for å sikre at det ikke oppstår noen forutsigbare mønstre i den endelige lengden til lydklippet og bakgrunnsstøyet. Når bakgrunnsstøyet legges til over det opprinnelige lydklippet ved hjelp av funksjonen "add_background_noise", måles også lydnivået på de to filene før sammenslås (med "manipulate_audio") gjennom Python-modulen PyLoudNorm (3.9.8). Dette gjøres for å sikre at bakgrunnstøyet ikke ender opp med å overdøyve det originale lydklippet. Nøyaktig hvor fremtredene bakgrunnstøyet ender opp, overlates også til tilfeldigheter med Python sin random-modul (3.9.12), for å sikre at det ikke skal oppstå forutsigbare mønstre i de manipulerte lydfilene. For ordens skyld, vi ønsker ikke mønstre i datasettet som ikke er relevante for hvorvidt en person skal verifiseres eller ikke verifiseres, da det kan føre til at maskinlæringsmodellen interpolerer feil og ender opp med en lavere treffsikkerhet en det den ellers ville gjort. Vi har i denne oppgaven brukt støydata fra MS-SNSD (3.12.5), som vi har lagt til som bakgrunnstøy over innsamlet taledata.

4.3.2.2 Manipulering med tilfeldige verdier

En annen metode for å utvide datasettet har vært å augmentere data ved å endre lyden med teknikker hentet fra en bloggpost fra Edward Ma i Medium [47]. Implementasjonen av disse teknikkene er lagret i filen "add_noise.py" (4.3).

Lydfiler har et spesifikt start og sluttpunkt. Det gjør at en modell uheldigvis kan bruke dette til å bestemme hvilken klasse lyden tilhører - noe som ikke kommer til nytte for vårt bruk. Derfor har vi lyst til å endre start og sluttpunkt til enhver fil med tilfeldighet. For å gjøre dette flytter vi lyden i en tilfeldig retning relativt til den originale posisjonen. På grunn av at lydfilene til slutt skal være på tre sekunder, som vi forklarer i seksjon 4.3.5 nedenfor, har vi spesifisert at lyden maksimalt kan flyttes et halvt sekund tilbake eller frem i tid for å være sikre på at hele lyden ikke erstattes. Etter flyttingen er utført vil det bli lagt på nuller (som representerer stillhet) der det nå er tomrom. Dette er vist i figur 4.13.

```python
def shift_audio(data, sampling_rate=16000, shift_max=0.5, shift_direction='both'):
    """
    Shifts the audio data in a certain direction
    :param data: The data to be shifted
    :param sampling_rate: The sampling rate of the audio
    :param shift_max: Maximum shift (in seconds)
    :param shift_direction: The direction of the shift
    :return: The shifted audio data
    """
    shift = np.random.randint(int(sampling_rate * shift_max))
    if shift_direction == 'right':
        shift = -shift
    elif shift_direction == 'both':
        direction = np.random.randint(0,2)
        if direction == 1:
            shift = -shift

    augmented_data = np.roll(data, shift)
    # Set to silence for heading/ tailing
    if shift > 0:
        augmented_data[:shift] = 0
    else:
        augmented_data[shift:] = 0
    return augmented_data
```

Figure 4.13: Flytting av lyd i tilfeldig retning

For å legge til støy lager vi en liste av tilfeldige tall tilsvarende lengde av lydfilen. Disse tallene blir da lagt til i den opprinnelige lydfilen. For å skape mer variasjon blir også støyfilene redusert ved å gange inn en tilfeldig faktor. Implementasjonen av dette er i figur 4.14.

```python
def add_random_noise(data, noise_factor):
    """
    Adds random noise to audio data
    :param data: The data to be added to
    :param noise_factor: The factor of noise to be added
    :return: The audio data with random noise added
    """
    noise = np.random.randn(len(data))
    augmented_data = data + noise_factor * noise
    # Cast back to same data type
    augmented_data = augmented_data.astype(type(data[0]))
    return augmented_data
```

Figure 4.14: Legge til tilfeldig støy

Noen lydfiler er over tre sekunder, mens andre er under. Derfor har vi før augumentering av hver lyd valgt å gjøre om lydfiler til denne lengden. For filer som er over tre sekunder, blir tilfeldige tre sekunder fra lyden hentet ut. For filer som er under, vil lyden settes inn et tilfeldig sted av en liste av nuller. For å sikre at ingen har samme utgangspunkt, vil denne kuttingen gjøres på nytt mellom hver

støyfil blir generert. Dette gjør vi i funksjonen *convert_audio_to_desired_length* som vist i figur 4.15.

```python
def convert_audio_to_desired_length(y, sr=16000, length=3):
    if len(y) < sr * length:  # If sound is longer, it will grab a random subset of audio
        pad = np.zeros(sr * 3)
        buffer = len(pad) - len(y)
        start = np.random.randint(0, buffer)
        pad[start:start + len(y)] = y
        y = pad
    elif len(y) > sr * length:  # If sound is smaller, it will pad audio randomly
        start = np.random.randint(0, len(y) - sr * 3)
        y = y[start:start + sr * 3]
    return y
```

Figure 4.15: Funksjonen *convert_audio_to_desired_length*

Hvis vi tar for oss et tilfeldig lydklipp fra datasettet av en person som sier *Åpne dør* i figur 4.16.

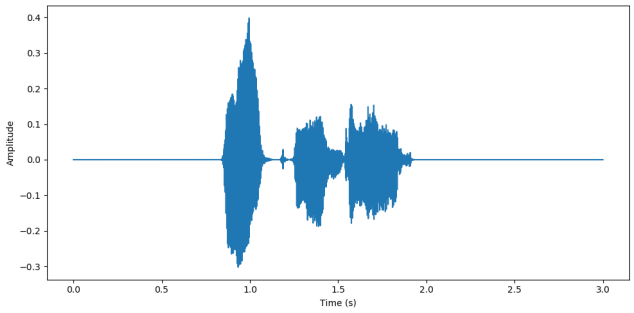


Figure 4.16: Opptak av frasen *Åpne dør*

Da kan vi generere augmenterte filer basert på denne. I figur 4.17 har vi generert fire filer på lydklippet fra figur 4.16.

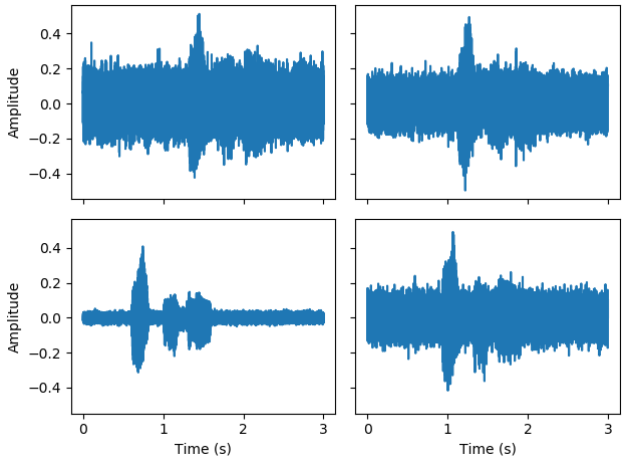


Figure 4.17: Fire eksempler av augumentering av lydklippet i figur 4.16

Det er verdt å merke at denne metoden skiller seg fra metoden for tillegging av bakgrunnsstøy som er beskrevet under seksjon 4.3.2.1 på flere måter. For det første er denne metoden hovedsakelig implementert for å endre start- og sluttpunkt på lydfiler, slik at det ikke skal oppstå forutsigbare mønstre i lydfilene. Metoden beskrevet under seksjon 4.3.2.1 er i hovedsak implementert for å utvide datasettet med representativ støydata, slik at modellen kan håndtere ulike dynamiske faktorer - som f.eks. bakgrunnstøy i lokalene til Avento. En annen viktig forskjell er at der metoden for tilegging av bakgrunnsstøy (4.3.2.1) bruker faktiske lydklipp for å gi en representativt fremstilling av lydmiljøene som kan befinne seg rundt innspillingsenheten etter installasjon hos Avento, bruker denne metoden syntetisk støy, først og fremst for å motvirke ufordelaktig interpolering hos maskinlæringsmodellene.

4.3.2.3 Visuelle hjelpemidler

For å finne frem til den beste løsningen for talegjenkjenning, har det også vært nødvendig å trene opp og teste ut store mengder forskjellige varianter av datsettene og ulike modellstrukturer. For å enkelt kunne sammenligne ytelsen til ulike modeller, forenkle feilsøking, og mer, har vi integrert en del visuelle hjelpemidler i prosjektets testmiljø.

Det er klassen Visualizer (fra fig. 4.3) som er ansvarlig for å konstruere og tegne opp disse visuelle fremstillingene. Klassen implementerer en forholdsvis enkel metode som bruker innebygde metoder i TensorFlow (3.9.1) og Pandas (3.9.10) til å analysere prediksjonene fra en maskinlæringsmodell. Deretter brukes Matplotlib (3.9.9.2) til å tegne en forvirringsmatrise (2.6.1), som gir oss overblikk over hvordan modellen har gjort prediksjoner på lyddata under validering, i forhold til de reelle verdiene modellen burde ha predikert dersom den hadde en

treffsikkerhet på 100 prosent. Koden som brukes for å tegne en slik matrise kan sees under, på figur 4.18:

```python
def draw_confusion_matrix(labels, predictions, num_classes):
    """
    Draws a confusion matrix for the given predictions.
    :param labels: The correct labels
    :param predictions: The predicted labels
    :param num_classes: Number of differenct classes in the labels
    """
    # Create the confusion matrix and normalize data
    cf = tf.math.confusion_matrix(labels=labels, predictions=predictions, num_classes=num_classes).numpy()
    cf_norm = np.around(cf.astype('float') / cf.sum(axis=1)[:, np.newaxis], decimals=2)
    cf_df = pd.DataFrame(cf_norm)

    # Draw the confusion matrix
    plt.figure(figsize=(8, 8))
    sns.heatmap(cf_df, annot=True)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

    for ax in axs.flat:
        ax.label_outer()

    plt.tight_layout()
    plt.show()
```

Figure 4.18: Funksjon for å lage forvirringsmatrise

I tillegg til forvirringsmatriser, har et annet viktig hjelpemiddel under testing vært visualisering av innholdet i en lydfil, fordi man gjerne vil sjekke hvordan de ulike matematiske prosessene påvirker filen. Et eksempel på hvordan en slik visualisering ser ut, finnes på figur 4.17, der amplituden i ulike lydfiler er plottet over tid. I tilegg til funksjon for å tegne amplitude over tid, implementerer også klassen metoder for å visualisere MFCC-verdier (2.5.6), Melspektogram (2.5.4.1) og STFT-spektrogram (2.5.2.1).

4.3.2.4 Annet

I forbindelse med trening og testing av maskinlæringsmodeller, har det også vært nødvendig å innføre en del hjelpemidler for å automatisere stegene som kreves for å gjøre klar et datasett til trening og testing. Metoder som er brukt som hjelpemidler for maskinlæringen, men som i og for seg er enkeltstående oppgaver og ikke passer i kategorier som visuelle hjelpemidler (4.3.2.3) eller preprosessering, er samlet i Python-modulen "utilities.py" (4.3). Her finnes bl.a. en metode for å automatisk dele inn et sett med talefiler i mapper for to etikketer, verifisert og ikke-verifisert, som vist på figur 4.19. Her finnes også metoder som f.eks. henter ut informasjon om hvor alvorlig modellen har bommet i de tilfellene der den har predikert falske-positiver og falske-negativer, mm.

```python
def automatic_labeling(folder, auth_rate=0.5):
    """
    Handles automatic labeling of files in a folder.
    It assumes the files follow the naming scheme 'Person-Id-Place.wav', i.e. 'Fredrik-1-Lab.wav'.
    The folder should only contain these files, no internal ordering.
    This method will divide the folder into two subfolders '0/' and '1/'.
    Files with the same 'Person' part will be assumed to belong together and will thus end up in the same subfolder.
    A share of 'auth_rate' of the persons will end up in '/1' subfolder, the rest in '0/'.
    :param folder: The folder to label
    :param auth_rate: The share of people to put in '1/' subfolder
    """
    if not os.path.exists(folder + '/0'):
        os.makedirs(folder + '/0')
    if not os.path.exists(folder + '/1'):
        os.makedirs(folder + '/1')
    handles = list(set([filename.split('-')[0] for filename in os.listdir(folder) if filename.endswith('.wav')]))
    shuffle(handles)
    auth_handles = handles[:int(len(handles) * auth_rate)]
    noauth_handles = handles[int(len(handles) * auth_rate):]

    for f in os.listdir(folder):
        if f.endswith('.wav'):
            if f.split('-')[0] in auth_handles:
                os.rename(folder + '/' + f, folder + '/1/' + f)
            elif f.split('-')[0] in noauth_handles:
                os.rename(folder + '/' + f, folder + '/0/' + f)
            else:
                raise LookupError('Something wrong with file {} labeling in folder: {}'.format(f, folder))
```

Figure 4.19: Metoden "automatic_labeling" for å splitte datasett i verifiserte og ikke-verifiserte filer

4.3.3 Annen Lydprosessering

Sammen med metodene for tillegging av støy og endringer av start- og sluttpunkt for lyden (4.3.2), har det også vært nødvendig å prosessere lyddata på andre måter for å sikre at maskinlæringsmodellene kan interpolere godt fra datasettene de mates med. En god maskinlæringsmodell kan bl.a. ikke legge for stor vekt på ekstremverdier i datasettene som blir brukt for å trene den, da dette f.eks. kan føre til en skjevhet i hvordan modellen utfører prediksjoner. Viktigheten av dette for maskinlæringsmodeller som trener på taledata, er diskutert videre i Bhanja, Samit, et.al [88], som også har dokumentert normaliseringsmetoden vi har brukt for å normalisere MFCC-verdier (4.3.3.2).

4.3.3.1 Normalisering av lydnivå

For å normalisere lydnivået til -20dBFS slik beskrevet i 2.4, har vi implementert en metode for dette i filen *normalize.py*. Denne filen bruker metoder fra klassen AudioSegment fra pakken pydub (**??**). For hver lydfil som skal normaliseres, gjøres filen om til et AudioSegment gjennom metoden *from_wav*. Deretter kan vi gjøre ønskede operasjoner på filen. For å normalisere lydnivået bruker vi funksjonen i figur 4.20.

```python
def normalize(audio, new_dBFS):
    """
    Normalizes audio based on given dBFS
    :param audio: An object of AudioSegment to change dBFS from
    :param new_dBFS: The desired dBFS
    :return: An AudioSegment with desired dBFS
    """
    diff = new_dBFS - audio.dBFS
    return audio.apply_gain(diff)
```

Figure 4.20: Implementasjon av normalisering av lydnivå

Her er *audio* et objekt av AudioSegment, og *new_dBFS* den ønskede verdien for dBFS. Etter at vi har sendt en lydfil til denne funksjonen, kan vi så eksportere resultatet til en annen mappe gjennom metoden *export* fra AudioSegment. Resultatet blir at filene er normalisert.

4.3.3.2 Normalisering av MFCC-verdier

Vi har også implementert en metode for å normalisere MFCC-verdiene (2.5.6) som brukes for å trene maskinlæringsmodeller - slik at dersom det finnes lydfiler med ekstreme MFCC-verdier i forhold til resten av datasettet, skal ikke dette gi enorme utslag på vektene til maskinlæringsmodellene. Teknikken for å normalisere MFCC-verdier går ut på at hver koeffisient får en verdi mellom 0 og 1 relativt til maksimum og minimumsverdien til datasettet. Først blir det dannet to matriser som holder på disse verdiene, før hvert element i MFCC-en blir endret i henhold til denne formelen (5).

$$k' = \frac{k + min}{min - max} \tag{5}$$

Her er *k'* den normaliserte verdien av den opprinnelige MFCC-koeffisienten *k*, og min og max er minimums- og maksimums-verdien for hver linje i MFCC-matrisen, respektivt. Dersom man har disse verdiene kan man normalisere verdiene ved å implementere formel 5 inn i koden, slik som i figur 4.21.

```python
def normalize(self):
    """
    Normalizes the dataset based on values in min/max arrays
    """
    for mfcc in self.dataset:
        for i in range(40):
            for j in range(301):
                mfcc[i][j] = (mfcc[i][j]-self.min_matrix[i])/(self.max_matrix[i]-self.min_matrix[i])
```

Figure 4.21: Implementasjon av normalisering av MFCC-verdier

4.3.3.3 Omgjøring av lydfiler til MFCC-verdier

For å gjøre om lydfiler til MFCC-verdier beskrevet i 2.5, har vi istedenfor å trenge å implementere alle de matematiske skrittene kunne bruke biblioteket li-

brosa. Når man laster inn lyden gjennom biblioteket, og får et sett av datapunkter, kan vi enkelt omforme disse punktene ved å bruke funksjonen *feature.mfcc()* slik vi har gjort i figur 4.22.

```
y, sr = librosa.load(path, sr=16000, mono=True, duration=self.dur)
y = librosa.util.fix_length(y, sr*self.dur)
mfcc = librosa.feature.mfcc(y, n_mfcc=40, hop_length=160, win_length=400, n_fft=512)
```

Figure 4.22: Implementasjon av omgjøring til MFCC

Det man sitter igjen med i variabelen "mfcc" er en to-dimensjonell matrise med alle MFCC-verdiene ut ifra parameterne gitt.

4.3.4 Validering med egen frase

Da oppgaven ble diskutert med veileder, kom vi frem til at vi skulle lage egne maskinlæringsmodeller som vi skulle trene. Oppdragsgiver var enig i forslaget, slik at systemet kunne bli mest mulig tilpasset for dem.

Den første framgangsmåten vi så for oss var at vi kunne ta et datasett av stemmer som sier tilfeldige fraser (der hver person har sin egen frase), og hente ut karakteristikker fra hver av disse (4.3.1.1). Vi tok først opptak av medstudenter som sier hver sin egne frase (3.12.2). Denne frasen var en valgfri frase, men som en retningslinje ble som oftest en tilfeldig overskrift fra en nettavis brukt. Det ble tatt tre opptak av hver student, slik at det både var nok data for trening og validering av modellene. Alle lydopptak var tatt opp med samme mikrofon med samme opptaksenhet, slik at forholdene var så like som mulig (3.12.2).

Disse lydfilene kunne vi så omforme til et datasett til bruk for trening av modeller. Dette gjorde vi ved å hente ut MFCC-verdier, slik som beskrevet i 4.3.3.3, ved å legge inn framgangsmåten inn i en AudioPreparer (4.3.1.1). For hvert lydopptak vi samlet inn tok vi i utgangspunktet for oss to forskjellige scenarier. Det første scenariet var å mate hele lydfilen inn i modellen, slik at den kunne bestemme selv hvilke deler som var signifikante. TensorFlow og Keras forventer samme form på dataen når man bruker den til modellen. For å sikre at alle fraser ble sagt ferdig, lot vi derfor som at alle lydfilene var ti sekunder lange ved å utvide lengden av lyden med nuller til å tilsvare lengden. For å implementere dette kan man bruke *librosa.util.fix_length* i det man laster inn filen med librosa. Det andre scenariet vi så for oss var å hente ut og bruke tilfeldige delmengder av størrelse et sekund fra lydfilen, slik at den heller så på instanser av talen.

Da vi trente modeller på datasettene fikk vi ganske dårlige resultat uavhengig av lengde, med en valideringsrate på rundt 80 prosent. I praksis er dette veldig dårlig for vår hensikt, da det skal bli brukt til et låsesystem, men da vi aldri hadde jobbet med nevrale nettverk var det faktisk et lovende resultat. For å komme videre måtte datasettet justeres og tilpasses til vårt behov. I første omgang tok vi flere opptak, slik at vi fikk et dobbelt så stort datasett. Bare å

få flere lydfiler økte valideringsraten, men var ikke tilstrekkelig. For teknikken der vi brukte delmengder av et sekund fra hver fil, genererte vi flere tilfeldige utvalg, som gjorde at både CNN (2.2.1) og RNN (2.2.2) fikk bedre nøyaktighet på validering, på nærmere 90 prosent for denne teknikken. Det kan ha kommet av at datasettet ble større, og modellen hadde mer data å trene på.

Da vi fortsatt ønsket høyere nøyaktighet for valideringen, prøvde vi å justere flere detaljer for å dytte resultatene våre oppover. Et eksempel kunne man endre hopp-lengden når vi henter ut MFCC-verdier, som er antall prøver innenfor hver korttids Fourier-transformasjon (2.5.2)(Se *hop_length* i[109]). En annen teknikk var å prøve ut forskjellige hyperparametere til modellene. Dette ble utført ved hjelp av prøv- og feil-teknikk. Resultatet fra denne utprøvingen er vist i 4.3.1.2, der vi forklarer hvilke parametere som ble valgt. Vi klarte med disse modellene å få nøyaktigheten på valideringssettet til å komme seg opp mot 98 prosent, også uavhengig av bruk av CNN eller RNN.

Problemet med modellen var at den var veldig god på å skille mellom data den hadde trent på, men dersom vi introduserte lydopptak som verken var i trening- eller validerings-settet ble disse lydopptakene ofte ikke gitt riktig etikett. Dette er et sikkerhetstilfelle beskrevet i 4.3.7. Det betyr at modellen ikke vet hvilken etikett den skal gi for personer som er ukjente for den, som er et problem dersom vi skal bruke modellene på et låsesystem.

Som følge av disse resultatene kan det også bety at modellene er overtrent. Goyal, Kechit [101] beskriver at dersom man overtrener modeller vil de klassifiserer basert på små detaljer og ikke være generalisert for flere tilfeller. Det vil si at modellen ikke klarer å klassifisere ny data, da de lærer seg for mye av detaljene av dataen som ligger i datasettet. Derfor må vi ta høyde for å ikke trene modellene på for mange epochs ved senere modeller.

For å gi en generell tendens mot å ikke slippe folk inn, prøvde vi å utvide vårt eget datasett med CommonVoice-datasettet beskrevet i 3.12.4. Vi ønsket å gjøre dette da vi falsk-positiv raten var for høy, mens vi kan tåle noen ekstra falske negativer. På den måten fikk vi et utvidet datasett, med tanke på at vi ikke hadde mange opptak i datasettet. Modellene våre fikk valideringsrate på opp mot 100 prosent. Det hjalp også på klassifisering av lydopptak av personer som modellen hadde ikke trent på, siden den var mer skeptisk til hvem den ga validert etikett.

Selv om man fikk bedre resultat blant ukjente mennesker var det ikke noe spesielt mønster knyttet til dette. På grunn av at det var så mange lydfiler i datasettet var det naturlig at flere personer ikke kom inn i systemet. Modellene har muligens trent seg opp på dataen til hvert enkelt opptak, men ikke spesifikt karakteristikkene i talen. En grunn for det kan være at folk sa forskjellige fraser, så det vil selvfølgelig være variasjoner i datasettet. Uavhengig om vi brukte RNN eller CNN var utfallet omtrent det samme, som er en annen

indikator på at modellene trener seg på dataen.

For å sjekke om dette var tilfelle, samlet vi inn opptak av nye personer som sa frasen til personer som lå i det validerte datasettet, og la til disse opptakene i datasettet for de ikke-validerte lydklippene. Tanken her var at modellen nå skulle "tvinges" til å ikke kun se på frasene for prediksjon, men også hvem som ytret frasen (ettersom eksempler på samme frase lå i både det verifiserte og det ikke-verifiserte datasettet). Vi samlet inn slik data for tilnærmet alle frasene som fantes i det validerte datasettet.

Fremgangsmåten ga en tydelig økning i treffsikkerheten til maskinlæringsmodellene - omkring 9 av 10 prediksjoner var riktige. Dette ga en interessant pekepinn for hvordan vi måtte tenke, men å lage et ferdig system baserte på slike datasett var dessverre ikke realistisk. F.eks. om bedriften ønsker å legge til en ny verifisert person med frasen "slipp meg inn i lokalet", ville vi måttet ha samlet inn en knippe lydklipp fra forskjellige ikke-verifiserte personer som også sa "slipp meg inn i lokalet". En annen tanke rundt dette resultatet er at hele lydklippet må være med når vi skal klassifisere hvilken person som sier hva, siden konteksten var viktigere enn selve stemmen. Derfor er det ikke hensiktsmessig å hente ut tilfeldige sekund fra lydfilen, da det ikke er så lett å se hvordan verdiene er i forhold til hverandre.

4.3.5 Frasebasert validering

Modellene våre klarte ikke å klassifisere personer veldig godt når hver person hadde egne fraser, så det neste skrittet å bygge modeller basert på personer som sier samme frase. Det er tenkt at på denne måten vil modellene se mer på karakteristikkene i stemmen enn hva menneskene sier, siden dataen blir mer sammenlignbar med hverandre. Derfor kom vi til enighet med oppdragsgiver om at de ville bruke frasen "Åpne dør" for å låse seg inn i dere kontorlokaler. Derfor lagde vi en webapplikasjon med et spørreskjema vi kunne sende rundt for å samle inn lydklipp av mennesker som sier denne frasen. Systemet for denne applikasjonen er beskrevet i detalj i seksjon 4.7.

Videre måtte vi finne ut av hvordan vi så skulle bearbeide dataen. Da vi i forrige seksjon konkluderte med at vinduer på ett sekund ikke var hensiktsmessig, var den første tanken vår å fortsette å bruke et ti-sekunders vindu som et utgangspunkt. Ved å bruke vinduer på ti sekunder, fulgte det derimot et problem. Dersom man analyserte bildene var det veldig mye tomrom i filene, som gjør at store deler av input ikke har noe med klassifiseringen å gjøre. Derfor prøvde vi å se på et mindre vindu for å snevre inn vinduet til å kun ha med frasen åpne dør.

På de fleste filene vi samlet inn gjennom skjema er det som oftest et sekund i starten der det ikke er noe aktivitet, som mest sannsynlig kommer av litt forsinkelse ved interaksjon med nettsiden når folk tar opp lyd. Vi kunne da

kutte det første sekundet av alle filer for å korte ned lydfilene. Etter å ha kuttet dette sekundet, hadde filer som var lenger enn tre sekunder litt tomrom på slutten. Vi kunne derfor også kutte slutten av lydfilene slik at alle filene totalt var tre sekunder lange. For filer som var da under tre sekunder, la vi på nuller på slutten for å gjøre de like lange.

Ved å bruke de samme prosseseringsteknikkene som før, hadde modellen gode trening- og test-resultater, men gjorde det igjen dårlig med ukjent data. En teori var at modellen forsatt så på dataen i sin helhet fra hver fil og ikke karakteristikkene. Så, for å prøve å få mer varierende data tok hentet vi ut tilfeldige tre sekunder fra opptaksfilene, istedenfor å alltid kutte første og siste del. Dette gjorde vi ti ganger per fil, slik vi i senere viser i 4.4.1.1 i funksjonen *convert_audio_to_desired_length*. Dersom en av filene ikke allerede var 3 sekunder ble hele lydklippet lagt inn et tilfeldig sted i en liste av nuller, slik at start- og slutt-punktet på korte filer også var variert.

For å trene mer på karakteristikkene kunne vi også legge til data som er modifisert med støy. Det gjør at datasettet vil lære seg ujevntheter i dataen for å se på de viktigere verdiene, i tillegg å øke mengde data [112]. En metode for å implementere dette var å legge over eksterne lydklipp av støy over det eksisterende datasettet ved teknikken beskrevet i 4.3.2.1. For å sikre at det var nok prøver av brukere som skulle bli klassifisert som verifisert ble 100 tilfeldige lyder generert på dem, kontra 10 på prøver som ikke skal være verifisert. Etter å ha testet datasettet fikk vi igjen gode resultat på selve testing og treningen. Å innføre denne teknikken klarte å bedre klassifiseringen av personer ukjente for systemet. Resultatet var forsatt ikke optimalt til å bli brukt til et låsesystem, men vi skjønte at det var viktigere med modifisert datasett.

Lydfilene med støy var muligens alt for repetitiv. Selv om vi hadde et stort utvalg av forskjellige filer, vil modellen ved større datasett ha repeterende støyfiler. En idé var å generere støy selv som var helt uforutsigbar for modellen, dette i håp om at modellen lærer seg med om stemmen. Derfor introduserte vi en metode basert på teknikker i 4.3.2.2 for å generere tilfeldig støy på toppen av en lydfil. Tidligere tok vi tilfeldige tre sekunder fra hver lydfil. Dersom man genererer mange filer, er det likevel mulig for modellen å kunne bruke start- og slutt-punktet til der talen begynner for å klassifisere, fordi at hver lydfil fortsatt hadde et sammenhengene start- og slutt-punkt. Derfor introduserte vi også en funksjon, *shift-audio*, som flytter lyden et halvt sekund i en tilfeldig retning. Da kunne ikke modellen bruke start og slutt som et referansepunkt lenger.

Da vi kombinerte disse nye metodene for å gjøre lyden ble resultatet mye bedre på den ukjente dataen. Modellene klarte i større grad å klassifisere ukjente personer som uverifisert. Selv om den var bedre på å klassifisere ukjent data, hadde den en verre nøyaktighet på testdataen enn før, på mellom 70 og 80 prosent. Selv om modellene blir dårligere, vil det muligens si at istedenfor å se på hver enkelte lydfil, ser den heller på hvordan verdiene ligger i forhold til hverandre

spesifikt i enhver tale.

Vi ville så få resultatene til å bli det samme som det vi har opplevd tidligere. Når vi har bygget opp modeller har vi gjort dette ved to etiketter, autorisert og uautorisert. På de uautoriserte personenene genererte vi igjen ti prøver per opptak i systemet (totalt 30 prøver per person), mens autoriserte personer fikk generert 100 filer per person. I utgangspunktet var dette basert på 30 personer som var uautorisert og to som var autorisert. Dette ville forhåpentligvis gjøre at modellene overtrente på autoriserte personer, og lærte seg mer om disse.

En annen ting vi ikke hadde tenkt på, var at lyden fra opptakene samlet inn på nettet var spilt inn på mange forskjellige enheter i forskjellige forhold. Det gjør at lydnivået kan variere, slik beskrevet av Jerad Lewis i [110]. Derfor, for å forhindre at lydnivå på hver mikrofon hadde noe å si, ble så hver lydfil normalisert for prosessering til -20 dbFS fra 4.3.3.1 før prosessering. Da fikk vi begrenset mest mulig ulikheter ved lydfilene som stammet fra ulike opptaksenheter, før de blir gjort om til MFCC-verdier.

Når dette datasettet ble brukt til å trene nevrale nettverk var nøyaktigheten som vanlig opp mot 100 prosent på treningsdata og rundt 80 prosent på testdata. Deretter ble modellen kjørt på lyder som er ukjent for modellen, og det viste seg at vi fikk en Falsk-Positiv-rate på 0, altså at ingen personer som ikke skal ha tilgang kom inn. For å forsikre oss om dette hentet vi inn lydfiler fra ti personer vi visste ikke hadde spilt inn lyd fra før. Her var det kun én person som ble klassifisert som autorisert. Problemet her var at denne personen ble autorisert med veldig høy sannsynlighet. Det betyr at det vil gjelde også flere ukjente personer. Spørsmålet ble da å finne ut av hvordan vi kan hindre at dette skjer. Siden vi ikke hadde så mange lydopptak, er det mulig at datasettet vårt rett og slett var for lite til å dekke et representativt utvalg stemmer for å kunne skille mellom alle mulige stemmer i den norske befolkning.

Mellom CNN og RNN, var det RNN som ga de beste resultatene under denne framgangsmåten. CNN hadde et mer varierende resultat mellom hver test, mens RNN ga alltid feil etikett til den samme personen som er beskrevet i avsnittet over. RNN har også mye raskere treningstid, som er positivt da vi senere skal bruke en server til å trene modellene - i tillegg til at denne modellen må trenes på nytt når det registreres og slettes nye brukere for systemet. Resultatet var et nettverk basert på to LSTM-celler 2.2.2.1 med 256 og 128 noder hver før det ble sendt til klassifikasjonen. Disse LSTM-cellene var definert med 20 prosent dropout mellom dem slik at hver epoke ikke ga samme resultat. Implementasjonen av dette ble vist i 4.3.1.2. Nettverket ble så bygget opp på datasettet med ti epoker.

Lyden i seg selv er nå normalisert etter -20 dbFS, kan det være mulig at vi må normalisere mer data? Det kan godt være at vi får bedre resultat dersom vi prøver å normalisere MFCC-verdiene før vi mater de inn i maskinen.

Spesielt i konvolusjonelle nettverk bør dette ha en positiv innvirking, da konvolusjoner kartlegger verdier basert på et gjennomsnitt av verdier rundt. Derfor implementerte vi metoden beskrevet i 4.3.3.3 for å normalisere MFCC-verdiene. Etter ha normalisert MFCC-verdiene til lydfiler, kan vi begynne å se et klart bilde av stemmeavtrykket. For å vise dette har vi tatt for oss to tilfeldige lydfiler og utført denne prosessen. I figur 4.23 og 4.24, kan vi se likheter mellom disse lydfilene, der vi ser at striper der det ikke er stemme er i samme farge.

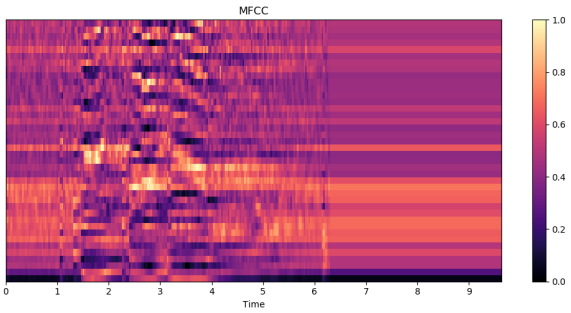


Figure 4.23: Normalisert MFCC 1

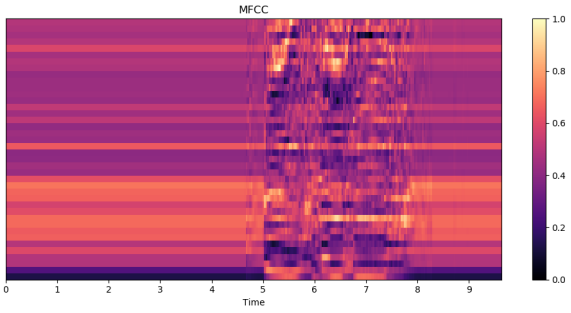


Figure 4.24: Normalisert MFCC 2

Det ville vært fordelaktig å gjøre dette for lydfiler som ikke er augumentert, siden man klart kan se hvilke verdier som representerer stemmen. Når vi utfører normaliseringen på lydfiler som er prosessert med støy derimot, blir de helt annerledes. I figur 4.25 og figur 4.26, ser vi at disse linjene ikke signifikante lenger.

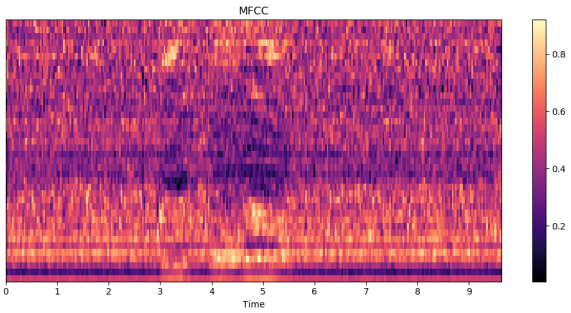


Figure 4.25: Normalisert MFCC med støy 1

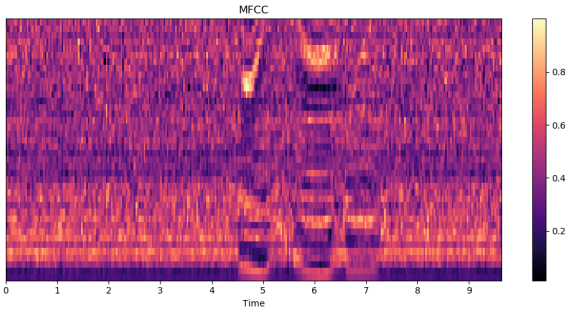


Figure 4.26: Normalisert MFCC med støy 2

I teorien betyr at det er vanskeligere å skille mellom hva som er koeffisienter for stemmen, og hva som bare er støy. Dette vil være ufordelaktig for en modell, da vi vil at stemmen skal vekte mer enn støy. Etter å ha prøvd modellen på lydfilene med denne normaliseringen, ble ikke bare ukjent data dårligere enn før, men testfasen fikk også dårligere prediksjoner. Konklusjonen er derfor at teknikken kan være lur å bruke på data som ikke er prosessert med støy. Dette hadde krevet et mye større og varierende datasett, siden vi ikke kunne generert nye, tilfeldige filer.

Vi har sett at ukjente stemmer for modellen for det meste blir predikert som uverifisert, bortsett fra et par stemmer modellen med ganske høy sikkerhet tror skal være verifisert. Dette kan variere i helhold til mengden data som ligger i det verifiserte datasettet. For å demonstrere variasjoner ved bruk av to forskjellige datasettet, tar vi for oss en en test på RNN nettverker der vi sammenligner bruken av to verifiserte personer opp mot tre. I matrisene under er tabell 7 trent på to verifiserte personer, mens tabell 8 er trent med tre verifiserte.
På valideringssettet til modellen hadde tre verifiserte en dårligere falsk-positiv-rate enn med to verifiserte, men en falsk-negativ-rate på 100 prosent. Fordelen er at på alle data som skulle være verifisert ble det, men fortsatt en viss usikkerhet

		Predicted	
		Positive	Negative
Actual	Positive	0.95	0.05
	Negative	0.06	0.94

Table 7: 2 Verifiserte

		Predicted	
		Positive	Negative
Actual	Positive	1	0
	Negative	0.13	0.87

Table 8: 3 Verifiserte

på personer som ikke skal det. For å se hvordan antall verifiserte påvirker prediksjonene av ukjent data, tok vi også en test av modellen opp mot det ukjente datasettet i tillegg til nye opptak av verifiserte.

		Predicted	
		Positive	Negative
Actual	Positive	0.4	0.6
	Negative	0.2	0.8

Table 9: 2 Verifiserte

		Predicted	
		Positive	Negative
Actual	Positive	0.77	0.23
	Negative	0	1

Table 10: 3 Verifiserte

Da vi hadde introdusert en ny person ser vi at med tre verifiserte i tabell 10 at ingen uverifiserte ble verifisert, og hadde falsk-positiv-rate på 0 til sammenligning med 20 prosent med to verifiserte slik vi ser i tabell 9. Samtidig har vi en høyere sann-positiv-rate, slik at det kreves færre forsøk for å bli verifisert. Fra sammenligning ser vi at modellene varierer ut fra hvor mye data modellen har for å trene seg opp, som også bidrar til riktigere prediksjoner.

4.3.6 Transfer Learning

På grunn av begrenset datasett måtte vi tenke nye tanker for å kunne løse vår problemstilling. Innenfor maskinlæring er det et konsept kalt *Transfer Learning*, som går på å bruke eksisterende trente modeller og deres vekter som basis for ditt eget nettverk, slik Lisa Torrey og Jude Shavlik forklarer i dypere detalj [102]. De eksisterende modellene er trent på et større datasett på større

problemstillinger. Vektene importeres og legges til i vår modell, før man *fryser* lagene. Det vil si at man ikke åpner for at disse lagene blir modifisert. Deretter mater man input gjennom nettverket, og bruker output til dine egne klassifiseringsoppgaver.

I vårt tilfelle vil vi aller helst finne en modell som tidligere har blitt trent opp på MFCC-verdier eller lignende som kan brukes i forhold til lyd. Dessverre er det begrenset hva som finnes på nettet i denne sammenhengen. Det neste steget var da å finne en modell som var trent opp på bilder, siden vår data kan representeres som et bilde. Keras (3.9.2) har en innebygd modell som bruker 50 konvolusjonelle lag, kalt ResNet50 (3.12.6). Denne modellen har tidligere blitt trent opp på en database av bilder, kalt ImageNet, der den ferdig trente modellen sine vekter ligger ute på internett. Etter å ha lastet ned vektene kan vi laste inn dette i ResNet50-modellen og modifisere output-laget til å passe vårt behov, ved å ha en node hver for verifisert og en for uverifisert [103].

Før man kunne teste modellen måtte vi omforme datasettet vårt til å passe til slik modellen var trent opp. ResNet50-arkitekturen er bygget opp på tre input-kanaler, slik at den kan trenes på fargebilder med verdi for Rød, Grønn og Blå. Vi måtte derfor assimilere dette tilfellet med vårt datasett, da vi våre matriser av MFCC-verdier kun er av én dimensjon. Den enkleste metoden for å utvide datasettet til dette formål er ved hjelp av numpy sin *stack*-funksjon, som dupliserer seg selv etter gitte parametere. For å konvertere vår data til å representere tre kanaler måtte vi derfor duplisere matrisen tre ganger med funksjonen.

Den første testen gjennomført var basert på et av de bedre resultatene ovenfor, der vi hadde 30 uautoriserte mennesker og 2 autoriserte. Det viste seg å være utrolig dårlig, der alle verdiene i test-settet ble klassifisert som autorisert. Tanker rundt dette er at alle "bildene" av MFCC-verdiene er for like til at ResNet50 skal skille på dem. Siden ImageNet-vektene er trent opp på bilder av over 100 000 forskjellige etiketter, kunne det være en idé å trene opp modellen på enkeltpersoner.

Vi forsøkte å gjennomføre en test der vi har kun én person som er autorisert for å sjekke om dataen klarer å skille denne personen fra resten av datasettet. Dette var heller ikke en suksess. Treningsdataen får ofte en nøyaktighet på rundt 90 prosent etter 10 epochs, men har veldig dårlig test-rate (rundt 40 prosent). Når vi tegner en forvirringsmatrise av resultatene, er resultatet også alltid uniformt, der all data prediktert som enten positivt eller negativt. Eksempel på dette er vist i tabell 11.
Det tyder på at modellen ikke lærer seg noe om bildet i det hele tatt. Det er mulig at siden vektene fra ImageNet er trent opp på så mange forskjellige etiketter, kan det være vanskelig å skille mellom små detaljer dersom bildene er like.

Med grunnlag om at bildene er like, var en tanke at vi heller kunne tilpasse

		Predicted	
		Positive	Negative
Actual	Positive	1	0
	Negative	1	0

Table 11: Eksempel på forvirringsmatrise ved bruk av ResNet50

nettet til vårt formål. Derfor forsøkte vi å ikke fryse vektene fra ImageNet, men heller kun bruke de som en initialisering. Under trening får denne modellen en nøyaktighet på 100 prosent under trening. Trening var mye tregere enn det det var med fryste vekter, men modellen virket å ta bedre beslutninger. Under trening ved 10 epoker ved ett autorisert individ får den også en rate på opp mot 90 prosent under testing, som kan være lovende. Likevel oppstår samme problem som tidligere, at folk som er ikke skal være autorisert får feil etikett. Det gjør at modellen ikke er noe bedre enn hva vi har gjort tidligere. I tillegg er den tregere å trene opp enn ved nettverk med færre lag, slik som RNN-modellen i 4.3.1.2.

4.3.7 Endelig resultat

For å konkludere det beste resultatet ser vi på utredningene ovenfor. Vi har sett at over hele prosessen har modeller klart å trene seg opp med nøyaktighet på opp mot 100 prosent, og problemet har gått mer på å finne ut av hvordan vi kan få lydklipp som er ukjente for systemet til å bedre seg.

Da vi prøvde å lage modeller på grunnlag av at folk sa sin egen frase, ville modellene trene seg opp bra, men fungerte dårlig med ukjent data. Her var det også et problem at modellen så mer på selve frasen enn karakteristikkene i frasen, som gjorde at uverifiserte brukere kunne si en verifisert frase fra å komme inn. Denne metoden fungerte dersom man hadde lagt til lydopptak av verifiserte fraser spilt inn av uverifiserte mennesker for å gi kontrast mellom verifisert og uverifisert tale. Det er ufordelaktig for bedriften, siden de da må finne mange opptak av mennesker som ikke skal bruke systemet, noe som vil ta lang tid.

Siden vi da visste at vi trengte lydopptak av samme frase, samlet vi inn datasett av mennesker som sa frasen "åpne dør". Fra dette kunne vi hente ut tilfeldige tre sekunder, flytte lyden i tidsserien for å variere start- og slutt-punkt, samt augumentere dataen ved å både kombinere lyden med støyfiler og tilfeldige verdier (4.3.2). Fra dette fikk vi resultat som kunne brukes i et system, der den klarte å klassifisere riktig. Det var fortsatt noen personer i det "ukjente" datasettet som ble klassifisert som en verifisert person med høy sannsynlighet, som er et kritisk sikkerhetsledd under bruk. Dette gjaldt på både CNN og RNN, men RNN var mer konsistent med hvilke personer som ble klassifisert som verifisert av de ukjente personene. I tillegg var RNN raskere å trene, noe som er fordelaktig når vi skal trene modellene på en server. Det implementerte oppsettet for dette

RNN-et er beskrevet i 4.4.1.2.

4.3.8 Test av innspilt lyd

Et annet sikkerhetsspørsmål var om man kan spille av et opptak for modellen av en verifisert bruker, og benytte dette for å låse seg inn. På denne måten vil innbruddstyver f.eks. kunne plassere en gjemt mikrofon utenfor kontorlokalene, og bruke de innsamlede opptakene til å låse seg inn senere. Det er dette som er faren ved bruk biometriske metoder som bruker lagringsmuligheter. For å teste om dette var mulig tok vi lydopptak ved bruk av en iPhone samtidig som vi brukte systemet, for å forsikre oss om at opptaket var et verifisert opptak. Da vi senere spilte av lyden framfor opptaksenheten ble dessverre denne talen også verifisert. Det gjør at det ville vært mulig for en innbruddstyv å plassere en opptaksenhet utenfor systemet, for å så bruke disse opptakene til å komme inn i kontorarealet. Ved å lytte på lydklippet ved å høre om døren åpnes, eller bare anta at brukeren kommer inn når de ikke prøver flere forsøk, kan tyven enkelt vite hvilke forsøk som er validert.
Dette sikkerhetsleddet er kritisk for at systemet skal kunne brukes til enhver tid. Anbefalingen vil da være å kombinere tale med andre verifiseringsmetoder dersom det skal brukes. For eksempel kunne man kombinert dette med ansiktsgjenkjenning, slik som løsningen til den forrige bachelorgruppen hos Avento. Selv om det vil øke sikkerheten ved at flere faktorer må til for å verifiseres, kan dette være et problem ved at det muligens vil kreve enda flere forsøk for å bli verifisert, da begge tjenestene må verifisere på det samme forsøket.

4.4 System for talegjenkjenning

Etter forskning på hvilke modeller som har fungert best etter vår problemstilling omformet vi testmiljøet for modellene til et ferdig system, slik det foreligger i det ferdige produktet. Hovedsakelig er dette for å kun inkludere de mest relevante filene i systemet, men også for å slå sammen metoder for utredning for at alt går automatisk. Fra delen om utredning av modeller i 4.3.4, fant vi ut at den frasebaserte valideringen ga det beste resultatet, og det er denne implementasjonen vi har brukt her. Etter implementasjonen av dette har vi representert det ferdige undersystemet i et klassediagram:

4.4.1 Klasser i systemet

Under beskriver vi mer i detalj klassene fra figur 4.29.

4.4.1.1 AudioHandler

AudioHandler er en utvidelse av AudioPreparer beskrevet i 4.3.1. Forskjellen mellom disse klassene er at AudioHandler også normaliserer og argumenterer dataen før den legges inn i datasettet, slik at prosessen er automatisk. Siden vi i det endelige produktet vet hvordan datasettet utarbeides, kan vi legge inn nye data i datasettet uten å generere alle filene på nytt. I tillegg blir lyden

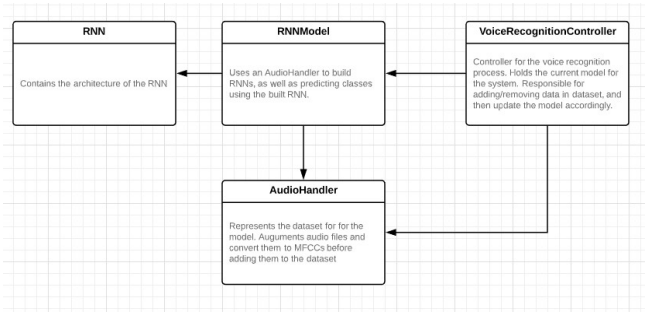


Figure 4.27: Klasse-/Moduldiagram for systemet for talegjenkjenning

automatisk gjort om til MFCC, slik at det kan brukes direkte til trening. Det vil si at *load*-metoden fra 4.3.1.1 blir omgjort til slik som i figur 4.28.

```
def load(self, path, label, id, num_noise=20):
    """
    Loads a single file into the dataset
    """
    # Downsample (on librosa load)
    y, sr = librosa.load(path, sr=16000, mono=True)

    # Generate augumented files
    for i in range(num_noise):
        # Get three second audio files
        new_y = convert_audio_to_desired_length(y, sr=sr, length=3)
        # Shift the audio in random direction
        new_y = shift_audio(new_y, sampling_rate=sr, shift_max=0.5)

        if i % 2 == 0:  # Half of the files will be augumented with random noise
            new_y = add_random_noise(new_y, np.random.uniform(0.01, 0.1))
            new_y = np.asarray(new_y, dtype=np.float32)
        else:   # Half of the files will be merged with noise files
            noise_file = random.sample(os.listdir('VoiceRecogntion/noise/'), 1)
            noise_y, noise_sr = librosa.load('VoiceRecogntion/noise/' + noise_file[0], sr=16000)
            noise_y = convert_audio_to_desired_length(noise_y, sr=16000, length=3)
            new_y = merge_with_noise_file(new_y, noise_y, np.random.uniform(0.1, 0.2))

        # Add the augumented files to the dataset
        self.extract_mfcc_and_add_to_dataset(new_y, label, id)

    # Add the raw file to the dataset
    y = librosa.util.fix_length(y, sr * 3)
    self.extract_mfcc_and_add_to_dataset(y, label, id)
```

Figure 4.28: Metoden *load* i AudioHandler

I denne metoden laster vi først inn lyden, før vi går inn i en for-løkke. Denne tar inn en parameter på antall støyfiler som skal genereres. Deretter bruker den

en metode *convert_audio_to_desired_length*, som henter ut tilfeldige tre sekunder fra filen, og flytter denne i tilfeldig retning. Siden vi både skal ha tilfeldig støy og sammenfletting av lydfiler med støy, genererer vi dette annenhver gang ved å ta modulusen av den nåværende verdien for i, i for-løkken. Etter dette tar vi og henter MFCC-verdiene og legger til i datasettet ved en annen metode *extract_mfcc_and_add_to_dataset*.

4.4.1.2 RNN

Siden det beste resultatet fra utredningen av modeller var ved bruk av RNN, har vi tatt med oss modellen videre til det endelige systemet. I denne filen er det en funksjon *build_rnn_model*, som bygger strukturen til modellen, og returnerer den. Dette er den samme RNN-modellen som i 4.3.1.2.

4.4.1.3 RNNModel

Klassen for å bygge og bruke modeller heter RNNModel. Den tar inn en AudioHandler som parameter, og bruker dataen hentet ut fra denne til å trene en RNN-Modell fra filen ovenfor. Inne i RNNModel-klassen er metode for å bruke modellen til å klassifisere lyddata som blir tatt som input, gjennom TensorFlow sin metode *model.predict()*.

4.4.1.4 VoiceRecognitionController

For å holde orden på klassene ovenfor er klassen VoiceRecognitionController implementert. Klassen skal gjøre det mulig at systemet er kontinuerlig ved å holde orden på hvilken modell som er i systemet, og å lage nye modeller ettersom brukere blir registrert eller slettet i systemet. Her lagres også hvilken AudioHandler som blir brukt, slik at man slipper å trene helt på nytt dersom systemet feiler.

4.4.2 Prossessen i sin helhet

For å bruke delsystemet initialserer man først en VoiceRecognitionController. Her kan man velge om man skal ta inn filadressen til en AudioHandler som parameter dersom man vil bruke en eksisterende. Hvis ikke lages en ny AudioHandler basert på lydfiler som ligger i systemet. Etter AudioHandler er generert, bygger modellen seg opp på datasettet fra denne gjennom RNNModel. Da er systemet klart til bruk.

Fra VoiceRecognitionController kan man så legge til en ny lydfil i AudioHandler ved *add_to_dataset*, og slette alle lydfiler som hører til en email ved *remove_on_id*. Etter at filer er lagt til eller slettet, må man trene modellen på nytt basert på det nye datasettet. Det gjøres gjennom *update_models* som bygger en ny modell i RNNModel. For å bruke modellene til å klassifisere er det også implementert en metode *predict*, som tar inn filstien til den lydfilen som skal predikeres, som parameter.

4.5 System for innspillingsenhet

På innspillingsenheten må vi ha et system for å ta opp opptak under betingelsene som er gitt. Dette systemet skal være koblet opp mot serveren for å autentisere data. Enheten må også kontrollere hva som fysisk skal skje med lyssignal og døren ettersom hvilken etikett opptaket blir klassifisert som.

Enheten lytter etter lyd, og bruker API-et til Google (3.9.4) for å avgjøre hvorvidt det blir sagt "Åpne dør". Isåfall sendes et signal for å vise gult lyd til brukeren, som indikerer at frasen er detektert. Etter dette sendes lyden til serveren for klassifisering og returnerer den antatte etiketten. Ut ifra hvilken etikett lyden blir klassifisert som vil enheten sende ut signal om å:

1. **Lyse grønt og åpne dør** - Dersom lyden blir verifisert
2. **Lyse rødt** - Dersom lyden ikke blir verifisert

I tillegg skal det være en gul diode, som representerer at frasen har blitt detektert og sendt til server for validering, i tilfelle det tar lang tid å få kontakt med serveren. I figur 4.29 kan du se klassestrukturen til innspillingsenheten

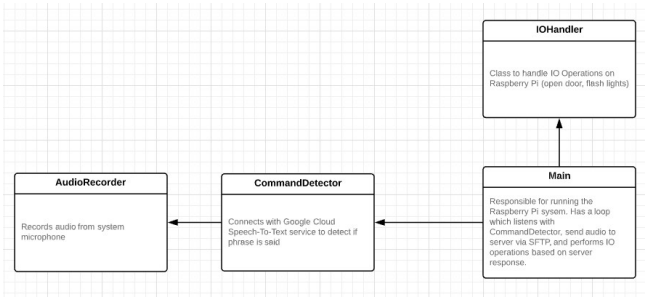


Figure 4.29: Klasse-/Moduldiagram for systemet på inspillingsenheten

4.5.1 Klasser i systemet

Under beskriver vi hvilke klasser vi har utviklet for det endelige systemet for innspillingsenheten.

4.5.1.1 AudioRecorder

Når vi skal ta opp lyd fra systemet har vi gjennom biblioteket PyAudio (3.9.6) laget en klasse AudioRecorder som gjør dette. Klassen er basert på et eksempel funnet i en guide av Google [111], men er tilpasset til vårt behov. Denne tar opp lyd fra systemet med en viss samplingsrate og en chunk-størrelse og lagrer data med denne chunk-størrelsen i en buffer. Denne bufferen er definert som en kø, som gjør at man kan hente chunks i rekkefølge.

4.5.1.2 CommandDetector

Et krav fra oppdragsgiver var at man skal kunne snakke utenfor kontorarealene uten å nødvendigvis ville åpne opp døren. Derfor har vi implementert en funksjonalitet ved hjelp av Google Cloud (3.9.4), som detekterer om frasen for å åpne døren blir sagt. Klassen CommandDetector sender lyd tatt opp fra en AudioRecorder til Google sin tale-til-tekst-tjeneste, som returnerer hvilke ord som blir sagt. På denne måten trenger vi kun å sende inn lydklipp til prediksjon for verifisering, dersom vi detekterer at det er frasen "Åpne dør" som blir sagt.

4.5.1.3 IOController

Siden dette systemet er basert på mange fysiske komponenter har vi hatt behov for en klasse for å håndtere disse. Denne heter IOController. Her skal man kunne sende signal ut fra innspillingsenheten når handlinger blir utført. Det er metode for å låse opp døren, og tre metoder for å sende ut et lyssignal til en rød, grønn og en gul diode for å indikere status. Metoden for å låse opp døren er ikke implementert, på grunn av at vi ikke har tilgang til å teste systemet i person. Likevel kan vi teste om en dør eventuelt ville åpnes opp basert på lyset til den grønne dioden.

4.5.1.4 main

main er et script som syr de øvre klassene sammen. Her er det implementert en løkke som tar opp lyd med AudioRecorder, lytter etter kommando med CommandDetector, sender lydfil til server gjennom SFTP, og til slutt aktiverer hendelser med IOController. Dette vil kjøres om og om igjen så lenge systemet er på.

4.5.2 Oppkobling og styring av dioder

På Raspberry Pi, bruker vi klassen IOController (4.5.1.3) for å styre hvilke signaler som skal sendes ut fra portene. Slik vi har skrevet innledningsvis i seksjonen (4.5) trenger vi en grønn, en rød og en gul diode som skal gi brukeren tilbakemelding ved bruk av systemet. Før vi har koblet opp diodene har vi tegnet et kretsdiagram for systemet, vist i figur 4.30, gjennom webapplikasjonen CircuitLab [94].

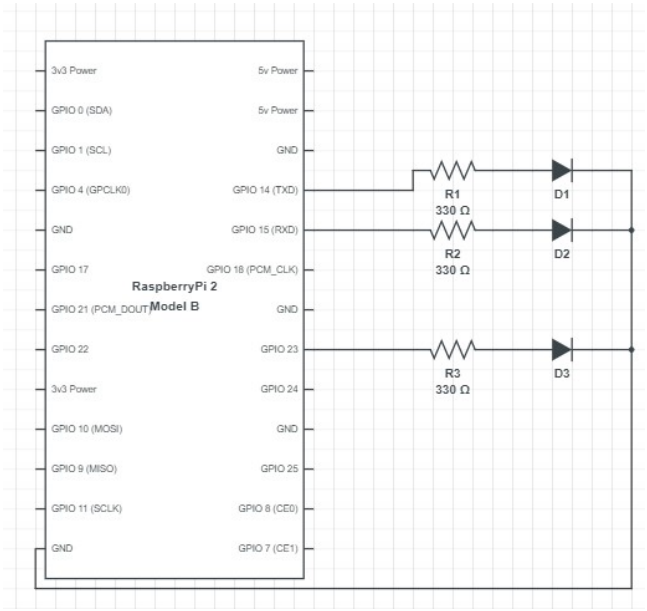


Figure 4.30: Krets brukt for dioder til Raspberry Pi

I figuren ser vi portene til Raspberry Pi på venstre side. For å prøve å bruke så lite porter som mulig har vi brukt en felles pin for jording. For å ha mindre belastning på diodene har vi brukt resistorer på 330 Ohm mellom jord og pinsene. Hver farget diode er så koblet opp til hver sin pin for utsignal: 14 for grønn (D1), 15 for gul (D2) og 23 for rød (D3). Etter å ha konstruert tegningen av kretsen kan vi så koble opp systemet, som i helhet er vist i figur 4.31

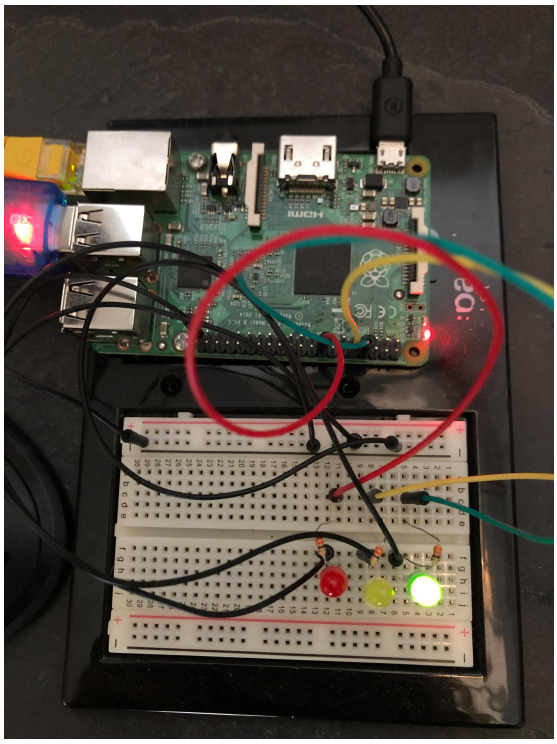


Figure 4.31: Ferdig oppkoblet krets

For å ikke ha felles ledning for alle jord, bruker vi et brødbrett for å skille kretsen. Her er jord koblet opp mot den vertikale negative aksen (-), som man da kan koble opp på de horisontale aksene. Her setter vi så inn pins og kobler opp systemet ut fra kretsen i figur 4.30.

For å styre hvilke dioder som skal være på eller av, kan vi bruke python-biblioteket RPi.GPIO (3.9.23). Først velger man hvilken modus man skal bruke ved *GPIO.setmode()*. Her kan man enten bruke *GPIO.Board* eller *GPIO.BCM* som parameter. Fordelen ved å bruke *BCM* kontra *Board* som parameter er at den bruker et sett predefinerte GPIO-pinner som gjør at de samme pinnene lyses opp uavhengig av hvilken Raspberry Pi-modell man bruker, da modeller har et varierende antall. Board vil kun kartlegge pinnene ut fra den gjeldene Pi-en. Pin-opsettet til Raspberry Pi 2 Model B som vi bruker er vist i figur 4.32.

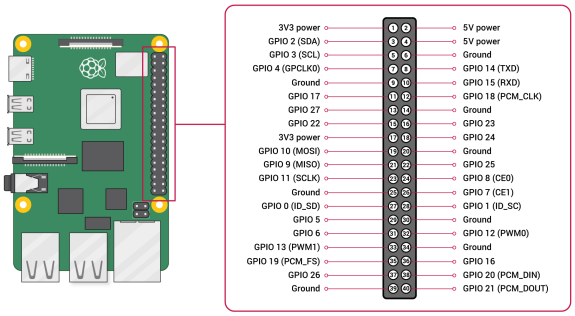


Figure 4.32: Raspberry Pi 2 Model B Pin Layout [95]

Etter å ha definert modus kan vi så begynne å initialisere hva ulike pins skal gjøre. Her kan man enten sette funksjonaliteten til en pin ved *GPIO-setup()*-funksjonen. Denne tar inn to parametere, hvilken pin man skal sette opp, og *GPIO.IN/GPIO.OUT* ut fra om du ønsker at pinen skal representere et input eller output, respektivt. For eksempel initialiserer vi den grønne dioden som et output ved å skrive *GPIO.setup(GREEN_PIN, GPIO.OUT)*. Påfølgende blir dette gjort for de andre diodene. Deretter kan vi sette status på pinen ved bruk av *GPIO.output()*. Denne funksjonen tar også inn et pin-nummer, men istedenfor å sette dette til input/output setter vi True eller False basert på om vi vil at dioden skal lyse (True) eller slukkes (False). For å lyse opp den grønne dioden vi nettopp initialiserte, skriver vi derfor *GPIO.output(GREEN_PIN, True)* i koden vår.

4.6 Administrative Systemer

For at løsningen vi har utviklet skal kunne beskrives som komplett, har det som nevnt vært nødvendig å få på plass et administrativt system med et webgrensesnitt, slik at administratorer enkelt kan se på loggføring og gjøre brukerbehandling, med mer. HTML (3.7.2) og CSS (3.7.3) har henholdsvis blitt brukt for å definere innhold og design for nettsiden, mens Python (3.7.1) i samvirke med Flask (3.9.15) og andre Python-bibliotek har blitt brukt for å utvikle backend-logikken. JavaScript (3.7.4) er også blitt brukt, både for å gjøre interaksjon enklere for brukeren (f.eks. ved å sjekke input på klientsiden), men også i forbindelse med backend-logikk for å håndtere innsending av lydfiler.

Bildet under viser en oversikt over de forskjellige klassene/modulene vi har utviklet og som til sammen utgjør den administrative nettsiden. Vi vil gjennomgående vise til disse klassene for å forklare hvordan den administrative nettsiden er utviklet og sammensatt. I tillegg til dette kommer også HTML-, CSS- og JavaScript-filer. En oversikt over disse finnes under seksjonen for systemarkitektur (4.7.3).

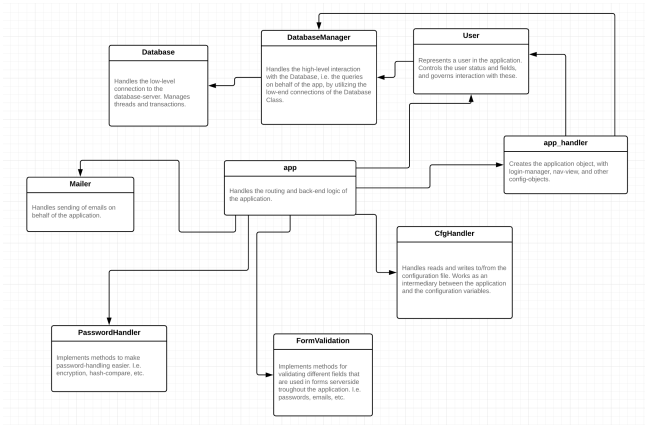


Figure 4.33: Klasse-/Moduldiagram for administratornettsiden

4.6.1 Ruting og generell logikk

Selve Flask-objektet (3.9.15) som representerer applikasjonen lages i modulen "apphandler" (4.33). Når objektet blir laget, initialiseres også andre "hjelpe-objekter" som applikasjonen trenger for å gjøre jobben sin. Dvs. en LoginManager (3.9.15.1) for autorisering og autentisering, samt en "DatabaseManager" og en "Mailer" (4.33) for henholdvis interaksjon med databasen og for å kunne sende ut epost.

Den overordnete logikken for applikasjonen finnes i "app"-modulen (4.33). Her brukes app-objektet som lages i "apphandler" (4.33) til å konstruere konvoluttadresser for applikasjonen, og disse blir tilknyttet logikk som skal kjøre på serveren når en bruker besøker den gitte konvolutadressen. Et eksempel på dette er vist under 4.34, for konvolutadressen "/log", der administratorer kan se loggføring for bruk av systemet for talegjenkjenning.

```
@app.route('/log', methods=['GET'])
@login_required
def log():
    """
    Endpoint for the log-tab of the application web-page.
    Admins will be displayed all the log-entries from the database.
    """
    log_data = app.config["DATABASE"].retrieve_log_data()
    return render_template('log.html', title="Log", log=log_data)
```

Figure 4.34: Hva som skjer på serveren når noen besøker '/log'

Først verifiserer serveren at personen som prøver å aksessere siden er logget inn, og at HTTP-forespørselen som er sendt til adressen er av typen GET. Selv om det i dette tilfellet ikke er noen åpenbar sikkerhetstrussel å tillate andre HTTP-metoder, har vi gjennomgående fulgt prinsippet om minst tilgang (2.11.6) for å være på den sikre siden. Deretter hentes logg-informasjonen fra databasen, som

ved hjelp av Jinja2 (3.9.16) blir dynamisk plassert i HTML-dokumentet som returneres til brukeren (dersom han/hun er logget inn og bruker HTTP-GET).

```
{% extends 'layout.html' %}
{% block container %}
<div class="user_table">
    {% if log %}
        <table id="table_sortable">
                <th onclick="sortTable(1);">Time</th>
                <th onclick="sortTable(2);">Status</th>
                <th onclick="sortTable(3);">Validation</th>
            {% for data in log %}
                <tr>
                    <td>
                        {# Timestamp for the entry #}
                        {{ data[1] }}
                    </td>
                    <td>
                        {# The status of the entry #}
                        {{ data[2] }}
                    </td>
                    <td>
                        {# Probability on entry #}
                        {{ data[3] }}
                    </td>
                </tr>
            {% endfor %}
        </table>
    {% else %}
        <p>No data in the log yet. This will be added automatically as soon as the service is used.</p>
    {% endif %}

</div>

<script src="/static/js/table.js"></script>

{% endblock container %}
```

Figure 4.35: HTML / Jinja for loggføring

Bildet ovenfor (4.35) viser et eksempel på hvordan Jinja2 (3.9.16) gjennomgående blir brukt for den administrative nettsiden. Som kommer frem fra figur 4.34, blir loggføringene gitt videre til template-motoren i form av en variabel kalt "log". Vi kan så enkelt aksessere denne dataen, omtrent som om det var vanlig Python, ved å plassere koden innenfor klammeparenteser. Dette kommer til god nytte over hele nettsiden, da vi har behov for å inkludere dynamisk data i HTML-sidene ved mange tilfeller (loggføring, brukerbase, innstillinger osv.).

4.6.2 Tilbakemelding til bruker

Å gi god tilbakemelding til brukeren er et viktig aspekt ved å utvikle nettsider, da det i stor grad er nødvendig for å gjøre siden brukervennlig. En kombinasjon av JavaScript (3.7.4), Jinja2 (3.9.16) og Flask-Flashing (3.9.15.2) gjør at vi kan gi brukeren tilbakemeldinger på en konsis måte som er enkel å forholde seg til. Den lille kodesnutten med HTML/Jinja under (figur 4.36), inkluderes i hver eneste HTML-side som blir returnert til brukeren (fordi den inkluderes i malen som brukes for all HTML), slik at HTML-koden vil inneholde en egen seksjon med de meldingene som trengs å vises til brukeren. Deretter brukes JavaScript-koden under (figur 4.37), som også inkluderes i malen som brukes for all HTML, til å automatisk hente ut de meldingene som skal vises til brukeren.

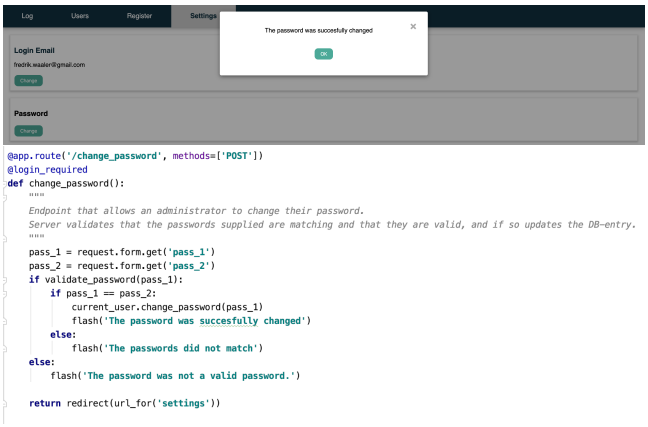
```jinja
{% with messages = get_flashed_messages() %}
        {% if messages %}
            <div class="popup">
                <div class="popup-content">
                <span class="popup-close">×</span>
                <div class="popup-messages">
                    {% for message in messages %}
                        <p>{{ message }}</p>
                    {% endfor %}
                </div>
                <input class="form-button popup-close-btn" type="submit" value="OK">
                </div>
            </div>
        {% endif %}
{% endwith %}
```

Figure 4.36: Generisk HTML/Jinja for å inkludere flashes

```javascript
/**
 * This scripts displays any pop-ups if there are any.
 * Blurs the document behind the popup, and associates the popup with a close-button.
 */

// Retrieve the required fields from HTML-doc.
let popup = document.getElementsByClassName( classNames: "popup");
let close = document.getElementsByClassName( classNames: "popup-close");
let close_btn = document.getElementsByClassName( classNames: "popup-close-btn");

// If there is a popup, noscroll for body
if (popup.length > 0) {
    document.body.style.overflow = "hidden";
}

// If there is an associated close-section (i.e. outside the popup), make it close the popup on click.
if (!isEmpty(close)) {
    close[0].onclick = function () {
    popup[0].style.display = "none";
    document.body.style.overflow = "auto";
    }
}

// If there is an associated close-button for the popup, make it close the popup on click.
if (!isEmpty(close_btn)) {
    close_btn[0].onclick = function () {
    popup[0].style.display = "none";
    document.body.style.overflow = "auto";
    };
}

/**
 * Helper function the check whether objects in JS are empty.
 * @param obj: The object
 * @returns {boolean}: True if empty, else False.
 */
function isEmpty(obj) {
    for(let key in obj) {
        if(obj.hasOwnProperty(key))
            return false;
    }
    return true;
}
```

Figure 4.37: Generisk JS for å vise flashes

Koden lager så en modal som vises frem til brukeren. Det som gjør denne løsningen så elegant, er at man som utvikler nå enkelt kan gi tilbakemelding til brukeren på en uniform og grafisk tilfredstillende måte, ved kun forholde seg til Flask sin Flash-funksjon (3.9.15.2). JavaScript og Jinja2 vil automatisk ta seg av uthenting og fremvisning av meldingene. Et eksempel på hvordan dette gjøres i praksis, og hvordan det blir seende ut for brukeren, kan man se under - i hvilket tilfelle brukeren har fått tilbakemelding i forbindelse med endring av passord på siden for innstillinger (figur 4.38).




```python
@app.route('/change_password', methods=['POST'])
@login_required
def change_password():
    """
    Endpoint that allows an administrator to change their password.
    Server validates that the passwords supplied are matching and that they are valid, and if so updates the DB-entry.
    """
    pass_1 = request.form.get('pass_1')
    pass_2 = request.form.get('pass_2')
    if validate_password(pass_1):
        if pass_1 == pass_2:
            current_user.change_password(pass_1)
            flash('The password was succesfully changed')
        else:
            flash('The passwords did not match')
    else:
        flash('The password was not a valid password.')

    return redirect(url_for('settings'))
```

Figure 4.38: Logikk som flasher tilbakemelding til bruker og grafisk fremvisning av tilbakemeldingen

4.6.3 Databaser og datalagring

Gjennomgående i utviklingen av applikasjonen, har det også vært nødvendig å forholde seg til håndtering og lagring av data som vedvarer fra besøk til besøk, f.eks. loggføringene som hentes fra database på figur 4.34. Det er blitt brukt PostgreSQL (3.8.3) for å lagre data for den administrative applikasjonen. Figuren under 4.39 gir en grafisk fremstilling av hvordan data er strukturert og representert i databasen.

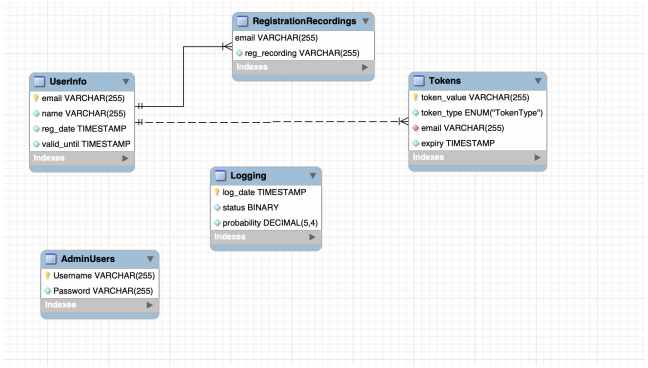


Figure 4.39: ER-Diagram for database til administratornettside

Tabellen "AdminUsers" lagrer innloggingsinformasjon for administratorbrukere. "UserInfo" lagrer brukerinformasjon om brukere av systemet for talegjenkjenning, slik at hver entitet i denne tabellen representerer en enkelt bruker. Dette gjør det enkelt å lagre flere tokens pr. bruker i "Tokens", som i vårt tilfelle brukes disse i forbindelse med f.eks. registrering og tilbakestilling av passord,

for å sikre at uvedkommende ikke skal kunne registrere seg eller endre tilfeldige passord, selv om de har tilgang til konvolutadressen for å endre passord. I tilleg kan man også da enkelt lagre flere lydklipp pr.bruker i "RegistrationRecording". Logging tabellen brukes for å ta vare på informasjon om forsøk på innlåsing via systemet.

Selv om vi har opprettet databasen med tabeller, lagret data, definert prosedyrer osv., gir ikke dette oss mye nytte dersom vi ikke kan "snakke" med databasen gjennom Python. For at vi skal kunne gjøre dette, slik at lagret data skal kunne håndteres gjennom backend-logikk, bruker vi Psycopg2 (3.9.17). Som kan sees fra kodeutdraget nedenfor (figur 4.40), brukes "pool" modulen fra Psycopg2 til å opprette et "basseng" med forbindelser til databasen. Bassenget med forbindelser blir så innkapslet i "Database"-klassen. På denne måten kan en forbindelse til databasen opprettes én gang - ved å initialisere et objekt av Database-klassen. For å gjøre operasjoner mot databasen trenger man bare å hente en forbindelse fra bassenget.

```python
import psycopg2
from psycopg2 import pool
import numpy as np
from psycopg2.extensions import register_adapter, AsIs
psycopg2.extensions.register_adapter(np.int64, psycopg2._psycopg.AsIs)


class Database:
    """
    Class represents a database.
    The database takes use of a connection pool to save run time on operations.
    """

    def __init__(self, **kwargs):
        """
        The database object has a connection-pool that is used for managing connections to the specified database.
        Note: Must be provided with sufficient data to establish a connection.
        :param database: The name of the database to connect to
        :param host: The host address
        :param user: The user logging in to the dbsm
        :param password: Users password for dbsm
        :param port: Port for external host
        """
        self.connectionPool = pool.SimpleConnectionPool(minconn=1, maxconn=10, **kwargs)

    def get_connection(self):
        """
        Returns a connection from the connection pool to the database.
        @:return Returns a connection from the connection pool to the database.
        """
        return self.connectionPool.getconn()

    def return_connection(self, connection):
        """
        Is used to put back an active connection to the connection pool.
        :param connection: The active connection
        """
        self.connectionPool.putconn(connection)

    def close_all_connections(self):
        """
        Closes all active connections for the connection pool.
        """
        self.connectionPool.closeall()
```

Figure 4.40: Python-klassen "Database"

For å gjøre interaksjon mot databasen enda enklere, har vi pakket inn en forbindelse fra bassenget ovenfor (figur 4.40) i en egen klasse, "CursorFromConnectionPool". Kodeutdraget nedenfor (figur 4.41) viser klassen.

```python
class CursorFromConnectionPool:

    def __init__(self, **kwargs):
        self.database = Database(**kwargs)
        self.connection = None
        self.cursor = None

    def __enter__(self):
        """
        When a Cursor is used, we get a connection from the pool and a cursor on that connection,
        the cursor is returned.
        :return: A cursor for the database.
        """
        self.connection = self.database.get_connection()
        self.cursor = self.connection.cursor()
        return self.cursor

    def __exit__(self, exc_type, exc_val, exc_tb):
        """
        When we are done with the cursor, it will close itself,
        make sure that it's connection commits and then puts itself back in the database.
        If some error occurs when using the cursor, it will by default perform a rollback on its connection.
        """
        if exc_val is not None:
            self.connection.rollback()
        else:
            self.cursor.close()
            self.connection.commit()
        self.database.return_connection(self.connection)
```

Figure 4.41: Python-klassen "CursorFromConnectionPool"

Ved å pakke inn forbindelsen i en egen klasse, sikrer man at forbindelser mot databasen alltid opprettes og lukkes på en ordentlig måte, slik at man ikke risikerer at Databasen befinner seg i en ugyldig tilstand. Som man kan se fra utdraget med kode fra klassen, brukes Python sine innebygde metoder "__enter__" og "__exit__". På denne måten vil en forbindelse til databasen automatisk åpnes og lukkes hvis den blir brukt med en "with"-funksjon [26]. I tillegg vil alle transaksjoner mot databasen lagres automatisk når en forbindelse brukes, og dersom det oppstår en feil i Python mens en forbindelse er åpen, vil transaksjonen som styres av den aktuelle forbindelsen automatisk rulles tilbake (rollback) for å sikre at databasen ikke havner i en inkonsekvent tilstand.

Dette gjør at interaksjon mot databasen fra Python kan gjøres veldig enkelt - f.eks. er kodeutdraget under (figur 4.42) alt som skal til for å hente ut innloggingsinformasjon for en administrator med den spesifiserte eposten. All annen konkret interaksjon mot databasen er strukturert i lignende metoder i klassen "DatabaseManager" (figur 4.33), som gir utviklere et overordnet grensenitt mot databasen. På denne måten trenger man kun bekymre seg med spesifikke spørringer mot databasen dersom man ønsker å videreutvikle systemet, og ikke med de underliggende forbindelsesmekanismene som er vist i "Database" og "CursorFromConnectionPool".

```
def get_admin_by_email(self, email):
    """
    Queries the database for user info belonging to the admin with the specified email.'
    The user info is returned as a single tuple.
    :param email: The email to search by.
    :return: If the email is present in the database, the corresponding data is returned as a tuple.
    If not, None is returned.
    """
    with self._database as cursor:
        cursor.execute('SELECT * FROM get_admin(%s)', (email,))
        return cursor.fetchone()
```

Figure 4.42: Funksjon i klassen DatabaseManager for å hente innloggingsinformasjon for administrator

Et siste aspekt ved tilkobling til databasen gjennom Python, er at vi må lagre informasjon for tilkobling (tjener, skjema, bruker, passord) et sted, slik at applikasjonen automatisk skal kunne koble seg til databasen ved oppstart. For at denne dataen ikke skal kunne "lekke" til angripere gjennom uheldige feilmeldinger eller lignende, er det nødvendig at denne informasjonen ikke hardkodes inn i kildekoden. Hardkoding av slik informasjon i kildekoden vil også by på andre sikkerhetstrusler, ettersom koden f.eks. er offentlig tilgjengelig på GitHub (3.8.1). For å løse dette har vi lagret informasjon for tilkobling i en egen konfigurasjonsfil, utenfor kildekoden. Denne hentes ut gjennom en egenlaget klasse som "wrapper" ConfigParser 3.9.22, kalt "CfgHandler" (figur 4.33), som sikrer at konfigureringsinformajson ikke lekker gjennom f.eks. feilmeldinger, og som ellers håndterer sikker lesning og skriving til konfigurasjonsfilen på vegne av applikasjonen.

4.6.4 Registrering

Et viktig aspekt ved administratorsiden, var for oppdragsgiver at en administrator skulle ha mulighet for å registrere nye brukere. Som vist på skjermbildet under (figur 4.43), har en administrator muligheten for å registrere både nye brukere for talegjenkjenningssystemet, samt nye administratorer.

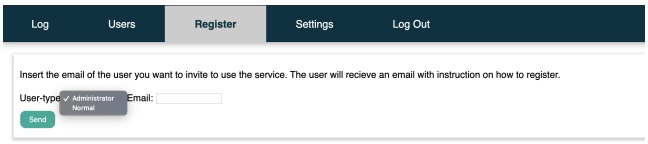


Figure 4.43: Side for registrering av nye brukere

Epost for invitasjon - Når en administrator registrerer en ny bruker ved å klikke på send, vil en epost med en invitasjonslink sendes til den spesifiserte epost-adressen. For å sende ut disse epostene, brukes klassen "Mailer" (figur 4.44).

```python
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from jinja2 import FileSystemLoader, Environment
from CfgHandler import config_handler

class Mailer:

    # The mailer uses this SMTP-server to send mails.
    # Port 587 is secure TLS on google smtp-server.
    server_name = "smtp.gmail.com"
    server_port = 587
    smtp_server = smtplib.SMTP(server_name, server_port)

    # Jinja loader to load templates as mail-content
    env = Environment(loader=FileSystemLoader('./templates/'))

    def __init__(self, email, password, envelope_name=None):
        """
        The Mailer is used to send mails on behalf of the specified user.
        The envelope name is the envelope sender of the mail. By default this equals the from-sender.
        :param email: The email to send from
        :param password: Password to login to email
        :param envelope_name: The envelope-sender of the email
        """
        self.email = email
        self.password = password
        if envelope_name:
            self.envelope_name = envelope_name
        else:
            self.envelope_name = self.email

    def change_authentication(self, email, password):
        """
        Changes the authentication parameters (email and password) for the mailer.
        :param email:
        :param password:
        :return:
        """
        self.email = email
        self.password = password

    def open_server(self):
        """
        Must be used before sending mails with the mailer.
        Readies the mailer for use with encryption and the set mail-login.
        """
        self.smtp_server.connect(self.server_name, self.server_port)
        self.smtp_server.ehlo()
        self.smtp_server.starttls()
        self.smtp_server.login(self.email, self.password)

def close_server(self):
    """
    Should be used after sending mails with the sender to ensure that the connection does not stay open.
    """
    self.smtp_server.quit()

def send_mail(self, subject, content, receiver, mime_type="plain"):
    """
    Sends a mail to the specified receiver from the mail that was specified as login.
    :param subject: The subject of the mail
    :param content: The content of the mail
    :param receiver: The receiver of the mail
    :param mime_type: What kind of MIME-type is the content (i.e. 'plain', 'html' etc. )
    """
    message = self.generate_message(subject, content, receiver, mime_type)
    self.smtp_server.sendmail(self.email, receiver, message.as_string())

def generate_message(self, subject, content, receiver, mime_type):
    """
    Generates a mail of the given mime-type with the given content.
    :param subject: The subject of the mail
    :param content: The content of the mail
    :param receiver: The receiver of the mail
    :param mime_type: The MIME-type for the mail
    """
    msg = MIMEMultipart('alternative')
    msg['subject'] = subject
    msg['from'] = self.email
    msg['to'] = receiver
    content = MIMEText(content, mime_type)
    msg.attach(content)
    return msg
```

Figure 4.44: Klassen "Mailer"

Mailer-klassen bruker smtplib (3.9.21) og Google sin åpne SMTP-server [113] for å gi utvikleren et enkelt grensenitt mot en SMTP-klient som kan brukes til å sende ut epost. Et objekt av klassen initialiseres med innloggingsinformasjon til eposten-kontoen man ønsker og bruke, og utsending av epost kan deretter enkelt gjøres ved å bruke metoden "send_mail". Et eksempel på dette er vist under i figur 4.45, der metoden brukes for å sende ut en epost med HTML-innhold,

i forbindelse med registrering av nye brukere til systemet for talegjenkjenning. På denne måte kan man som utvikler enkelt sende ut epost ved hjelp av et "Mailer"-objekt, så lenge man passer på å åpne og lukke serveren før og etter bruk.

```
def send_registration_admin(self, reciever, token):
    """
    Sends a mail to the reciever, informing about registration for the service as an admin.
    :param reciever: The email of the reciever
    :param token: The token to use for registration
    """
    template = self.env.get_template('empty.html')
    href = '<a href="https://{}/admin_reg?token={}">{}</a>'.format(config_handler.get_value('SETTINGS', 'BASE_URL'), token, "Link til registrering")
    content = """\n
            Hei!
            Du mottar denne linken fordi du inviteres som administrator til Avento sitt talegjenkjenningssystem for inngang til kontorlokalene.
            Du kan registrere deg her: LINK

            Mvh Avento
    """
    content = content.replace('LINK', href)
    mail = template.render(content=content, title='Admin Registration')
    self.send_mail("Registration Administrator", mail, reciever, mime_type='html')
```

Figure 4.45: Funksjon for å sende invitasjonslink over mail

Oppgaven med å sende ut mail med invitasjonslinker, slik at nye brukere (administratorer og for talegjenkjenning) kan registrere seg, er i hovedsak den viktigste oppgaven til Mailer-klassen. Figur 4.45 viser som sagt hvordan dette gjøres for registrering til brukere for talegjenkjenningssystemet. Dersom man blir tilsendt en slik link, slik at man får tilgang til siden for registrering, vil man bli veiledet gjennom et registreringsskjema som kulminerer i skjermen som er vist nedenfor (figur 4.46).

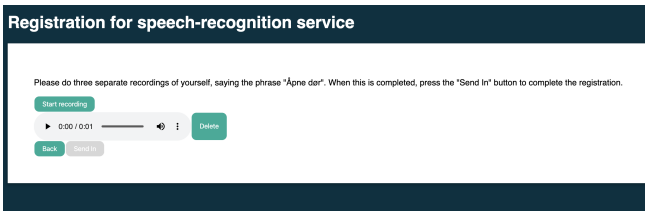


Figure 4.46: Registreringsskjema for nye brukere - ett av tre opptak er tatt

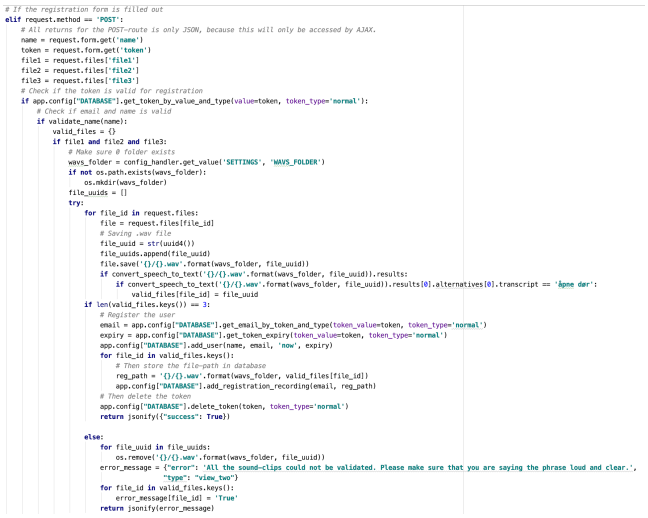
En bruker som registrerer seg vil bli bedt om å ta tre opptak av seg selv som sier frasen "åpne dør", og han/hun vil som vist ha mulighet til å høre på lydklippene som tas opp før de sendes inn.

Et problem vi støtte på når vi skulle utvikle skjema for registrering, var hvordan vi skulle håndtere opptak og innsending av lydfiler. JavaScript har en innebygd metode (navigator.mediaDevices.getUserMedia) som gir tilgang til opptaksenheten på brukerens dataenhet, men denne funksjonaliteten blokkeres av f.eks. iOS [114]. Det var dog viktig at siden for registrering var tilgjengelig på alle slags enheter, siden vi ønsket et representativt utvalg med opptak. Vi landet derfor på RecorderJS (3.9.20) for å gjøre opptak gjennom nettleser, som gjør det mulig å ta opptak på alle slags enheter og nettlesere. Når det kommer til innsending av data, har HTML (3.7.2) ingen innebygd støtte for innsending av lydfiler gjennom sin vanlige mekanisme for innsending av data, "forms".

Det er dog mulig å sende inn lyddata gjennom AJAX (3.8.2), ved å sende inn JavaScript-"FormData" istedenfor et klassisk HTML-form. Dette byr dog på en ny utfordring: AJAX håndterer HTTP-forespørsler gjennom en separat tråd enn den nettleseren opprinnelig bruker for å kommunisere med serveren, dermed kan vi ikke bruke Flask (3.9.15) som vi vanligvis ville gjort for å sende brukeren videre til en ny side etter registrering, gi tilbakemelding og redigere innholdet i HTML basert på innsendt data gjennom Jinja (3.9.16) - fordi dette ikke ville bli sendt tilbake til tråden som bruker for å styre logikken i brukerens nettleser. Vi har løst problemet med en alternativ tilnærming der vi kombinerer det faktum at JavaScript kan brukes for å styre data som fremvises i nettleser, med det faktum at AJAX også kan motta data fra serveren.

```
/**
 * Sends the registration-recordings to the server using an AJAX-call when the send-button is clicked on.
 */
$('#send').click(function (e) {
    e.preventDefault(); // Dont send the html-form where the data is originally collected.
    // Send as FormData instead.
    let form = new FormData();
    form.append( name: 'file1', clip1,  fileName: 'clip1');
    form.append( name: 'file2', clip2,  fileName: 'clip2');
    form.append( name: 'file3', clip3,  fileName: 'clip3');
    form.append( name: 'name', name);
    let token = document.getElementById( elementId: 'token').value;
    form.append( name: 'token', token);
    $.ajax({
        type: 'POST',
        url: '/user_reg',
        cache: false,
        headers: { "cache-control": "no-cache" },
        data: form,
        processData: false,
        contentType: false,
        async: false,
        // On response from server
        success: function (data) {
            // If server responds with success, reg is completed, redirect user to /successful_reg.
            if (data.success) {
                window.location.href = '/successful_reg';
            } // If server does not respond with success...
            else {
                // If there was an error in the first part of the form.. reset recordings and start form over again
                if (data.type === 'view_one') {
                    document.getElementById( elementId: 'delete1').click();
                    document.getElementById( elementId: 'delete2').click();
                    document.getElementById( elementId: 'delete3').click();
                    goto_view_one();
                    errorAlert.style.display = 'block';
                    errorAlert.innerText = data.error;
                } // If there was an error in the second part of the form.. Delete the sound clips that was not valid
                else if (data.type === 'view_two') {
                    if (!data.file1) {
                        document.getElementById( elementId: 'delete1').click();
                    }
                    if (!data.file2) {
                        document.getElementById( elementId: 'delete2').click();
                    }
                    if (!data.file3) {
                        document.getElementById( elementId: 'delete3').click();
                    }
                    // Display error message
                    errorAlert2.style.display = 'block';
                    errorAlert2.innerText = data.error;
```

Figure 4.47: Logikk for registrering i JS med AJAX



```python
# If the registration form is filled out
elif request.method == 'POST':
    # All returns for the POST-route is only JSON, because this will only be accessed by AJAX.
    name = request.form.get('name')
    token = request.form.get('token')
    file1 = request.files['file1']
    file2 = request.files['file2']
    file3 = request.files['file3']
    # Check if the token is valid for registration
    if app.config["DATABASE"].get_token_by_value_and_type(value=token, token_type='normal'):
        # Check if email and name is valid
        if validate_name(name):
            valid_files = {}
            if file1 and file2 and file3:
                # Make sure @ folder exists
                wavs_folder = config_handler.get_value('SETTINGS', 'WAVS_FOLDER')
                if not os.path.exists(wavs_folder):
                    os.mkdir(wavs_folder)
                file_uuids = []
                try:
                    for file_id in request.files:
                        file = request.files[file_id]
                        # Saving .wav file
                        file_uuid = str(uuid4())
                        file_uuids.append(file_uuid)
                        file.save('{}/{}.wav'.format(wavs_folder, file_uuid))
                        if convert_speech_to_text('{}/{}.wav'.format(wavs_folder, file_uuid)).results:
                            if convert_speech_to_text('{}/{}.wav'.format(wavs_folder, file_uuid)).results[0].alternatives[0].transcript == 'åpne dør':
                                valid_files[file_id] = file_uuid
                    if len(valid_files.keys()) == 3:
                        # Register the user
                        email = app.config["DATABASE"].get_email_by_token_and_type(token_value=token, token_type='normal')
                        expiry = app.config["DATABASE"].get_token_expiry(token_value=token, token_type='normal')
                        app.config["DATABASE"].add_user(name, email, 'now', expiry)
                        for file_id in valid_files.keys():
                            # Then store the file-path in database
                            reg_path = '{}/{}.wav'.format(wavs_folder, valid_files[file_id])
                            app.config["DATABASE"].add_registration_recording(email, reg_path)
                        # Then delete the token
                        app.config["DATABASE"].delete_token(token, token_type='normal')
                        return jsonify({"success": True})

                    else:
                        for file_uuid in file_uuids:
                            os.remove('{}/{}.wav'.format(wavs_folder, file_uuid))
                        error_message = {"error": 'All the sound-clips could not be validated. Please make sure that you are saying the phrase loud and clear.',
                                         "type": "view_two"}
                        for file_id in valid_files.keys():
                            error_message[file_id] = 'True'
                        return jsonify(error_message)
```

Figure 4.48: Logikk for registrering i Python

Figur 4.47 og 4.48 over, viser hoveddelen av logikken for registrering til systemet for talegjenkjenning, som henholdsvis er en kombinasjon av JavaScript med AJAX, og Python. Lydklippene sendes først inn til serveren ved hjelp av AJAX. Deretter gjøres validering av token for registrering, annen form-data osv., før Google Speech-To-Text (3.9.4.1) brukes for å validere at det faktisk blir sagt ”åpne dør” i de innsendte frasene. Filene lagres lokalt før de valideres, da dette er noe som kreves av Google Speech-To-Text. Selve valideringen er et kritisk steg, da de innsendte lydklippene brukes til å trene opp maskinlæringsmodellene. Ved å trene opp modellen på annen data, ødelegges integriteten til datasettet, og man risikerer at brukere med onde hensikter kan innføre data som kompromisserer systemets funksjonalitet. Ved å innføre andre fraser i datasettet, kan det f.eks. godt tenkes at modellen vil slippe inn alle som sier den gitte frasen fremover. Dersom noen av lydklippene ikke kan valideres forkastes derfor fra serveren. Etter validering blir en respons fra serveren returnert, med informasjon om hvilke lydklipp som skal forkastes. AJAX fanger opp disse og oppdaterer nettsiden deretter, ved å fjerne de ikke-validerte klippene, men ved å beholde de validerte, slik at brukeren slipper å spille inn disse på nytt. I dette tilfellet gis tilbakemelding gjennom JavaScript og CSS, og ikke gjennom Flask sin innebygde mekanisme for tilbakemelding, ”flashing” (3.9.15.2). Dette pga. av at meldinger som ”flashes” må være knyttet til tråden som styrer flyten for nettleseren - som nevnt ikke er mulig ved bruk av AJAX. Det er også verdt å legge merke til at alle responser fra serveren i dette tilfellet blir innkapslet med et funksjonskall til ”jsonify” - dette er en innebygd funksjon i Flask som pakker responsen i JSON-format, slik at den skal være lesbar for JavaScript. Når et registreringsforsøk til slutt er godkjent og serveren har mottatt tre gyldige filer med frasen ”åpne dør”, blir brukerinfo lagt til i databasen, maskinlæringsmodellene trenes opp på nytt og brukeren blir varslet om at registreringen var suksessfull.

Registrering for nye administratorer blir gjennomført på en lignende måte, men den forskjell at administratorer trenger å levere epost og passord til serveren, mens nye brukere til talegjenkjenningssystemet på levere personinformasjon og lydklipp.

4.6.5 Innlogging og brukerhåndtering

Når en administrator har gjennomført registreringen, kan han/hun logge inn på den administrative nettsiden fra skjermen som er vist nedenfor (figur 4.49).

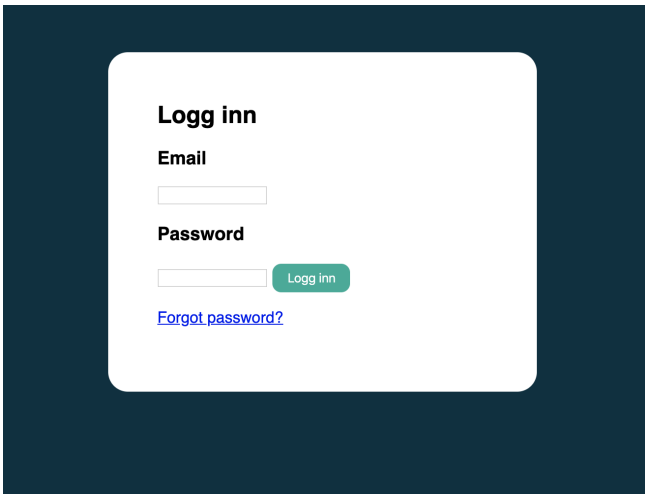
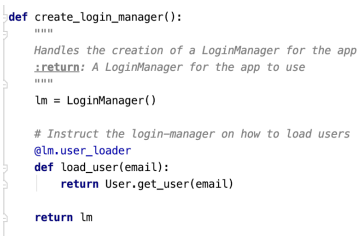


Figure 4.49: Side for innlogging

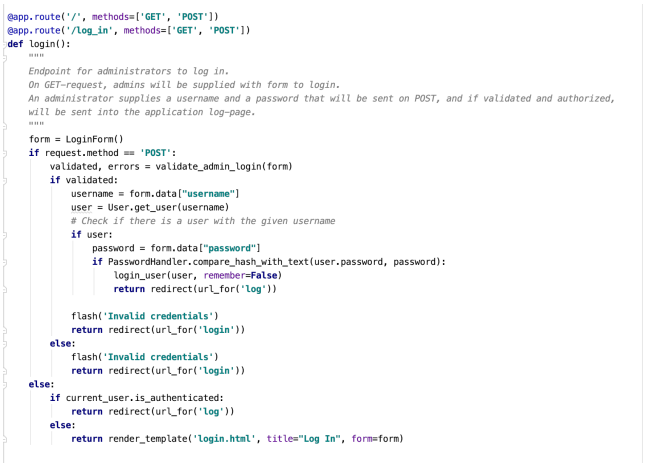
Innlogging og brukerhåndtering for applikasjonen baserer seg i hovedsak på Flask sin egen modul for autorisering og autentisering, flask-login (3.9.15.1). Flask-login tilbyr klassen "LoginManager", som gis til applikasjonobjektet, og som automatisk holder oversikt over innloggede brukere, session-cookies, og mer. For at dette skal være mulig, krever modulen at applikasjonen implementerer en klasse som representerer et brukerobjekt, i vårt tilfelle User.py (figur 4.33). User-klassen implementerer enkle metoder for å hente informasjon om brukeren (epost, aktiv-status, osv.). Deretter gis en "forklaring" til LogingManager på hvordan den unikt kan identifisere brukere, som vist under (figur 4.50).
Applikasjonen holder så automatisk oversikt over hvilken bruker som er logget inn, på tvers av forespørsler, gjennom variabelen "current_user". Dette gjør at vi f.eks. enkelt kan kalle "current_user.get_email()" for å hente eposten til innlogget bruker når vi skal hente brukerspesifikk data fra databasen. I tillegg slipper vi som utviklere å tenke på at applikasjonen i produksjonsmiljøet tjener flere ulike brukere av gangen med ulike tråder, fordi flask-login automatisk tar seg av dette.



```python
def create_login_manager():
    """
    Handles the creation of a LoginManager for the app
    :return: A LoginManager for the app to use
    """
    lm = LoginManager()

    # Instruct the login-manager on how to load users
    @lm.user_loader
    def load_user(email):
        return User.get_user(email)

    return lm
```

Figure 4.50: Initialisering av LoginManager med User-klassen

Kodeutdraget nedenfor viser hvordan applikasjonen håndterer en innlogging (figur 4.51).



```python
@app.route('/', methods=['GET', 'POST'])
@app.route('/log_in', methods=['GET', 'POST'])
def login():
    """
    Endpoint for administrators to log in.
    On GET-request, admins will be supplied with form to login.
    An administrator supplies a username and a password that will be sent on POST, and if validated and authorized,
    will be sent into the application log-page.
    """
    form = LoginForm()
    if request.method == 'POST':
        validated, errors = validate_admin_login(form)
        if validated:
            username = form.data["username"]
            user = User.get_user(username)
            # Check if there is a user with the given username
            if user:
                password = form.data["password"]
                if PasswordHandler.compare_hash_with_text(user.password, password):
                    login_user(user, remember=False)
                    return redirect(url_for('log'))

            flash('Invalid credentials')
            return redirect(url_for('login'))
        else:
            flash('Invalid credentials')
            return redirect(url_for('login'))
    else:
        if current_user.is_authenticated:
            return redirect(url_for('log'))
        else:
            return render_template('login.html', title="Log In", form=form)
```

Figure 4.51: Login-logikken i applikasjonen

Ved en HTTP-GET til konvoluttadressen '/login' vil applikasjonen kun returnere HTML-siden som tillater en bruker å logge inn. Kun når det gjøres en HTTP-POST forespørsel, vil applikasjonen prosessere data, gjøre interaksjon mot databasen og så videre, dette for å hjelpe å sikre applikasjonen mot angrep som CSRF (2.11.3). Ved å ikke gjøre logiske operasjoner gjennom HTTP-GET, følger vi OWASP sin anbefalte prosedyre for å sikre applikasjonen mot den enkleste formen for CSRF-angrep, der en link til en operasjon som gagner angriperen for eksempel inkluderes i form av et bilde [115]. Dette gjelder generelt for hele applikasjonen, og ikke bare for konvolutadressen '/login'. Når et innloggingsforsøk gjøres gjennom HTTP-POST, valideres først innsendt data (brukernavn og passord) for å sjekke at dataen er trygg å håndtere. Deretter sjekkes det om det finnes en bruker i databasen med assosiert brukernavn, og isåfall sammenlignes det krypterte passordet i databasen med klartekstpassordet som ble

sendt inn med innloggingen. Til dette brukes klassen PasswordHandler (figur 4.33), som wrapper bcrypt-modulen til flask (3.9.19). Ved godkjent innlogging instrueres LoginManager (3.9.15.1) om at brukeren skal logges inn, og brukeren blir sendt videre til siden for loggføring.

4.6.6 Sikkerhet

I tillegg til sikkerhetstiltakene som er nevnt ovenfor, med tanke på at det ikke gjøres noen server-prosessering ved annet enn HTTP-POST, har vi også gjennomført en rekke andre tiltak for å gjøre webapplikasjonen mer sikker. Under følger en kortfattet oppsummering av disse:

1. Administratorer får kun registrere seg med passord som følger OWASP sine retningslinjer for sikkerhet [27]. Det vil si at alle passord som brukes i forbindelse med applikasjonen skal være mellom 8-20 karakterer langt, må inkludere tall, samt små og store bokstaver. Passord har heller ikke lov til å inkludere noen uregelmessige spesialtegn. Restriksjonen opprettholdes ved å bruke regulære uttrykk (3.9.18). I tilegg bruker vi flask sin bcrypt-modul for å kryptere passord før lagring, og sørger for at alle passord krypteres med unike salter. Dette gjør passordene for applikasjonen spesielt sikre mot brute-force angrep (2.11.5).

2. Applikasjonen er kun tilgjengelig gjennom HTTPS (2.9.1), som sikrer at tyvlyttere ikke kan fange opp data (som f.eks. brukernavn og passord) som sendes til serveren som tjener applikasjonen.

3. Som man kan se fra figur 4.45, krever funksjonen for å sende ut invitasjonsepost til en bruker for talegjenkjennings-systemet, en "token". En slik token er en "url-safe", Base64-kryptert strengverdi som genereres ved hjelp av secrets-biblioteket (3.9.13) i Python. Vi har her fulgt den anbefalte metoden for å generere slike tokens, ved å bruke metoden "token_urlsafe" for å generere en token på 32 bytes [68] - dette skal være tilstrekkelig for å beskytte seg mot brute-force angrep (2.11.5). En generert token inkluderes i linken som tillater brukere å registrere seg, og er ment å sikre at kun inviterte brukere får tilgang til siden. Forøvrig brukes også tokens på samme måte for å regulere tilgangen til siden som lar nye administratorer registrere seg, og til siden som lar brukere resette passordet sitt.

4. All data som sendes inn til serveren valideres på serversiden gjennom modulen "FormValidator" (figur 4.33). Modulen bruker regulære uttryk (3.9.18) for å sikre at all innkommende data imøtekommer det forventede formatet og ikke inneholder spesialtegn som potensielt kan være skadelige. Dette er hovedsakelig med på å beskytte applikasjonen mot CSRF- (2.11.3), XSS- (2.11.4) og SQL-Injeksjon- (2.11.2) angrep, som alle har fellesnevneren at de prosesserer data som på en eller annen måte ikke burde vært validert. I alle tilfeller der brukeren sender inndata til applikasjonen brukes også JavaScript på klientsiden til å validere dataen, men dette er mer for

brukervennlighet, ettersom en proxy-server (2.11.1) kan brukes til å endre inndata etter at den har passert validering på klientsiden.

5. Alle spørringer som gjøres mot databasen gjøres gjennom egendefinerte SQL-prosedyrer for å hjelpe å sikre applikasjonen ytterligere mot angrep som SQL-injeksjoner (2.11.2. Input til en SQL-prosedyre kan nemlig ikke endre strukturuten på spørringer som gjøres, ved å introdusere nøkkelord som "FROM" og "WHERE". SQL for disse funksjonene, samt opprettelse av tabellene er lagt med som vedlegg 7.7.

4.6.7 Design

Oppdragsgiver ønsket at designet til nettsiden skulle være simpelt og ryddig, slik at det var lett å ha oversikten. Vi fremmet forslag om hvordan nettsidene skulle se ut gjennom wireframes, som er en forenkling av det endelige designet til nettstedet. Ved å bruke wireframes kunne vi diskutere med oppdragsgiver hvordan de ville at nettsiden skulle se ut. De endelige wireframene finnes i vedlegg 7.9.

Siden prosjektet ble laget for Avento, har fargepaletten brukt til å lage nettsiden vært de fargene som Avento bruker på deres hjemmeside. Det er fordi at bedriften vil at de ansatte skal føle seg "hjemme" både når de administrerer eller registrerer en stemme. Fargene vises i figur 4.52.

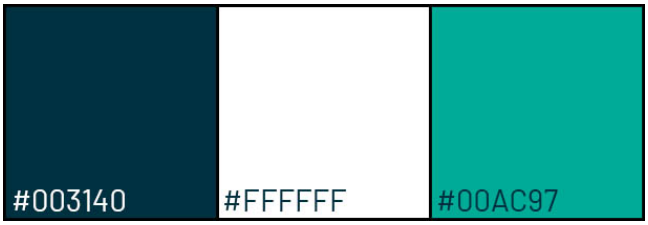


Figure 4.52: Fargepalett til administrasjonsapplikasjonen

Fonten vi har brukt, Barlow, er også den samme som på deres nettsted. Dette er fonten brukt for å nevne fargekode i figuren. Den blå og den hvite fargen brukes om hverandre til det meste innhold på siden, mens den mer grønne fargen indikerer knapper eller felt der handlinger blir gjort.

4.6.8 Distribusjon på server

For at ansatte i Avento skal få brukt den administrative nettsiden, har det vært nødvendig å distribuere nettsiden til en webserver slik at den kan nås via internett.

Oppdragsgiver har stilt med en egen webserver, som vi i felleskap har kommet frem til at skal kjøre Ubuntu 18.04 Server - da kommandolinjen her er et

miljø både oppdragsgiver og prosjektgruppen føler seg komfortable i.
I hovedsak har vi jobbet mot denne serveren ved å bruke SSH (2.9.6), ettersom det er dette oppdragsgiver har åpnet for. For å installere Apache2 (3.9.24), Mod_wsgi (3.9.25) og andre diverse tjenester som kreves for å kunne kjøre Flask-applikasjonen til den administrative nettsiden på serveren, har vi i hovedsak fulgt en veiledningartikkel fra DigitalOcean [105]. Vi har dog måtte gjøre en rekke unntak fra denne veiledningen, pga. spesifikke aspekter både ved vår applikasjon og ved vår server. En liste med mer detaljerte beskrivelser av disse unntakene er lagt ved som vedlegg til rapporten 7.7.
Da serveren ikke har noe grafisk grensenitt, har vi brukt kommandolinjeverktøyet til PostgreSQL (3.8.3), *psql*, for å jobbe mot databasen på serveren [106]. Mer spesifikt har vi brukt psql for å lage databasetabellene som trengs for applikasjonen, registrere databasefunksjoner, endre brukerinstillinger for Postgres, og lignende. For å installerede PostgreSQL og psql på serveren har vi også brukt en veiledningsartikkel fra DigitalOcean [107].

For at Flask-applikasjonen skal fungere som tiltenkt, har det også vært nødvendig å aktivere HTTPS (2.9.1) på serveren. For å få til dette har vi installert TLS-sertifikater og laget konfigurasjonsfiler for TLS mot applikasjonen ved hjelp av Certbot (3.9.26). Vi har her fulgt Certbot sine egne instrukser om hvordan dette skal gjøres for Apache på Ubuntu [108].

Et siste viktig aspekt ved serveroppkoblingen er at vi har måttet implementere funksjonalitet som er kritisk for at talegjenkjenningstjenesten skal fungere optimalt, men som ikke har vært implementert lokalt før oppkobling, da vi har brukt det lokale miljøet for testing av modeller. For det første må webserveren ta hensyn til utgåtte brukere for talegjenkjenninstjensten - dette er en funksjon som finnes da administratorer f.eks. skal ha muligheten til å registrere konsulenter som kun vil trenge tilgang til kontoret i en viss tidsperiode. For å sikre automatisk sletting av slike brukere, er det lagt inn en cron-job (3.9.27) på serveren som kjører python-scriptet under (figur 4.53) hver dag kl. 00:01. Scriptet sletter filene som tilhører utgåtte brukere fra serverens filsystem, og fjerner all data om brukerene fra databasen.

```
from CfgHandler import config_handler
from DatabaseManager import DatabaseManager
import os

dbm = DatabaseManager(host=config_handler.get_value('SETTINGS', 'DB_HOST'), database=config_handler.get_value('SETTINGS', 'DB_DATABASE'),
user=config_handler.get_value('SETTINGS', 'DB_USER'), password=config_handler.get_value('SETTINGS', 'DB_PASSWORD'))

expired_users = dbm.get_expired_users()
print(expired_users)
for user in expired_users:
        email = user[0]
        # Get path to users file on disc and remove
        reg_records = dbm.get_registration_recordings(email)
        for entry in reg_records:
                file_path = entry[1]
                os.remove(file_path)
        # Remove user from DB
        dbm.delete_user(email)
```

Figure 4.53: Python-script for å automatisk slette utgått brukere

Vi har også lagt inn en cron-job for å gjennomføre automatisk opptrening av maskinlæringsmodellen som styrer talegjenkjenningstjenesten dersom det enten er lagt til nye brukere for systemet, eller dersom noen er slettet. Denne

cron-jobben kjører enkelt nok et bash-script som restarter applikasjonen, da applikasjonen er satt opp slik at den da automatisk vil trene modellen med den data som er relevant. Den data som er relevant er all data som ligger i filsystemet, da lydklipp blir fjernet fra systemet om tilhørende bruker slettes, og lydklipp legges til automatisk i filsystemet ved registrering.

Det finnes servermiljøer som gjør at vi kan trene maskinlæringsmodeller på serveren "on the fly", f.eks. TensorFlow Serving [116]. Vi har dog, sammen med oppdragsiver, kommet frem til at det er tryggere (og enklere) å restarte systemet hver gang, ettersom modellen brukes i Flask-applikasjonen, som vanligvis vil kjøres over flere tråder og prosesser samtidig. Det er sannsynligvis heller ingen fare for at noen vil bruke systemet kl 1 på natten, så vi anser dette som en gunstig løsning.

Bildet under (figur 4.54) viser crontab med en oversikt over de to cron-jobbene som vi har lagt inn på serveren:

```
SHELL=/bin/sh
CRON_TZ=Norway
0 00 * * 1-7 sudo python /var/www/AventoSR/AventoSR/delete_expired.py
0 01 * * 1-7 sudo /var/www/AventoSR/./retrain_models.bash
```

Figure 4.54: Crontab med instrukser om hvilke cron-jobs som skal kjøre, og når.

4.7 Side for innsamling av lydopptak

Da vi laget et spørreskjema for å samle inn lydopptak (3.12.3) opprettet vi en nettapplikasjon for dette tilfellet. Denne applikasjonen er laget med HTML, CSS og JavaScript, som er koblet opp mot en backend skrevet i Flask. Nettsiden har ett dokument for selve spørreskjemaet, og ett dokument som fungerer som en konfirmasjon på at undersøkelsen er ferdig. Strukturen til hvordan systemet er satt opp blir beskrevet i 4.7.3. Dokumentet for spørreskjemaet er koblet opp mot to separate JavaScript-filer som gir støtte for interaksjon med siden. I tillegg er en CSS-fil definert med forskjellige stiler for å gi et bedre utseende til applikasjonen.

4.7.1 Distribusjon

For å tjene nettapplikasjonen har vi benyttet oss av en virtuell datamaskin. Denne datamaskinen er generert av VMDeploy, som er en tjeneste levert av universitetet. Datamaskinen er levert med Ubuntu 18.04 Server. Siden kan nås gjennom lenken *https://www.fredrifw-bachelor.uials.no*. Siden denne applikasjonen også er en Flask-applikasjon, er denne applikasjonen mer eller mindre distribuert på samme måte som det administrative systemet (4.6.8).

4.7.2 Endelig design

Det viktigste kriteriet for denne applikasjonen var at den skulle kunne brukes av hvem som helst. Dette for å ha en bredere utvalg av opptak, slik at ikke kun en målgruppe sendte inn lyd. Siden skulle derfor ha et utseende som skulle være lett å skjønne uavhengig av teknisk kompetanse. Derfor gikk vi for et design uten mange farger, og knapper som skiller seg fra det andre. Hver del på nettsiden er i sin egen boks for å skille hvilke funksjoner som brukes hvor

4.7.2.1 Startside

På startsiden får brukeren en introduksjon til spørreskjemaet og hva brukeren kan forvente når de går gjennom. Denne er vist i figur 4.55.

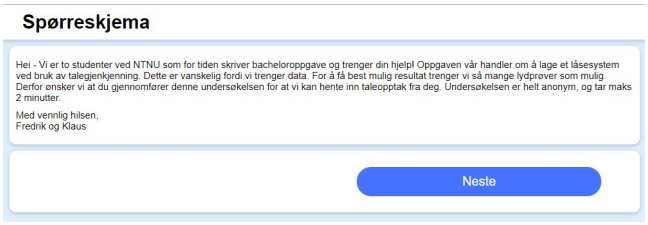


Figure 4.55: Startside

4.7.2.2 Utfylling av personlig informasjon

Det er fordelaktig å vite kjønn og alder på brukeren for å sikre at man har et bredt nok datasett. Derfor trengte vi både kjønn og alder på brukeren. Knappen for å gå videre på nettsiden blir aktivert når begge feltene er fylt ut. Denne siden er vist i figur 4.56.

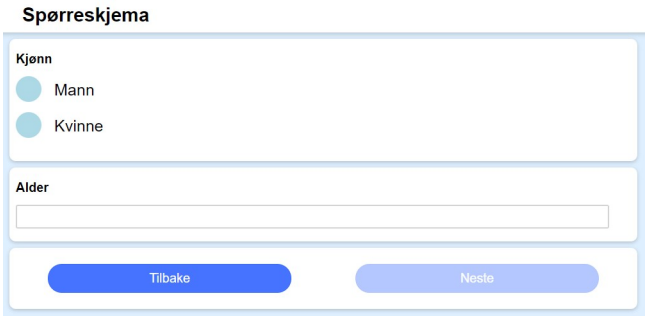


Figure 4.56: Side for personlig informasjon

4.7.2.3 Innspilling av lyd

På denne siden får brukeren mulighet til å ta opp lydklipp av seg selv. Brukeren får informasjon om hva de skal si og hvor mange ganger de skal si det. Hvordan

siden ser ut i starten er i figur 4.57.

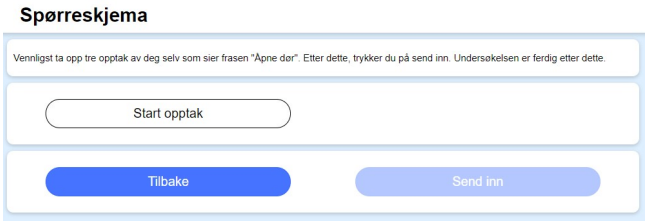


Figure 4.57: Opptaksside

Etter at brukeren har tatt opp stemmer, har de mulighet til å høre på og slette ethvert opptak. Etter tre opptak åpnes det for at brukeren kan sende inn spørreskjemaet. Da vil siden se ut som i figur 4.58

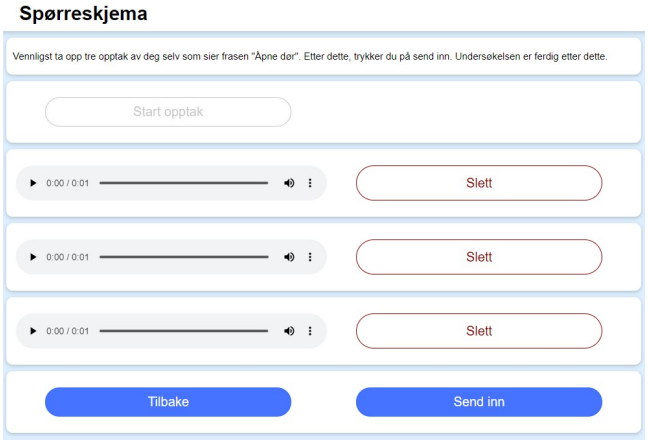


Figure 4.58: Opptakside når tale er spilt inn

4.7.3 Systemarkitektur

Systemet for applikasjonen er bygget opp av:

- **init** - Dette er selve flask-applikasjonen. Her bestemmes det hva som skjer til når brukeren sender en request til nettsiden. Ved GET-requests (2.9.1.1) vil brukeren få de forespurte dokumentene fra serveren, før så å vise dette i sitt nettleservindu.
- **templates** - Templates er en mappe der alle HTML-filene ligger.
- **Navigator** - Navigator er JavaScript-filen som har orden på navigasjon på siden. Den har funksjoner som utføres når knapper på nettsiden blir trykket på. Navigasjonselementene i HTML-dokumentet er definert med

en funksjon fra denne i parameteren *onclick*. Funksjonene gjemmer og viser de forskjellige sidene ettersom hvor brukeren befinner seg på siden.

- **Recorder** - Recorder er JavaScript-filen legger til rette for å ta lydopptak av brukeren. Lydopptaket bruker en en Recorder.js. Ettersom lydopptak blir spilt inn via denne klassen, vil lydfilene som en bytestreng i filen. Filen har også funksjoner som holder orden på om noen av filene blir slettet, slik at man kun har de tre lydopptakene brukeren vil ha. Etter tre lydopptak er tatt opp, åpnes det for at brukeren kan sende inn skjemaet.

5 Drøfting

I dette kapittelet ser vi på ulike perspektiver av arbeidet vi har gjort i prosjektet. I begynnelsen går vi igjennom hvilke tanker vi har rundt resultatene og utvikling av de forskjellige komponentene i systemet, før vi drøfter selve prosjektgjennomføringen. Spesielt viktig å bemerke seg er at det har oppstått komplikasjoner som følge av utbruddet av Covid-19, som igjen har ført til at vi har måttet inngå kompromisser i forhold til opprinnelig plan.

5.1 Talegjenkjenning

I denne seksjonen diskuterer vi ulike aspekter ved talegjenkjenningsdelen til oppgaven. Vi beskriver først hvilke erfaringer vi har hatt ved arbeid med systemet, og ser så på hvilke bemerkninger vi har i forhold til datasettet brukt. Deretter vil vi diskutere om løsningen kan brukes i praksis, spesielt med tanke på sikkerhet.

5.1.1 Arbeid med Tensorflow og Keras

I prosjektets startfase hadde vi begrenset med kunnskap innenfor maskinlæring og nevrale nettverk. Da det fantes begrensede rammeverk for dette allerede, og vi helst ville ha et rammeverk vi kunne tilpasse til vårt eget bruk, krevde prosjektet en del studering i forkant for å finne ut hvilke teknikker som kunne brukes og hvordan vi skulle løse problemet. Heldigvis fantes det rammeverk som gjorde det mulig å bygge nevrale nettverk uten å ha spesialisert kunnskap om emnet, slik som Tensorflow og Keras, slik at vi relativt fort fikk kommet i gang med testing. På denne måten har vi sluppet å sette oss veldig mye inn i selve matematikken bak modellene, og heller fokusere på hvordan modellene fungerer i praksis. Studering omhandlet mer hva de ulike lagene i modellene gjorde for å oppnå ønsket resultat og hvordan vi kan prosessere datasettene våre for å gi modellene bedre data, i tillegg til mye annen generel og overordnet kunnskap og maskinlæring.

Da vi begynte prosjektet prøvde vi oss frem på hvordan disse rammeverkene fungerte ved å følge guider på nettet. Tensorflow selv gir ut guider på hvordan man bruker rammeverket ut ifra hvor erfaren du er[104], så på den måten kunne vi jobbe oss oppover til mere komplekse problem. Da fikk vi en forståelse på hvordan modeller var satt opp, og hva som gjorde at modellene klarte å bruke verdier for å klassifisere data.

5.1.2 Testsystem

Miljøet for testing og utredning av modellene var definert i 4.3.1. Her har vi jobbet på hver vår enhet for å komme oss frem til det beste resultatet. Det skal sies at dette er vanskelig når begge gruppemedlemmer prøver ut forskjellige ting og utfører forskjellige teknikker. Dersom en gjør en endring for å få til

noe spesifikt vil dette kanskje føre til at den andre får ubrukelig kode ettersom. Utover prosjektet ble testing gjort lokalt hos medlemmene for å forhindre disse problemene. Vi hadde en klar mappestruktur i starten for testmiljøet, som hjalp oss da dette problemet oppsto. Ved et eventuelt neste arbeid med utprøvelse av diverse teknikker, bør det diskuteres hvilke hovedklasser som er nødvendige for prosjektet, og hvilke roller klassene skal ha. Det fører til at man enklere kan referere til spesifikke underledd når man gjennomgår resultatene. Ved bruk av versjonskontroll bør alle andre filer brukt legges i en ekstern mappe, så man enkelt kan definere at denne mappen ikke skal inkluderes i en konfigurasjonsfil, slik som i *.gitignore* for Git.

5.1.3 Utvikling av modeller

Siden vi begge var ferske med maskinlæring og ikke hatt noe formell erfaring med det, har prosessen ved utvikling av modeller vært veldig tidkrevende. Beslutninger om hva man skulle prøve ble tatt ut ifra hvilke teorier vi hadde lest om, samt dialogene med veiledere. Det har gjort at vi har tilegnet oss kunnskap basert på hvilke subjektive erfaringer vi har gjort i prosjektet. En bemerkning vi har gjort oss er at maskinlæring handler veldig mye om hvilket datasett man bruker i modellene, kontra hvordan modellene har blitt satt opp. Som følge av dette gjør det at mye tid går til å preparere ulike datasett, slik at man kan trene en modell på dette. Selve treningsprosessen er også tidkrevende da vi trener modeller baser på alle de ulike datasettene. Som læring har vi fått en større forståelse om hvordan man skal gå frem når man lager datasett, slik at man slipper å teste ut modeller med mange underskritt. Siden vi ikke hadde mye kunnskap om nevrale nettverk selv, gikk det også tid på finne ut hvordan nettverkene fungerte, og hvordan vi skulle sette de opp for å få et godt resultat. Da vi ut over i prosjektet fikk et grunnlag på hvordan man setter opp en modell, satte vi opp utgangspunktsmodeller (4.3.1.2) for de ulike nettverkene. Da kunne vi begynne å fokusere på selve dataen, slik at vi fikk en pekepinn på hvordan de ulike datasett påvirket utfallet deres.

Vi har fått til å trene modeller og klassifisere data med modellene slik at lyd som ligger i selve datasettet er korrekt. Problemet ligger i at det har vært vanskelig å koble dette opp mot ny, ukjent data for modellene. Problemstillingen her er at vi vil at personer som er ukjente for modellen ikke skal kunne komme inn. Det er derfor vanskelig å definere hva en god modell er i disse tilfellene, men å holde av en delmengde filer fra treningen var et godt tiltak. På grunn av denne problemstillingen er det derfor viktig at man har en klasse (Predictor i vårt tilfelle) for å sjekke modellen opp mot slik data, og se på statistikken generert av denne klassen opp mot de andre modellene. Dersom man holder av rundt 10 prosent av dataen til dette formål, kan man få en pekepinn på på om ukjente personer blir utelatt fra systemet eller ikke.

5.1.4 Datasett

Ut ifra våre resultater ved modellene er det en del problemstillinger. Vårt siste resultat (4.3.7) beskrev et scenario der enkelte individer som ikke var i det opprinnelige datasettet modellen ble trent på, i noen tilfeller ville kunne bli klassifisert som verifisert. Det har vist seg vanskelig å konstruere modellen slik at dette ikke skal være et faremoment. En hovedfaktor er at vi ikke har fått et stort nok datasett gjennom spørreskjemaet (4.7) til å representere en hel befolkning. Det gjør at siden det finnes felles faktorer i fingeravtrykket til ulike talere, vil noen individer bli klassifisert som en av de verifiserte talerne, da de ligner mer på dem enn de uverifiserte talerne.

Spørsmålet er da hvordan man kan forbedre datasettet. Vi har forsøkt oss på metoder som å generere flere augumenterte lydfiler for personer som er verifisert, slik at modellene lærer seg deres karakteristikker bedre enn de uverifiserte. Dette har vist seg å klare å klare å gi modellene prediksjoner med høyere sannsynlighet. Selv om vi har flere filer, løser det fortsatt ikke problemet av å representere en hel befolkning, som gjør at vi fortsatt sitter igjen med det samme problemet. Derfor har vi trengt et større datasett, noe som ikke har latt seg gjøre. Dette blir snakket mer om i neste seksjon (5.1.5).

5.1.5 Innsamling av data

Å samle inn data via internett har vært krevende. Både gruppemedlemmer og veiledere har delt lenken til spørreskjemaet vårt på det sosiale mediet Facebook og lagt ut i den felles informasjonskanalen for universitetet, Innsida. Rundt halvparten av alle lydfiler som opprinnelig lå i datasettet under testing var samlet inn via direkte kontakt med bekjente. Konklusjonen her er at selv om de fleste i kontaktnettverket både til medlemmene og veileder har hatt hjemmekontor, er det vanskelig å få dem til å trykke på en lenke og samle inn data. En grunn for det kan være at når folk arbeider og driver møtevirksomhet gjennom en datamaskin hele dagen, vil muligens digitale ressurser bli ignorert utenfor arbeidstid. Likevel skal vi ikke se bort ifra muligheten for at folk generelt ikke pleier å trykke på lenker.

En metode som viste seg å være god for å samle inn data var å sende ut skjemaet gjennom appen Nabohjelp laget av OBOS. Gjennom denne appen kan man spørre personer i en gitt radius om hjelp til hverdagslige ting, som for eksempel "Jeg skal ha familien over på middag. Lurte på om noen har et par stoler til overs?". Vi sendte ut spørreskjemaet gjennom appen, og fikk totalt sett over en uke, lydklipp fra 31 nye mennesker, som bidro i større grad når vi skulle trene modellene, siden vi fikk flere prøver av uverifiserte mennesker. Fordelen her kontra ved publisering gjennom sosiale medier osv. er at applikasjonen har kun hensikt mot å hjelpe hverandre, slik at mottakere sannsynligvis har vært mer åpne for å trykke på linken.

Dersom samfunnet var åpent hadde fysiske metoder vært mer fordelaktig. Det var tenkt at vi kunne stå utenfor butikker, på campus og andre steder og spurt om hjelp. Her kunne vi tilbudt kaffe eller lignende som takk for at de svarte på spørreundersøkelsen. I dette tilfellet måtte vi velge steder der vi kunne få et varierende datasett som tar hensyn til mennesker med forskjellig kjønn og alder. Herunder også ta hensyn til dialekter, og tatt opptak av folk både ved studiested, hjemsted og andre miljøer.

5.1.6 Ferdig system for talegjenkjenning

Da det ferdige systemet for talegjenkjenning skulle fremstilles måtte vi finne ut av hvilke faktorer fra testsystemet vi skulle trekke fram. Hva som kom med i dette systemet var basert på hvilke resultater vi fikk fra 4.3.7. Her kunne vi igjen jobbe sammen med versjonskontroll, siden det var inneforstått hvilke komponenter i systemet som måtte være der. For å i større grad følge objektorienterte prinsipper, ble filer som var laget som script heller sammensveiset til metoder i klasser. En grunn for at dette ikke var gjort tidligere, var poenget om at det var ulike delledd i prosessen, og disse filene var ofte endret på for forskjellige utprøvelser. Vi implementerte derfor de metoder som var hensiktsmessige for problemet inn i klassen (4.4), i motsetning til å lagre filene lokalt først. Da vi flyttet over filer fikk klasser nye navn for å skille mellom testoppsettet og det ferdige systemet. Siden annen funksjonalitet var lagt til i klassene var også det en fordel for å bedre vite ansvaret til klassen.

5.1.7 Sikkerhet ved bruk systemet

I oppgaven har vi sett på ulike sikkerhetsperspektiver som er kritiske i forhold til systemet. Det første synspunktet omhandler administrasjonssiden. Her har vi distribuert serveren med en rekke krypteringer i form av TLS ved kommunikasjon med serveren, og kryptering av kritisk data i systemet slik som passord, mm. (4.6.6). Det andre synspunktet omhandler mer selve bruk av låsesystemet.

For at en biometrisk metode skal kunne brukes i et system, må den ha en falsk positiv-rate på 0, siden det ikke skal være mulig å komme inn. Når vi har laget talegjenkjenningsmodellene har vi tatt høyde for dette, og validert modellene opp mot ukjent data for systemet. Vi har konkludert med at vårt system ikke er sikkert nok for dette, selv om modellene for det meste fungerer, vil det fortsatt være avvik i form at noen personer med høy sannsynlighet vil bli klassifisert som validert, selv om ikke modellene har trent opp på denne talen. Med dette er det fordelaktig å bruke en annen form for verifiseringsmetode i tillegg til talegjenkjenning. For eksempel kunne man koblet systemet mot et kamera og brukt metoder som ansiktsgjenkjenning som et ekstra sikkerhetsledd. Likevel ser vi på løsningen som brukbar til oppdragsgivers behov. På grunn av begrensing fra låseselskapet deres, kan vi kun overstyre deres system innenfor åpningstidene til bedriften (08.00-16.00). Da vil det være ansatte tilstede i lokalet, som gjør at de heller kunne bruke tale til å komme inn på lokalet kontra en RFID-brikke.

Med tanke på modellene er det også viktig at man ikke kan replisere en stemme ved å spille av et opptak fra personene. Fra testene vi har kjørt i 4.3.8, har vi kommet frem til at dette er en trussel. Det betyr at bedriften må ta hensyn til det når de velger om løsningen er brukbar eller ikke.

En funksjonalitet vi har måttet ofre til fordel for sikkerhet, er klassifikasjon av enkeltmennesker, slik at vi kan se hvem som har kommet inn i systemet eller ikke. Som følge av det har vi brukt binær klassifikasjon, fordi vi da i større grad kan skille mellom mennesker som skal være validert i systemet eller ikke. Det har gjort at administrasjonssiden blitt mer minimal i forhold til slik den allerede var utviklet, i form av at man kun kan se om forsøk er godkjent eller ikke i motsetning til forsøk per bruker. Det gjør at man heller ikke heller får mulighet til å se forsøksstatistikk for brukerne heller.

Oppdragsiver og prosjektgruppen har dog vært klar hele veien på at dette er en oppgave som i hovedsak er valgt ut ifra et læringsperspektiv. Selv om det endelige produktet har sine utfordringer med tanke på sikkerhet, er det dog et godt utgangspunkt for videre utvikling. Samtidig gir innsikter fra prosjektet en god pekepinn for hva som funger og hva som ikke fungerer innenfor talegjenkjenning og biometriske metoder generelt.

5.2 Side for taleopptak

Da vi lagde siden for taleopptak (4.7) satte vi resten av prosjektet på pause, slik at vi fikk utviklet siden så raskt som mulig, som gjorde at vi fort kunne få inn data og bruke for opptrening av maskinlæringsmodellene. For å gjøre det enklest mulig prøvde vi å bruke standard JavaScript for å benytte oss av funksjonalitet. Medlemmene har erfaring med å lage nettsider, men grunnet komplikasjoner ved den innebygde taleopptaksfunksjonen, slik det er beskrevet i 4.6.4, tok det dessverre lenger tid enn forventet. Siden denne applikasjonen ikke var en del av det opprinnelige prosjektet tok det litt av tiden til andre prioriteringer. Da dette skjedde hadde vi jobbet godt og var etter den opprinnelige planen, så det ga ikke et spesielt negativt utfall.

5.3 Raspberry Pi

I prosjektet har vi brukt en Raspberry Pi som en hardware-enhet til å ta opp lyd og snakke med serveren. Ved bruk av en Raspberry Pi kommer det begrensninger man må være observante på. Siden enheten bruker en ARMv7-prosessor, er den ikke nødvendigvis kompatibel med alle python-moduler. TensorFlow er et av rammeverkene som ikke støttes på Raspberry Pi. Det finnes underrammeverk for å kompensere for dette, slik som TensorFlow Lite, men man kan dog kun bruke ferdigtrente modeller som så er importert til enheten. Derfor vi ikke kunne trene modeller direkte på enheten, og måtte istedenfor flytte dette over til serveren til administrasjonssiden. Med dette oppstår det litt ekstra forsink-

ing ved validering av en stemme som skal inn i lokalet. Likevel ser vi på dette som en fordel, siden vi nå oppbevarer all sensitiv informasjon som navn og lyddata kun på serveren, og ikke på Raspberry Pi. Det gjør at systemet blir sikrere.

Det er også i motsetning rammeverk som ikke kan kjøres på datamaskiner, men som fungerer på en Raspberry Pi. For å bruke output-portene til en Raspberry Pi benytter vi oss av pakken RPi.GPIO (3.9.23), som ikke lar seg installere på en datamaskin. Man må derfor ekskludere denne koden dersom man tester systemet på en datamaskin, slik at man ikke får en kompileringsfeil når maskinen prøver å laste inn slike filer.

5.4 Administrasjonsside

Arbeidet med en administrative nettsiden har gjennomgående fungert bra. Nettsiden har blitt utviklet gradvis ved siden av maskinlæringsmodellene. Nøyaktig design og funksjonalitet for nettsiden har blitt diskutert og bestemt stykkevis i samarbeid med oppdragsgiver. Dette som en følge av at nøyaktig funksjonalitet for nettsiden ville ha variert dersom vi f.eks. benyttet oss av Azure sitt API (2.1) fremfor maskinlæringsmodeller som vi har laget selv. F.eks. ville en løsning som brukte Azure sitt API ikke hatt mulighet for å vise den samme informasjonen i brukerpanelet - enkeltbrukere for Azure registreres ikke via epost, men ID, enkeltbrukere for Azure ville trengt bilde for identifisering, osv. Nettsiden er dog forsøkt laget meget generell, slik at det er enkelt å utvide brukerpanelet og panelet for loggføring med mer informasjon dersom man skulle ønske dette. En enkel utvidelse vil kunne gjøres ved å utvide tabellen som holder på brukerrelatert data i databasen og lage ekstra listeoppføring i HTML-siden for brukerpanelet, der data inkluderes gjennom Jinja 3.9.16. Med tanke på at nøyaktig utseene og funksjonalitet for den administrative nettsiden i perioder har vært noe uklar, har det dog vært viktig for oss å sikre at vi i alle fall har sett for oss en nogenlunde lik løsning som det oppdragsgiver har tenkt. Vi har derfor kontinuerlig brukt enkle skisser som har vært fremlagt for oppdragsgiver, slik at det har vært enkelt for oss å visualisere hva vi har tenkt, og enkelt for brukeren å komme med konkrete tilbakemeldinger. Et eksempel på slike skisser finnes som wireframes i vedlegg 7.9.
Som kommer frem av skissene, kan man se at det har vært tenkt å legge ved opptaket for de ulike forsøkene på å låse seg inn, i loggen. Dette ble til slutt forkastet da oppdragsgiver hadde bekymringer for dette relatert til personvern. Det var dog greit å lagre klippene som ble brukt til registrering, da disse hadde et spesifikt bruksområde (opptrening av maskinlæringsmodeller).

5.5 Distribusjon av systemet

Som diskutert i 4.6.8, har det endelige talegjenkjenningssyetemt og den administrative nettsiden blitt lastet opp på en server som tilhører oppdragsgiver. Dette har vært en krevende del av prosjektet, da vi her, som i likhet ved maskinlæring, har måttet plukke opp mye ny kunnskap underveis. Det har til tider

vært krevende å forholde seg til et ikke-grafisk brukergrensesnitt, spesielt i de tilfellene der det har oppstått feil. Obskure problemer har bla. inkludert:

- Installasjon av Psycopg2 (3.9.17) måtte endres til en spesiell versjon da den opprinnelige versjonen kom med sitt eget SSL-adapter, som kræsjet med det som allerede fantes på serveren. Dette ga kun en "coredump" (innhold i minnet) fra når adapterene kræsjet, slik at vi selv måtte lese denne med GDB [120] for å finne ut av hva problemet var.
- Endring av en rekke konfigurasjonsfiler, filtillatelser og lignende for å få applikasjonen til å kjøre korrekt (7.7).
- Spesialinstallering av en rekke bibliotek (TensorFlow, Librosa, Werkzeug, etc..) som av ulike grunner ikke kjører som tenkt på serveren [121].

Selv om disse utfordringene til tider har vært veldig frustrerende, har de samtidig gjort at vi sittet igjen med en hel del kunnskap om distribusjon av applikasjoner på webservere. Noe vi begge kommer til å ta med oss videre.

5.6 Oppkobling av system

På grunn av pandemien var det ikke mulig å installere systemet hos oppdragsgiver. Det har gjort at vi har måttet improvisere. I prosjektet har vi brukt en Raspberry Pi, på grunn av at oppdragsgiver hadde denne enheten allerede. Derfor vil det være enkelt for dem å sette opp systemet, ved å kun koble til ethernet (internett), mikrofon og lysdioder, i tillegg til å endre SD-kort med operativsystem og lagring til vårt. Vi har ikke fått testet oppkobling til selve låsesystemet, men kan implementeres i metoden *open_door()* i IOHandler etter hvordan det er koblet opp. Prosessen for å koble opp Raspberry Pi er definert i filen *README.md* i prosjektmappen RPiSystem.

På grunn av situasjonen har vi simulert inngangen til kontorarealet hjemme hos en av medlemmene, ved å sette enheten ved siden av en dør i en gang. Siden vi ikke har implementert løsningen opp mot låsesystemet, kan vi heller se på lyset til den grønne diode for å indikere at døren åpnes, da denne koden kjøres rett etter åpne dør.

Det var et krav fra oppdragsgiver om at løsningen skal kunne brukes til andre bruksområder enn å åpne en dør. Dette kan implementeres ved å erstatte *open_door*-metoden med en annen metode, for eksempel for å brygge en kaffekopp. Ettersom det var vanskelig å få til talegjenkjenning uavhengig av hvilken frase som blir sagt, vil modellen fortsatt trigge på frasen "Åpne dør", som muligens ikke er relevant for et kaffesystem.

Ut over det er serveren den samme som vi har utviklet på. Det har gjort at det eneste oppdragsgiver selv må innføre for å bruke systemet er signalet til låsesystemet, da vi allerede har implementert kommunikasjonen mellom innspillingsenhet og server.

5.7 Bruk av systemet

På grunn av at vi ikke har hatt tilgang til kontorarealene til Avento, har vi heller ikke hatt mulighet for å teste systemet framfor deres ansatte. At systemet har fungert under testing for gruppemedlemmene er en ting, men vi vil se hvordan resultatene blir når flere ansatte har registrert seg, da det vil gi en bedre indikasjon på hvordan systemet er i praksis. Da kunne vi tatt en bedre vurdering på om dette er en god løsning, siden vi kunne fått tilbakemeldinger på hvordan systemet er å bruke. Det er mulig det er tilfeller der lydklipp ikke er gode nok, at lydklippene er for like uverifiserte lydfiler osv. I tillegg er det mulig at med mer data, at modellen vil være enten sikrere eller mer usikker på om en person er klassifisert som verifisert. At en en klassifikasjon er usikker kan for eksempel føre til at det kanskje krever flere forsøk å komme inn.

5.8 Kommunikasjon

Her beskriver vi hvordan kommunikasjon og gjennomføring med alle parter har gått gjennom prosessen. Vi drøfter tanker om prosessen etter at all kommunikasjon flyttet seg fra fysiske til digitale platformer under Covid-19-pandemien.

5.8.1 Innad i prosjektgruppen

Helt til utbruddet tok sted overholdt prosjektgruppen tidsplan beskrevet i forprosjektrapport om å møte på lab klokken 09.15 hver dag. Det gjorde at vi fikk en ordentlig struktur på hverdagen, som gjorde at dagene ble mer produktive. Som følge av dette var det felles lunsj klokken 12.00, før vi avsluttet rundt 16.00. På morgenen holdt vi stand-up-møte for å forsikre at begge medlemmer var inneforstått med hva den andre holdt på med.

Denne tidsplanen endret seg litt etter utbruddet. Begge prosjektmedlemmer har hatt mulighet til å bo hjemme hos foreldre i perioden, og tok bruk av dette tilbudet. Vi ble enige om å flytte start av prosjektet til klokken 10.00, så vi fikk kommet til rette innen den tid. På dette tidspunktet fulgte vi Scrum-prosessen og holdt stand-up-møte digitalt gjennom videochatfunksjonen på Facebook Messenger.

At prosessen har flyttet seg digitalt har både medført fordeler og ulemper. Ulempene har vært å miste den nære kontakten med hverandre når vi jobber med stoffet, så man like lett ikke får hjulpet hverandre dersom det oppstår et problem. Vi har likevel stilt problemstillinger utenfor møtene på Messenger, men fungerer ikke like optimalt når man må se på kode. Fordeler derimot har vært at medlemmene har jobbet mer effektivt, på grunn av færre forstyrrelser ved et hjemmekontor.

Gruppen har bestått av kun to medlemmer og ikke tre som anbefalt, som har gjort at arbeidsmengden til hvert medlem har økt med 50 prosent. Da det er en

ekstra belastning, har det likevel vært positivt, slik at vi lettere kan holde orden på hva hverandre driver med. Gruppemedlemmene har tidligere jobbet sammen på utviklingsprosjekt i annet emne ved universitetet, og allerede blitt kjent med hverandres arbeidsbaner, som har vært en positiv faktor innad i gruppen.

5.8.2 Oppdragsgiver

Med oppdragsgiver har kommunikasjonen også foregått utmerket. På det første møtet gjennomgikk vi problemstillingen med en større arbeidsgruppe i Avento, før vi ble tildelt en spesifikk kontaktperson. Fordelen med å ha et større panel i begynnelsen var at vi fikk ulike synspunkt på oppgaven, slik at vi fikk et mer overordnet inntrykk fra bedriften. Med kontaktperson har vi videre hatt god kontakt, og hatt møter der det er nødvendig. Det har ikke gått mer enn to sprinter mellom hvert møte, og har derfor kontinuerlig opptatert kontaktperson på prosjektet og diskutert mer i detalj hvordan bedriften vil at systemet skal være.

Utover møtene har vi hatt kontakt i Microsoft Teams der det har vært nødvendig. Dette har gjerne vært oppfølgingsspørsmål knyttet til kommunikasjonen i møtene. Under perioden uten fysisk kontakt har vi også tatt i bruk videotjenesten i Teams for å holde møter. Å kommunisere kun over nett har gått fint i perioden, men likevel har vi hatt nytte av fysiske møter i starten, så vi har bedre kunnet diskutere systemet ved hjelp av tegninger og figurer. I tillegg har det vært greit å møte personer man jobber med for å komme på et bedre sosialt nivå. Av disse grunnene bør det strebes å ha fysiske møter, selv om en eventuell kommunikasjon skal foregå over internett under prosjektets gang.

Av å ha en oppdragsgiver har gjort at vi har kunnet jobbe opp mot krav, slik at vi kan fylle et behov. Det har gjort at oppgaven har blitt snevret inn, og gjenspeiler slik en kunde i arbeidslivet ville oppført seg. At oppdragsgiver har utnevnt en spesifikk kontaktperson har gjort at denne personen har oppført seg mer som en *produkteier* innenfor Scrum-metodikken, siden han har hatt bedre innsyn i prosjektet, og stiller krav på vegne av bedriften.

5.9 Gjennomføring

I denne seksjonen har vi tatt for oss punkter under selve gjennomføringen og hvordan det har gått i sin helhet. Dette omhandler også bruk av verktøy og teknikker som har hatt innvirkning på hvordan vi har jobbet sammen i prosjektet.

5.9.1 Utviklingsmetodikk

Vi har i prosjektet brukt metodikken Scrum under gjennomføring. Vi har overholdt sprintlengden på 2 uker, og hatt et stand-up-møte hver arbeidsdag. Under endt sprint har vi gått over hvordan prosessene har vært og hatt påfølgende

møter med veiledere. Å ha en sprint på to uker har også gjort at vi kan gjøre oss ferdig med de inneværende oppgavene før et møte med veiledere, slik at vi kan stille konkrete spørsmål ovenfor dem. Rapportene fra sprintene finnes i vedlegg 7.6.

Det har vært en fordel å gjennomføre prosjektet på denne måten, siden vi kontinuerlig kan vurdere hvor langt vi har kommet og for å sette prioriteringer for hva som er viktig. En viktig del av prosjektet har vært å utrede modeller for talegjenkjenning, som vi antok ikke hadde et definert sluttpunkt, da det er vanskelig å finne en optimal modell. Derfor har vi kunnet tatt en vurdering på å begynne å fokusere mer på systemet i seg selv ut ifra hvor langt vi hadde kommet med modellene.

5.9.1.1 Bruk av backlog / Jira

I en backlog har vi delt opp alle oppgaver i prosjektet til å representere de ulike komponentene i systemet. Denne backloggen har vi lagret i en webapplikasjon kalt Jira, der man enkelt kan legge til nye oppgaver i backloggen. Når vi har planlagt sprinter har vi flyttet over oppgavene fra backloggen til sprinten før vi har startet den. Å ha en backlog har gitt en oversikt over hva som skal gjøres i prosjektet, som har ført til at vi kan bedre planlegge prosjektet.

Hver oppgave har vært lagt under hvert av delsystemene laget i oppgaven, så man enkelt kan skille mellom hvilke oppgaver som skal fullføres for å utforme hvert av delsystem. Oppgavene har fått spesifisert antatt tidsbruk, som vi har brukt som en pekepinn når vi har planlegget sprintene.

5.9.2 Tidsplanlegging

I forprosjektrapporten gjorde vi rede for antatt tidsbruk i prosjektet. Denne tidsbruken ble så gjort om til et Gantt-skjema ut ifra hvilke oppgaver som skulle gjøres når. Ettersom prosjektet har gått har det selvfølgelig vært endringer siden oppgaver enten tar lengre eller kortere tid, men å ha denne tidsestimeringen og prosjektert denne, har gjort at vi vet antatt omfang av oppgaver når vi skal planlegge sprintene.

5.9.3 Git

Git er et fantastisk verktøy som har gjort at vi kan jobbe sammen og strukturert gjennom prosjektet. Et problem vi ikke var klar over var at dersom vi kom over en viss størrelse på mappen var denne for stor for å laste opp på GitHub. Dersom vi gjorde en commit før vi pushet, var det vanskelig å fjerne den for store commiten uten å miste endringer i senere commiter. Dette skjedde da vi la til lydfiler brukt i repository-et. Derfor er det viktig å spesifisere en egen mappe for lydfiler, og få git til å ignorere denne i *.gitignore*, og heller overføre filene til hverandre med andre metoder.

6 Konklusjon

For å konkludere oppgaven har den vært en utrolig lærerik opplevelse. Det har vært interessant å ha en oppgave der vi har kombinert og videreutviklet kunnskap fra utdanningen med nye, mindre beherskede emner.

Det har vært en tung prosess å komme frem til det endelige resultatet innenfor maskinlæring, da vi har jobbet hardt for å få modellene til å fungere. I dag finnes er det lite teknologier innenfor talekjenkjenning for slike systemer på markedet, og som vi har nevnt er det begrenset med rammeverk som er ute nå. Å kunne utarbeide modellene har gjort at gruppemedlemmene kunne lære mer om et tema vi begge har ønsket å kunne mer om. Vi har resultert i å utarbeide talekjenkjenningsmodeller som i stor grad klarer å identifisere om en bruker skal være merket som validert eller ikke, som i seg selv har vært en stor gevinst. Utover det har vi i tillegg bedret modellene til å ta høyde for data som er ukjent for systemet, så denne dataen ikke med en feil kommer inn. Noe av denne dataen har vist seg å fortsatt bli merket som verifisert, som følge av ulike faktorer beskrevet i rapporten. At man kan spille av lyd fra validerte brukere og fortsatt komme inn er et sikkerhetsbrudd vi ikke har klart å komme rundt, men det vil fortsatt være et brukbart system i applikasjoner der dette ikke er kritisk, og gir samtidig en god pekepinn for hvor videre forskning innenfor tema burde fokusere.

I oppgaven har vi også utviklet et komplett system rundt modellene for talegjenkjenning. Vi har utarbeidet et administrativt system som har gjenspeilet oppdragsgivers tanker innenfor funksjonalitet. I systemet kan en administrator administrere brukere i systemet via en webapplikasjon. Systemet for talegjenkjenning er også lagret på denne serveren, slik at man kan bruke systemet uavhengig av innspillingsenhet. Dette systemet har vært distribuert på en Ubuntu Server. Vi har tatt med oss mye lærdom fra prosessen for å distribuere applikasjonen på denne.
Ved å bruke Python og Flask som backend har det krevd tid i å komme seg rundt diverse problem, da samspillet mellom ulike teknologier og protokoller ikke har vært optimalt (eks. maskinlæringsmodeller og flask, håndtering av talefiler og flask, mm.). Under utvikling av systemet har vi tatt høyde for sikkerheten av systemet med kryptert data, slik at det er vanskeligere å få innblikk i systemet fra et utenfraperspektiv. Dette gjenspeiles i databasen og ved registrering av passord.

Den siste komponenten i systemet er selve innspillingsenheten. Denne er koblet opp mot administrasjonssystemet, slik at man kan sende inn lydklipp for validering av modellene. Heretter vil enheten ta valg basert på hvilken respons systemet sender tilbake. Under Covid-19 pandemien har vi dessverre ikke hatt mulighet til å fysisk koble dette opp mot det faktiske kontoret til Avento for å begrense smittefare, og har heller simulert oppsettet hjemme. Vi har dog avtalt

at vi kan SD-kort fra vår lokale Rasperry Pi skal sendes til oppdragsgiver, slik at det er gjort til rette for at bedriften enkelt kan sette opp enheten til eget bruk.

Et krav fra oppdragsgiver er at applikasjonen skal være enkel å utvide og enkel å bruke. Vi har tatt hensyn til at det skal være enkelt å bruke tjenesten til andre applikasjoner. Ett problem er at for å få gode modeller har vi trengt å bruke lydopptak der samme frase blir sagt, som gjør at det vil være rart å for eksempel koble systemet opp mot en kaffemaskin for å brygge kaffe med frasen "Åpne dør". Likevel er det muligheter for å enkelt koble opp mot ved å endre koden i IOHandler til å si hva som skal gjøres. Å endre frase kunne man gjort ved å endre lydfilene i datasettet med tale av andre opptak. Serveren er helt uavhengig av innspillingsenheten, så dersom man for eksempel ville brukt en annen enhet enn en Raspberry Pi, kunne man fint gjort det ved å bruke de samme kallene opp mot serveren.

7 Referanser

References

[1] Porter, Kim. N/A. Hentet 28. januar 2020. https://us.norton.com/internetsecurity-iot-biometrics-how-do-they-work-are-they-safe.html

[2] Weisstein, Eric W.. 2020. Discrete Fourier Transform. Hentet 10. februar 2020. https://mathworld.wolfram.com/DiscreteFourierTransform.html

[3] Liu, Bin. 2007. Short-Time Fourier Transform. Hentet 11. februar 2020. https://www.sciencedirect.com/topics/engineering/short-time-fourier-transform

[4] DB-Engines. 2020. DB-Engines Ranking. Hentet 22. februar 2020. https://db-engines.com/en/ranking

[5] The Pallets Organization. 2020. Flask Documentation. Hentet 18. april 2020. https://flask.palletsprojects.com/en/1.1.x/

[6] Mindfire Solutions. 2018. Flask vs Django. Hentet 18. april 2020. http://www.mindfiresolutions.com/blog/2018/05/flask-vs-django/

[7] Countryman, Max. 2020. Flask-Login. Hentet 18. april 2020. https://flask-login.readthedocs.io/en/latest/flask_login.LoginManager.

[8] The Pallets Organization. 2020. Jinja Documentation. Hentet 18. april 2020. https://jinja.palletsprojects.com/en/2.11.x/

[9] Mozilla. 2020. Ajax Documentation. Hentet 19. april 2020. https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

[10] Di Gregorio, Federico. 2020. Psycopg Documentation. Hentet 19. april 2020. https://www.psycopg.org/docs/usage.html

[11] Python Software Foundation. 2020. smptlib - SMTP Protocol Client Documentation. Hentet 28. februar 2020. https://docs.python.org/3/library/smtplib.html

[12] https://www.psycopg.org/docs/usage.html. Psycopg2 Documentation. Hentet 22. februar 20. https://pypi.org/project/psycopg2/

[13] Wikipedia. 2020. Proxy server. Hentet 01. mars 2020. https://en.wikipedia.org/wiki/Proxy_server

[14] The OWASP Foundation. 2020. SQL Injection. Hentet 05. mars 2020. https://owasp.org/www-community/attacks/SQL_Injection

[15] The OWASP Foundation. 2020. Cross Site Request Forgery (CSRF). Hentet 05. mars 2020. https://owasp.org/www-community/attacks/csrf

[16] University of California, Berkeley. N/A. How to Protect Against SQL Injection Attacks. Hentet 07. mars 2020. https://security.berkeley.edu/education-awareness/best-practices-how-tos/system-application-security/how-protect-against-sql

[17] Diamond, Matt. 2016. Hentet 04. mars 2020. A plugin for recording/exporting the output of Web Audio API node. Hentet 27. februar 2020. https://github.com/mattdiamond/Recorderjs

[18] The Apache Software Foundation. 2020. Hentet 20. april 2020. Apache Documentation. https://httpd.apache.org/

[19] AlternativeTo. 2020. Alternatives to Apache HTTP Server. Hentet 20. april 2020. https://alternativeto.net/software/apache/

[20] Dumpleton, Graham. 2020. mod_wsgi Documentation. Hentet 21. april 2020. https://modwsgi.readthedocs.io/en/develop/

[21] Dumpleton, Graham. 2020. WSGI Documentation. Hentet 22. april 2020. https://wsgi.readthedocs.io/en/latest/index.html

[22] The Electronic Frontier Foundation. Certbot Homepage. Hentet 21. april 2020. https://certbot.eff.org/

[23] Vixie, Paul. 2019. Cron Manual. Hentet 03. mai 2020. http://manpages.ubuntu.com/manpages/trusty/man8/cron.8.html

[24] Chromium. 2020. Deprecating Powerful Features on Insecure Origins. Hentet 26. februar 2020. https://sites.google.com/a/chromium.org/dev/Home/chromium-security/deprecating-powerful-features-on-insecure-origins

[25] Waaler, Fredrik Dyvik, Klaus. 2020. Website for collection of sound-samples. Hentet 07. mars 2020. https://fredrifw-bachelor.uials.no

[26] The Python Software Foundation. 2019. Pep 343 – The "with" statement. Hentet 05.mars 2020. https://www.python.org/dev/peps/pep-0343/

[27] The OWASP Foundation. N/A. Authentication General Guidelines. Hentet 03. mars 2020 https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

[28] Jardine, James. 2017. Validation: Client vs. Server. Hentet 17. mars 2020. https://www.developsec.com/2017/06/19/validation-client-vs-server/

[29] Kite. 2020. Wavfile Documentation. Hentet 26.01.2020. https://kite.com/python/docs/scipy.io.wavfile

[30] The Python Software Foundation. 2020. os Documentation. Hentet 25. april 2020. https://docs.python.org/3/library/os.html

[31] The Python Software Foundation. 2020. random Documentation. Hentet 23. april 2020. https://docs.python.org/3/library/random.html

[32] The Python Software Foundation. 2020. pickle Documentation. Hentet 23. april 2020. https://docs.python.org/3/library/pickle.html

[33] The Python Software Foundation. 2020. re Documentation. Hentet 23. april 2020. https://docs.python.org/3/library/re.html

[34] Google. 2020. TensorFlow API Documentation. Hentet 01.22.2020. https://www.tensorflow.org/api_docs/python/tf

[35] Keras. 2020. Keras main Page. Hentet 01.22.2020. https://keras.io/

[36] Microsoft. 2020. The Microsoft Scalable Noisy Speech Dataset. Hentet 20. februar 2020. https://github.com/microsoft/MS-SNSD

[37] Hui, Jonathan. 2019. Speech Recognition - Feature Extraction MFCC PLP. Hentet 11. mars 2020. https://medium.com/@jonathan_hui/speech-recognition-feature-extraction-mfcc-plp-5455f5a69dd9

[38] SciPy Developers. 2020. Numpy and Scipy Documentation. Hentet 03. mai 2020. https://docs.scipy.org/doc/

[39] Robert, James. 2020. pydub - Manipulate audio with a simple and easy high level interface. Hentet 01. februar 2020. https://github.com/jiaaro/pydub

[40] Mozilla. 2020. Common Voice Home Page. Hentet 12. mars .2020. https://voice.mozilla.org/

[41] Torvalds, Linus. 2020. git Home Page. Hentet 24. april 2020. https://git-scm.com/

[42] GitHub. 2020. GitHub Home Page. Hentet 24. april 2020. https://github.com/

[43] Librosa. 2020. Librosa Documentation. Hentet 11. februar 2020. https://librosa.github.io/librosa/

[44] Pham, Hubert. 2017. PyAudio Documentation. Hentet 10. februar 2020. http://people.csail.mit.edu/hubert/pyaudio/

[45] Google. 2020. Google Cloud Home Page. Hentet 13. februar 2020. https://cloud.google.com/

[46] Google. 2020. Google Speech-To-Text API. Hentet 13. februar 2020. https://cloud.google.com/speech-to-text

[47] Ma, Edward. 2019. Data Augumentation for Audio. Hentet 23. mars 2020. https://medium.com/@makcedward/data-augmentation-for-audio-76912b01fdf6

[48] Andersen, Paul Bjørn. 2018. FFT. Hentet 31. mars 2020. https://snl.no/FFT

[49] Fayek, Haytham. 2016. Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between. Hentet 31. mars 2020. https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html

[50] Scipy Developers. 2019. numpy.hanning Documentation. Hentet 31. mars 2020. https://docs.scipy.org/doc/numpy/reference/generated/numpy.hanning.html

[51] Wikipedia. 2019. dBFS. Hentet 31. mars 2020. https://en.wikipedia.org/wiki/DBFS

[52] Scipy Developers. 2020. SciPy Home Page. Hentet 03. mai 2020. https://www.scipy.org/

[53] Cyrta, Pawel et al.. 2017. Speaker Diarization using Deep Recurrent Convolutional Neural Networks fro Speaker Embeddings. Hentet 29.01.2020. https://arxiv.org/pdf/1708.02840.pdf

[54] Torfy, Amirsina et al.. 2018. TEXT-INDEPENDENT SPEAKER VERIFICATION USING 3D CONVOLUTIONAL NEURAL NETWORKS. Hentet 04. februar 2020. https://arxiv.org/pdf/1705.09422.pdf

[55] Zhang, Aonan et al. 2019. FULLY SUPERVISED SPEAKER DIARIZATION. Hentet 04. februar 2020. https://arxiv.org/pdf/1810.04719.pdf

[56] Scipy Developers. 2020. Numpy Documentation. Hentet 17. april 2020. https://docs.scipy.org/doc/numpy/reference/

[57] Matplotlib. 2020. Matplotlib: Visualization with Python. Hentet 18. april 2020. https://matplotlib.org/

[58] SuperDataScience Team. 2018. Recurrent Neural Networks (RNN) - The Vanishing Gradient Problem. Hentet 16. april 2020. https://www.superdatascience.com/blogs/recurrent-neural-networks-rnn-the-vanishing-gradient-problem

[59] Olah, Christopher. 2015. Understanding LSTM Networks. Hentet 16. april 2020. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[60] Stanford Vision Lab, Stanford University, Princeton University. 2016. About ImageNet. Hentet 17. april 2020. http://image-net.org/about-overview

[61] The Raspberry Pi Foundation. 2015. Raspberry Pi 2 Model B. Hentet 17. april 2020 https://www.raspberrypi.org/products/raspberry-pi-2-model-b/

[62] Teknikmagasinet. N/A. Raskt microSD-kort, perfekt til smarttelefonen din. Hentet 17. april 2020. https://www.teknikmagasinet.no/produkter/gaming-data/lagringsmedia/minnekort/sandisk-ultra-microsdhc-32gb

[63] HowToDoInJava. N/A. REST Architectural Constraints. Hentet 20. april 2020. https://restfulapi.net/rest-architectural-constraints/

[64] Jetbrains r.s.o. 2020. PyCharm - The Python IDE for Professional Developers. Hentet 20. april 2020. https://www.jetbrains.com/pycharm/

[65] Kosse, Tim et al. 2020. FileZilla Home Page. Hentet 20. april 2020. https://filezilla-project.org/

[66] SSH Communications Security, Inc.. N/A. PuTTY - World's Most Popular Free SSH Client. Hentet 20. april 2020. https://www.ssh.com/ssh/putty/

[67] The Python Foundation. 2020. configparser - Configuration File Parser. Hentet 28. mars 2020. https://docs.python.org/3/library/configparser.html

[68] The Python Foundation. 2020. secrets - Generate secure random numbers for managing secrets. Hentet 21. april 2020. https://docs.python.org/3/library/secrets.html

[69] Cloudflare. N/A. What is the Internet Protocol?. Hentet 21. april 2020. https://www.cloudflare.com/learning/ddos/glossary/internet-protocol/

[70] Cloudflare. N/A. What is Transport Layer Security (TLS)?. Hentet 21. april 2020. https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/ 21. april 20 14.19

[71] Wikipedia. 2020. File Transfer Protocol. Hentet 21. april 2020. https://en.wikipedia.org/wiki/File_Transfer_Protocol

[72] SSH Communications Security Inc.. N/A. SSH Protocol. Hentet 21. april 2020. https://www.ssh.com/ssh/protocol/

[73] The OWASP Foundation. N/A. OWASP DOM-bases XSS. Hentet 21. april 2020. https://owasp.org/www-community/attacks/DOM_Based_XSS

[74] Arias, Dan. 2018. Hashing in Action: Understanding bcrypt. Hentet 21. april 2020. https://auth0.com/blog/hashing-in-action-understanding-bcrypt/

[75] Countryman, Max. 2020. Flask-Bcrypt. Hentet 21. april 2020. https://flask-bcrypt.readthedocs.io/en/latest/

[76] IBM Cloud Education. 2019. Relational Databases. Hentet 21. april 2020. https://www.ibm.com/cloud/learn/relational-databases

[77] SSH Communications Security Inc.. N/A. SFTP - SSH Secure File Transfer Protocol. Hentet 21. april 2020. https://www.ssh.com/ssh/sftp/ 21. april 20 14.40

[78] Putano, Ben. 2019. A Look At 5 of the Most Popular Programming Languages of 2019. Hentet 22. april 2020. https://stackify.com/popular-programming-languages-2018/

[79] Voskoglou, Christina. 2017. What is the best programming language for Machine Learning?. Hentet 22. april 2020. https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7

[80] Synopsys Inc.. N/A. Compare Repositories. Hentet 22. april 2020. https://www.openhub.net/repositories/compare

[81] Dubovikov, Kirill. 2017. PyTorch vs TensorFlow - spotting the difference. Hentet Hentet 22. april 2020. https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b

[82] MagicStack Inc.. 2020. asyncpg Documentation. Hentet 22. april 2020. https://pypi.org/project/asyncpg/

[83] Countryman, Max. 2020. Flask-Login Documentation. Hentet 22. april 2020. https://flask-login.readthedocs.io/en/latest/

[84] The Pallets Organization. 2020. Message Flashing. Hentet 22. april 2020. https://flask.palletsprojects.com/en/1.1.x/patterns/flashing/

[85] Microsoft. 2020. Azure Cognitice Services Speaker Recognition API. Hentet 22. april 2020. https://azure.microsoft.com/en-us/services/cognitive-services/speaker-recognition/

[86] Amazon. 2020. Amazon Transcribe API - Identifying speakers. Hentet 22. april 2020. https://docs.aws.amazon.com/transcribe/latest/dg/how-diarization.html

[87] Image of CNN Architecture. Hentet 23. april 2020 https://miro.medium.com/max/2000/1*vkQ0hXDaQv57sALXAJquxA.jpeg

[88] Bhanja, Samit, Das, Abhishek. 2018. Impact of Data Normalization on Deep Neural Network for Time Series Forcecasting. Hentet 23. april 2020. https://arxiv.org/pdf/1812.05519.pdf

[89] Wikiperdia. 2020. QuickTime. Hentet 23. april 2020. https://en.wikipedia.org/wiki/QuickTime

[90] Image of RNN Architecture. Hentet 23. april 2020. https://miro.medium.com/max/375/1*chs1MCz2rCK4_dFRLnUEIg.png

[91] Steinmetz, Christian. 2018. pyloudnorm. Hentet 23. april 2020 https://www.christiansteinmetz.com/projects-blog/pyloudnorm

[92] Pandas Development Team. 2020. pandas - Open Source Data Analysis. Hentet 23. april 2020 https://pandas.pydata.org/

[93] Stevens, S. S. et al. 2005. A Scale for the Measurement of the Psychological Magnitude Pitch. Hentet 06. mai 2020. https://psycnet.apa.org/record/1937-01149-001

[94] CircuitLab Inc.. 2020. CircuitLab - Circuit simulation and schematics. Hentet 06. mai 2020. https://www.circuitlab.com/

[95] The Raspberry Pi Foundation. N/A. GPIO. Hentet 08. mai 2020. https://www.raspberrypi.org/documentation/usage/gpio/

[96] Gamborg, Marius et al.. 2005. SIGNALREPRESENTASJONER FOR AUTOMATISK TALEGJENKJENNING. Hentet 11. mai 2020. https://publications.ffi.no/nb/item/asset/dspace:3142/05-01053.pdf

[97] Ecma International. 2020. Ecma standards. Hentet 11. mai 2020. https://www.ecma-international.org/publications/standards/Standard.htm

[98] Ramesh, Shashank. 2018. A guide to an efficient way to build neural network architectures- Part II: Hyper-parameter selection and tuning for Convolutional Neural Networks using Hyperas on Fashion-MNIST. Hentet 13. mai 2020. https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7

[99] Brownlee, Jason. 2018. A Gentle Introduction to Dropout for Regularizing Deep Neural Networks. Hentet 13. mai 2020. https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

[100] Borovicka, Tomas et al.. 2011. Selecting Representative Data Sets. Hentet 13. mai 2020. https://cdn.intechopen.com/pdfs/39037/InTech-Selecting_representative_data_sets.pdf

[101] Goyal, Kechit. 2020. What is Overfitting Underfitting In Machine Learning ?. Hentet 13. mai 2020. https://www.upgrad.com/blog/overfitting-underfitting-in-machine-learning/

[102] Torrey, Lisa et al.. 2009. Transfer Learning. Hentet 13. mai 2020. https://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf

[103] Martin, Scott. 2019. What Is Transfer Learning?. Hentet 13. mai 2020. https://blogs.nvidia.com/blog/2019/02/07/what-is-transfer-learning/

[104] Google. 2020. TensorFlow tutorials. Hentet 13. mai 2020. https://www.tensorflow.org/tutorials

[105] Singh, Kundan. 2013. How To Deploy a Flask Application on an Ubuntu VPS. Hentet 14. mai 2020. https://www.digitalocean.com/community/tutorials/how-to-deploy-a-flask-application-on-an-ubuntu-vps

[106] PostgreSQL Global Development Group, 2020. Psql. Hentet 03. mai 2020. https://www.postgresql.org/docs/9.3/app-psql.html

[107] Ellingwood, Justin, et. al. 2018. How To Install and Use PostgreSQL on Ubuntu 18.04. Hentet 03. mars 2020. https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-18-04

[108] Electronic Frontier Foundation. 2020. certbot instructions. Hentet 03. mars 2020. https://certbot.eff.org/lets-encrypt/ubuntubionic-apache

[109] Librosa Development Team, 2019. librosa.core.stft. Hentet 21. februar 2020. http://librosa.github.io/librosa/generated/librosa.core.stft.htmllibrosa.core.stft

[110] Lewis, Jerad. 2013. Understanding Microphone Sensitivity. Hentet 04.april 2020. https://www.mouser.tw/pdfDocs/Understanding-Microphone-Sensitivity-4.pdf

[111] Google, 2020. Transcribing audio from streaming input. Hentet 16. februar 2020. https://cloud.google.com/speech-to-text/docs/streaming-recognize

[112] Wang, Jason, et. al. 2017. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. Hentet 19. februar 2020. http://cs231n.stanford.edu/reports/2017/pdfs/300.pdf

[113] Carranza, Pablo. 2018. How To Use Google's SMTP Server. Hentet 08. februar 2020. https://www.digitalocean.com/community/tutorials/how-to-use-google-s-smtp-server

[114] MDN Contributors, 2020. navigator.MediaDevices.getUserMedia. Hentet 01. mars 2020. https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia

[115] Wichers, Dave, et.al. 2020. Cross Site Request Forgery (CSRF). Hentet 14. mars 2020. https://owasp.org/www-community/attacks/csrf

[116] Google, 2020. Tensorflow Serving. Hentet 19. april 2020. https://www.tensorflow.org/tfx/guide/serving

[117] Kjell Company. 2020. Plexgear m4 Mikrofon. Hentet 08. mai 2020. https://www.kjell.com/no/produkter/data/datatilbehor/mikrofoner/plexgear-m-4-mikrofon-p24051

[118] Kjell Company. 2020. Plexgear Usb Lydkort. Hentet 08. mai 2020. https://www.kjell.com/no/produkter/data/datamaskinkomponenter/lydkort/usb-lydkort/plexgear-usb-lydkort-for-pc-p31624

[119] Avento AS. 2020. Hentet 09. mai 2020. https://www.avento.no/

[120] GDB Developers. 2020. GDB: The GNU Project debugger. Hentet 02. mai 2020. https://www.gnu.org/software/gdb/

[121] Thomp, Jac. 2020. Greedy version requirement for werkzeug breaking fresh installs. Hentet 17. april 2020. https://github.com/pallets/flask/issues/3481

Vedlegg

- **Vedlegg 1** - Forprosjektrapport
- **Vedlegg 2** - Kravspesifikasjon
- **Vedlegg 3** - Deployment Diagram
- **Vedlegg 4** - Gantt-Skjema
- **Vedlegg 5** - UML Diagram
- **Vedlegg 6** - Sprintrapporter
- **Vedlegg 7** - SQL
- **Vedlegg 8** - Readme for server
- **Vedlegg 9** - Wireframes
- **Vedlegg 10** - Flowchart Github
- **Vedlegg 11** - Kildekode for Administrasjonssiden (I mappen AdminWebsite)
- **Vedlegg 12** - Kildekode for Inspillingsenhet (I mappen RPiSystem)
- **Vedlegg 13** - Kildekode for Testsystem for modeller (I mappen VoiceRecognition)
- **Vedlegg 14** - Innsamlede lydfiler (I mappen VoiceFiles)

7.1 Vedlegg 1 - Forprosjektrapport

Tittel:

Talegjenkjenning for låssystem

Kandidatnummer(e):

Klaus Dyvik: 492835
Fredrik Ferdinand Waaler: 492851

Dato: 21.01.20	**Emnekode:**	**Emne:**	**Dokument tilgang:**
	IE303612	Bacheloroppgave (Data)	- Åpen
Studium:		**Ant sider/Vedlegg:**	**Bibl. nr:**
Dataingeniør		22 /	- Ikke i bruk -

Oppdragsgiver(e)/Veileder(e):

Oppdragsgiver: Avento AS

Hovedveileder: Saleh Abdel-Afou Alaliyat

Biveileder: Anniken Karlsen

Oppgave/Sammendrag:

Avento AS foreslo opprinnelig en åpen oppgave, der målet var å implementere en løsning slik at man kunne ta i bruk talegjenkjenning for å regulere tilgang til inngangsdør. Bedriften ønsket at løsningen skulle være enkel å utvide, slik at den f.eks. også kunne kobles opp mot en kaffemaskin, og brukere kunne sette i gang kaffetrakting og få kaffe etter deres preferanse.

Etter videre utredning, er det bestemt at oppgaven nå innebærer å utvikle egne modeller (ved hjelp av kunstig intelligens og maskinlæring) for talegjenkjenning. Målet er fortsatt at disse modellene skal kunne brukes for å regulere tilgang til bedriftens inngangsdør, samt kunne utvides til å ta i bruk andre steder. Oppgaven innebærer også å implementere annen funksjonalitet som ligger til grunn for at et slikt system skal kunne være funksjonelt. Dette inkluderer administrative systemer (for brukerbehandling, loggføring, etc.), funksjonalitet for registrering av nye brukere, samt tilhørende støttefunksjonalitet som f.eks. databaser.

1 INNLEDNING

Begge gruppemedlemmer tilbrakte høsten 2019 på utveksling i utlandet, og hadde en plan allerede ved avreise at vi skulle skrive bacheloroppgave sammen. Vi var forberedt på at vi måtte velge en oppgave allerede før vi kom hjem. Etter god dialog og evaluering av egenskapene våre hadde vi lyst til å jobbe med databaser og/eller kunstig intelligens. Av de foreslåtte oppgavene rangerte vi tre oppgaver derav én oppgave med fokus på databaseoptimalisering og to oppgaver med kunstig intelligens. Rekkefølge hadde ikke så mye å si, men vi ble tildelt vårt førstevalg. Bedriften som hadde utlyst denne oppgaven trakk seg i siste stund, men heldigvis kunne vi skrive oppgave for Avento istedenfor.

Avento hadde et testprosjekt i fjor, der ansatte kunne laste opp bilder av seg selv, slik at et kamera verifiserer om det er samme person for å låse opp døren til deres arbeidslokale. Dette ble gjort ved bruk av relativt primitive komponenter på en Raspberry Pi (liten datamaskin).

De ønsket nå å se på lignende løsninger ved bruk av talegjenkjenning istedenfor bildegjenkjenning - primært for å erstatte bruk av RFID-chip i åpningstiden fra 08-16, men dersom mulig også for å erstatte/supplementere bruk av pin-kode utenom åpningstider. Forslag fra bedriften var at dette kunne gjøres ved å koble en mikrofon til Raspberry Pi som allerede var stasjonert utenfor bedriftens lokale. Bedriften ønsket også at en slik løsning skulle være utvidbar, slik at den f.eks. skulle kunne kobles opp mot kaffemaskinens API, slik at de ansatte kan bruke stemmen sin for å igangsette trakting på bedriftens kaffemaskin.

Vår grunnleggende problemstilling er dermed å finne ut av hvordan vi kan implementere en verifiseringsmekansime med talegjenkjenning som kan brukes for låssystemer for dører, samt utvikling av et slikt system.

2 BEGREPER

API - Application Programming Interface

RNN - Recurrent Neural Network

CNN - Convolutional Neural Network

3 PROSJEKTORGANISASJON

3.1 Prosjektgruppe

Studentnummer(e)
492835 - Klaus Dyvik 492851 - Fredrik Ferdinand Waaler

3.1.1 Oppgaver for prosjektgruppen - organisering

Prosjektgruppen har et delt ansvar under hele prosjektet. For oppfølging av at Scrum-metodikk utføres på en ordentlig måte har Klaus fått tildelt rollen som Scrum-master.

3.1.2 Oppgaver for prosjektleder

Vi har bestemt oss for at begge gruppemedlemmer deler ansvaret som ellers ville falt på en prosjektleder. Det vil si at begge gruppemedlemmer i felleskap er ansvarlige for følgende:

- Planlegging
- Evaluering
- Arbeidsfordeling
- Kontakt med bedrift og veileder
- Sørge for at mål nås i tide
- Sørge for god kommunikasjon

Delegasjon av arbeidsoppgaver vil bli forsøkt fordelt på en rettferdig basis. Sammen vil vi gjøre vårt beste for å delegere oppgaver slik at arbeidsmengden grovt sett blir jevnt fordelt og slik at begge gruppemedlemmer får ønskelige arbeidsoppgaver i den grad dette er mulig.

3.1.3 Oppgaver for sekretær

Det blir begge gruppemedlemmers oppgave å sammen sørge for at info fra møter med både arbeidsgiver, veileder(e) og andre relevante personer blir notert ned til den grad det er nødvendig.

3.2 Styringsgruppe (veileder og kontaktperson oppdragsgiver)

Oppdragsgiver:

- Kontaktperson: Oskar Emil Skeide, oskar.emil.skeide@avento.no

Veiledere:

- Hovedveileder: Saleh Abdel-Afou Alaliya, alaliyat.a.saleh@ntnu.no
- Biveileder: Anniken Susanne T. Karlsen, anniken.t.karlsen@ntnu.no

4 AVTALER

4.1 Arbeidssted og ressurser

Rom L167 ved NTNU i Ålesund er satt av til studenter, og grunnet nærliggende beliggenhet til gruppemedlemmer blir dette hovedarbeidsplass. Gruppen har også fått Avento sine lokaler ved Breivika i Ålesund til disposisjon, som vil bli brukt ved behov.

Ved Avento har vi kontakt med Oskar Emil Skeide, Vegard Vatn og Torstein Ødegård, der førstnevnte per nå er gruppens kontaktledd med bedriften. Dette gjør vi gjennom bedriftens prefererte kommunikasjonsmiddel: Microsoft Teams.

De fleste ressurser som vil bli brukt i dette prosjektet er dekket av studentmedlemskap for diverse utviklingsverktøy. Her kan vi som oftest prøve ut funksjonalitet før vi eventuelt må kjøpe et produkt/utvide funksjonalitet. I utgangspunktet regner vi ikke med at vi blir nødt til å kjøpe ressurser utenom det som er tilgjengelig gjennom skolen eller bedriften vi jobber for, med unntak av en mikrofon og muligens en Raspberry Pi (om det ikke er tilgjengelig gjennom arbeidsgiver).

Vi regner heller ikke med at vi kommer til å trenge tilgang til informasjon som er underlagt sikkerhetsbestemmelser på noen som helst måte. I verste fall vil dette være opptak av stemmene til bedriftens ansatte, men vi vil ha egen testdata som vi kan bruke for publisering, slik at dette ikke burde være et problem.

Vi vil rapportere til veiledere annenhver torsdag da vi møtes for å diskutere fremgang, med forbehold om endring i møtedag. Rapportering til bedrift vil skje ved behov eller så ofte bedriften skulle ønske det.

4.2 Gruppenormer – samarbeidsregler – holdninger

I bacheloroppgaven skal vi følge arbeidsmetodikken Scrum, der sprintene skal være på 2 uker. Vi skal møte for arbeid alle ukedager fra 09.15 – 16.00 om ikke annet er avtalt. Hver morgen skal gruppen gjennomføre et stand-up møte der vi forteller hva vi gjorde forrige dag, hva vi skal fokusere på den nåværende dagen, samt hvilke utfordringer vi står ovenfor. Disse møtene er grunnlag for daglig logg, som til dels er referat fra dette møtet og er ment å sikre

god kommunikasjon innad i gruppen. Hver oppgave som skal utføres skal legges inn i en backlog så vi får logget alt arbeid. Det er viktig å være konsekvent med å flytte oppgavene over til de ulike statusfanene: *å gjøre, under arbeid* og *fullført*. Det samme gjelder å pushe fullførte arbeidsoppgaver til git-repository, slik at det ikke oppstår mange konflikter under fusjonering av arbeidsgrener.

Begge gruppemedlemmer er innforstått med at selv om visse arbeidsoppgaver tildeles enkeltpersoner, betyr ikke det at ***ansvaret*** for at denne oppgaven blir godt gjennomført kun faller på personen som blir tildelt oppgaven. Gruppemedlemmene er ansvarlige for å ha en helhetlig forståelse for hverandres arbeid, og alle deler av det ferdigstilte prosjektet. Hverandres arbeidsoppgaver skal ha like høy prioritet for begge parter.

Gruppemedlemmene forplikter seg ellers til å påta seg nødvendig arbeid utenfor avtalt arbeidstid etter behov - dette inkluderer også ekstra tid som må brukes for å lære nødvendig materiale. Gruppen skal gjøre sitt beste for å imøtekomme krav og ønsker fra bedriften, til den grad det er fornuftig. Ellers sikter gruppen på å holde kommunikasjon med arbeidsgiver så kontinuerlig og åpen som mulig, slik at de er innforstått med alle avgjørende valg som tas underveis og slik at vi med høy sannsynlighet vil ende med et ferdig prosjekt som tilfredsstiller alle parters ønsker.

5 PROSJEKTBESKRIVELSE

5.1 Problemstilling - målsetting - hensikt

Hovedsakelig omfatter problemstillingen vår hvordan vi kan implementere en verifiseringsmekansime for låssystemer til dører ved bruk av talegjenkjenning. Ønskelig skal denne implementasjonen være såpass lett å utvide at den skal være enkel å tilpasse til andre systemer, som f.eks. til bruk av en kaffemaskin.

Ved prosjektets ende vil vi forhåpentligvis ha utviklet en løsning som gjenkjenner brukere basert på stemme og som med god sikkerhetsmargin ikke gjør feilslutninger om hvilke brukere som burde verifiseres, spesielt med tanke på "false-positives". Dette vil forhåpentligvis gi arbeidsgiver en enkel men sikker måte for å regulere tilgang til egne lokaler. En slik løsning vil også innebære administrative systemer for bl.a. behandling av brukerbase og loggføring, men også en utvidelse som gjør det mulig for administratorer å registrere brukere, eller for administratorer å la valgte brukere kunne registrere seg selv.

5.2 Krav til løsning eller prosjektresultat

Ansatte hos Avento skal kunne få tilgang til kontorarealet ved bruk av stemmen sin. Det legges vekt på at løsningen vil være funksjonell, til den grad det vil være mulig å åpne dør med stemme, slik at den kan tas i bruk av bruk av bedriften. Kanskje enda viktigere vil det være at vi har gjort nøye testing av ulike alternativer for implementering, slik at det er lagt et godt grunnlag for hva som er mulig/ikke-mulig og evt. hva som burde satses på videre dersom løsningen ikke rekkes å ferdigstilles. Det vil heller ikke ha mye nytte for seg dersom løsningen er funksjonell, men ikke sikker, ettersom det skal brukes til et så sikkerhetskritisk formål. Det vil derfor også legges stor vekt på hvorvidt løsningen(e) som utvikles er sikre, til den grad at antallet "false-positives" er lavt, og brukervennlig, til den grad at antallet "false-negatives" er lavt.

Etter avtale med arbeidsgiver vil denne rapporten fungere som en slags offisiell oppgavebeskrivelse. Med forutsetning at arbeidsgiver finner innholdet i rapporten

tilstrekkelig, vil dette brukes som et dokument for å måle hvorvidt prosjektets mål er nådd. Dersom dette ikke er tilfellet, vil en ny oppgavebeskrivelse utredes i samarbeid med arbeidsgiver og denne vil brukes som tilsvarende dokument.

5.3 Planlagt framgangsmåte(r) for utviklingsarbeidet – metode(r)

I starten vil mye av tiden gå til å tilegne seg informasjon om hvordan vi kan utvikle systemet for talegjenkjenning, da kunstig intelligens er et relativt nytt emne for samtlige gruppemedlemmer. Etter oppfordring fra veileder planlegger vi å teste ut fremgangsmåter for binær klassifisering (mellom verifiserte og ikke-verifiserte brukere) med ulike modeller som "Black-box"-metoder, recurrent neural networks og convolutional neural networks.

For å få testet disse modellene så godt som mulig er det viktig at vi har gode datasett som vi kan bruke til å trene algoritmene, i startfasen kommer vi derfor også til å prioritere tidsbruk på å samle inn slike data. Underveis- og etter endt utvikling vil vi sammenligne disse modellene, slik at vi kan gjøre nye vurderinger for hvilke modeller som burde prioriteres i arbeidet fremover. Det vil først og fremst legges vekt på at vi ønsker en så sikker modell som mulig, men også en som legger til rette for brukervennlighet.

I parallell med dette arbeidet vil vi begynne å se på utvikling av et system for administratorer og et registreringssystem. Ettersom dette er ganske fort gjort (i forhold til utvikling av modellene for talegjenkjenning) og ettersom begge gruppemedlemmer har god erfaring med utvikling av slike systemer, vil dette sannsynligvis ikke prioriteres før i de senere fasene av prosjektet.

5.4 Informasjonsinnsamling – utført og planlagt

Det finnes et par API-er som prøver seg på verifisering gjennom talegjenkjenning allerede, men disse er begrenset i form av funksjonalitet og fleksibilitet. Den beste løsningen for talegjenkjenning vi fant var *Azure Cognitive Services* fra Microsoft. Her måtte man identifisere en bruker før man eventuelt kunne verifisere brukerens stemme, siden man måtte

sende inn bruker-id når man skulle bruke tjenesten. Denne løsningen krevde også at man måtte sende inn lyd av én av de eksisterende ti frasene når man skulle identifisere en person. I og med at funksjonaliteten API'et tilbyr er såpass begrenset, fant vi i samarbeid med veileder ut at det vil være bedre å forsøke å utvikle egne modeller for talegjenkjenning. Dette vil gi oss økt fleksibilitet, i tillegg til at en løsning som baserer seg på et slikt API sannsynligvis ville blitt for enkel for en Bacheloroppgave i seg selv. Det kan dog være greit å vite at en slik løsning eksisterer, dersom våre forsøk på å utvikle egne modeller feiler og det enda er tid til å utvikle et funksjonelt system ved hjelp av dette API'et.

Utover Microsoft sin løsning så det ikke ut som at mange andre store firmaer hadde åpnet for bruk av talegjenkjenning-tjenesten deres, selv om det er kjent at de bruker det i sine egne produkter (Siri, Google Assistant, Amazon Echo, etc.). Av de andre løsningene vi fant, så det ut til at samtlige led av samme begrensninger som Microsoft's API, eller hadde annen, mer begrensende funksjonalitet.

Etter oppfordring fra veileder ble vi som nevnt oppfordret til å se på Black-box metoder, Recurrent Neural Networks og Convolutional Neural networks. Veileder har tilsendt oss et par artikler med god informasjon om hvordan noen av disse metodene fungerer og hvordan vi kan implementere slike løsninger til vårt formål. For videre informasjonssamling finnes det utallig informasjon på nettet, f.eks. forskningsartikler eller bloggposter fra andre som har implementert delvis lignende løsninger.

Vi blir også nødt til å hente inn informasjon om hvordan vi kan trekke ut data fra lydopptak for å unikt gjenkjenne stemmer. Per nå har vi, på oppfordring fra veileder, kontaktet NTNUs Kai Erik Hoff, i håp om at han har noen gode referanser til litteratur som omfatter emnet eller tips for hvordan slik informasjon kan hentes ut. Det finnes også flere artikler på nett som omfatter dette emne, f.eks. gir et kjapt google-søk flere gode resultater. I og med at maskinlæring og kunstig intelligens de siste årene har blitt såpass populære emner, regner vi ikke med at ha betydelig vanskeligheter i denne informasjonsinnsamlingen.

5.5 Vurdering – analyse av risiko

Det er tydelig at det finnes begrensninger på helt sikre låssystemer med talegjenkjenning, da slik teknologi ikke allerede finnes på markedet - tross populariteten for kunstig intelligens og

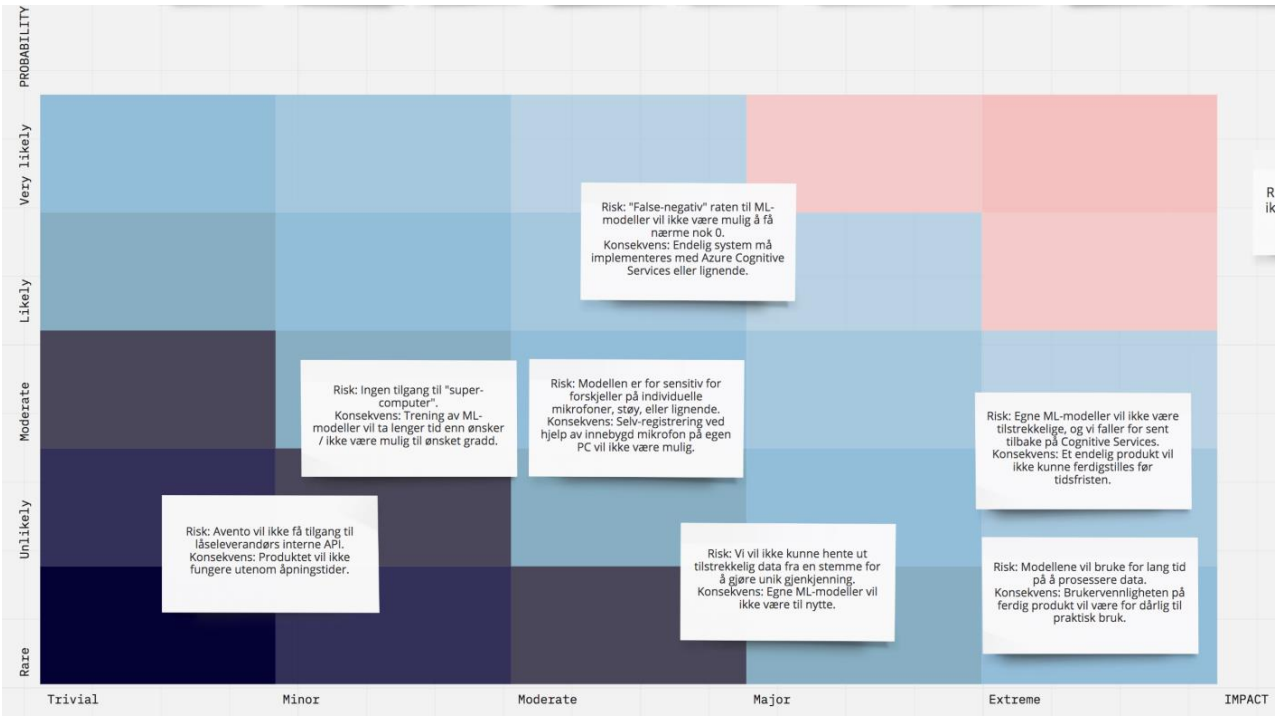
maskinlæring. I den grad vi får laget og testet ut flere modeller for et slikt system og/eller laget et funksjonelt system, regner vi med at sannsynligheten for å realisere prosjektet i tide er stor. Hvorvidt vi klarer å få "falsk-positiv"-ratioen til en modell så nære null at den i praksis vil kunne brukes av arbeidsgiver, er dog veldig usikkert på dette tidspunktet.

Dersom vi - etter omfattende arbeid med testing av egne modeller - finner at disse fremgangsmåtene ikke fungerer, kan vi bli nødt til å støtte oss på bruk av eksisterende tjenester (som f.eks. Microsoft sin *Azure Cognitive Services)* for å lage en funksjonell løsning. I dette tilfellet må en bruker først identifiseres, før brukeren kan verifiseres ved hjelp av stemme gjennom den eksisterende tjenesten. For å gjennomføre denne identifiseringen kan vi f.eks. bruke Aventos eksisterende tjeneste for ansiktsgjenkjenning.

For å lykkes skal ansatte i Avento kunne bruke stemmen sin til å komme inn på kontoret, dette uansett om det blir gjort ved hjelp av en modell vi har laget fra bunnen av eller en som støtter seg på eksisterende teknologi. At våre modeller ikke fungerer like godt betyr ikke at vi har gjort noe feil, men bidrar derimot til å evaluere i hvilken grad slike modeller faktisk kan brukes i talegjenkjenning. Vi ser til arbeid som utprøvelse og får heller adaptere dersom resultater ikke er tilstrekkelige.

Vi blir også nødt til å sørge for at modellene vi utvikler ikke er sårbare for angrep fra ikke-autoriserte brukere. En reell fare er f.eks. at en angriper kan gjøre opptak av stemmen til en registrert bruker, for så å bruke dette opptaket til å komme seg inn. Forhåpentligvis blir modellene såpass sikre at dette ikke blir et problem. Samtidig må vi være så nøye som mulig med å trene opp modellene med så representativ data som mulig. F.eks. slik at modellene ikke er predisponerte for å autorisere visse brukere. Det er også viktig at modellene er trent opp med data slik at den ikke utviser partiskhet for irrelevante karakteristikker, som f.eks. brukerens kjønn. Nye fremskritt i hacking av modeller for maskinlæring gjør også at dette kan være reelle sikkerhetstrusler vi må beskytte oss mot.

Aktuelle risikoer for prosjektet kan sammenfattes i risiko-matrisen under. Desto mer sannsynlig at en risiko vil inntreffe, desto lenger opp vil risikoen være plassert i matrisen. Desto større konsekvens risikoen vil ha, desto lenger mot høyre vil risikoen være plassert i matrisen.



5.6 Hovedaktiviteter i videre arbeid

Nr	Hovedaktivitet	Tid/omfang (timer)
A0	**Innhenting av informasjon**	**N/A**
A1	**System for talegjenkjenning**	**400**
A1.1	Innhenting av lyd	10
A1.2	Prosessering av lyd	30
A1.3	Black-box	N/A
A1.4	RNN	N/A
A1.5	CNN	N/A
A2	**Konfigurering av Raspberry Pi**	**30**
A2.1	Oppsett av taleopptak	15
A2.2	Sende lyd til server	15
A3	**Databaser**	**50**
A3.1	Database for taleopptak	25
A3.2	Database for statistikk	25
A4	**Server-konfigurering og deployment**	**30**
A5	**Nettside for administrasjon**	**100**
A5.1	GUI Nettside	50
A5.2	Logikk Nettside	50
A6	**System for registrering**	**N/A**
A99	**Integrering i eksisterende låssystem**	**N/A**

Prosjektet gjennomføres ved hjelp av den agile arbeidsmetodikken scrum, som gjør at ansvar blir fordelt ved oppstart av en ny sprint. Som vedlegg av forprosjektrapporten ligger et Gantt diagram som viser planen for når vi skal jobbe med de forskjellige komponentene.

5.7 Framdriftsplan – styring av prosjektet

5.7.1 Hovedplan

A0 Innhenting av informasjon

For å komme i gang med prosjektet må mye lesing og tilegning av informasjon om maskinlæring og nevrale nettverk til. Dette gjøres ved siden av forprosjektet i startfasen, slik at vi kan begynne på oppgaven så fort vi er ferdig med forprosjektet. Selv om forprosjektet er over, ser vi det også som nødvendig å undersøke forskjellige løsninger på nettet. Av denne grunn er det vanskelig å sette tidsbruk på dette, siden det er kontinuerlig arbeid utover prosjektet. Men vi antar at en betydelig mengde tid kommer til å bli lagt ned for å innhente nødvendig informasjon.

A1 System for talegjenkjenning

Systemet for talegjenkjenning er satt som A1, siden det er det mest omfattende punktet i prosjektet. Gruppemedlemmer skal se på ulike modeller, og finne ut av fordeler og ulemper med disse. For å kunne dekke flest modeller slik at utfallet blir best mulig, skal vi undersøke og teste forskjellige metoder hver for oss, slik at vi kan drøfte over fordeler og ulemper bedre. I begynnelsen ser vi på de mest utbredte metodene black-box, RNN og CNN. Da utforskning og prøving er essensielt for systemet ser vi det vanskelig å sette tidsbruk på dette, men vi antar det tar opp til 400 timer totalt sett.

A.1.1 Innhenting av lyd

Før systemet kan lages er det fordelaktig at vi har nok data både til å teste og trene maskinen vi skal lage. Som en del av forarbeidet må derfor lydprøver bli hentet inn. Det er fordelaktig at lydprøvene er i forskjellige miljø, forskjellige tidspunkt på dagen og av begge kjønn. Opptak tas av medstudenter eller ansatte ved universitetet. Hver opptak blir markert med navn og sted slik at ulike opptak kan skilles fra hverandre. Vi antar å ta opp lyd fra minst ett hundre personer som vil ta omtrent seks timer totalt sett.

A1.2 Prosessering av lyd

Når innhenting av lyd er ferdig må den prosesseres. Det må utforskes måter å hente ut karakteristikker av lydprøver som er unikt for hver stemme. Funksjonalitet for å automatisk hente ut disse karakteristikkene må også utvikles, dersom slik teknologi ikke allerede finnes.

A1.3 Black-box

Black-box er en metode der mye av læringen skjer i et gjemt lag, der man heller ser på hva som går inn og ut av systemet. Det er mulig at denne metoden kan oppdage mønstre i tale vi vil trenge for gjenkjenning.

A1.4 RNN

Recurrent neural networks er modeller som bruker en tidsserie til å se på utfallet til maskinen. Siden lyd er en tidsserie kan dette være en god modell for å beskrive mønstre i talen.

A1.5 CNN

Convolutional neural networks er modeller der bilder blir delt opp og kan av dette lære seg neste utfall. Derfor må vi gjøre om et lydsignal til et bilde før det sendes inn i maskinen.

A2 Konfigurering av Raspberry Pi

En Raspberry Pi må settes opp til å lytte på omgivelsene med en mikrofon. Denne skal plasseres ved inngangsdøren til Avento og sende lyd den tar opp til systemet vårt for autorisasjon. Avento har allerede en Raspberry Pi, men det er usikkert om det kreves tid på å optimalisere enheten for vår bruk.

A2.1 Oppsett av taleopptak

Vi må sette opp et system for taleopptak som tar opp lyd. Denne skal lytte på omgivelsene og sende inn lyd når aktivitet skjer. Det må sjekkes om det er tilstrekkelig lydaktivitet for at døren skal åpnes.

A2.2 Sende lyd til server

Etter at lyd blir tatt opp må den sendes over til serveren der løsningen er utplassert for å se om en person skal ha mulighet til å komme inn eller ikke. Her skal også lyden lagres, og eventuelt videre trene maskinen.

A3 Databaser

For å lagre informasjon må det lages et par databaser. Hvilken type database vi bruker kommer an på hvordan de forskjellige modellene kommuniserer med løsningen.

A3.1 Database for taleopptak

En av databasene skal holde på alle taleopptak som brukes i treningen av modellen. Her bør det stå hvem talene hører til, hvor talen er tatt opp og om det er en autorisert tale. Det er mulig at vi må trene maskinen flere ganger, så denne databasen må klare å kommunisere raskt med systemet.

A3.2 Database for statistikk

Hvordan modellen har fungert bør loggføres, både for bruk av testing og for sikkerheten til oppdragsgiver i fremtiden. Hvor god informasjon som blir hentet ut av systemet etter autorisasjon er per nå uvisst.

A4 Server-konfigurering og deployment

Ferdigstilt løsning på utplasseres på en server, slik at en Raspberry-pi kan kommunisere med løsningen over nett. Arbeid vil måtte gjøres for å klargjøre løsningen for “deployment” til server og for å konfigurere serveren slik at den er skikket og sikker til bruk.

A5 Nettside for administrasjon

Det må lages et rammeverk for produktet, slik at oppdragsgiver kan følge med på aktiviteten i systemet og administrere brukerbasen. Her bør all statistikk om hvor godt modellen fungerer vises, som for eksempel nøyaktighet, sensitivitet og spesifisitet. I tillegg kan en logg over aktivitet være interessant. Ut ifra tidligere erfaring har vi estimert at denne nettsiden burde være mulig å implementere på rundt 80 timer, dvs. en samlet arbeidsuke for begge gruppens medlemmer.

A6 System for registrering

Når Avento anskaffer seg nye ansatte eller venter gjester kan det hende de har lyst til å registrere disse i talegjenkjenningssystemet. Hvordan dette skal implementeres er fortsatt et spørsmålstegn. Derfor er det enda vanskelig å estimere tidsbruk på denne delen av prosjektet. Det er usikkert om en modell gir bedre resultater dersom all lyd er tatt opp av samme mikrofon, så avhengig av resultatet på treningen er dette punktet åpent. Dersom modellen fungerer like bra uavhengig av hvilken mikrofon man bruker, er det fordelaktig at en nettside opprettes slik at ansatte kan registrere seg hjemmefra.

A99 Integrering i eksisterende låsesystem

I testprosjektet oppdragsgiver hadde i fjor ble det ikke sendt et direkte signal til låsen. Dersom låsleverandøren deres åpner for det, ser vi på mulighet for å kunne snakke direkte med låsen for å åpne opp døren. Da arbeidsgiver ikke enda har tilgang til et API for eksisterende låsesystem, er det per nå vanskelig å si hvor lang tid slik integrering vil ta. Gitt gruppens erfaringer med ulike API'er estimerer vi at slik integrering skal kunne gjøres relativt fort (dvs. ikke ta en betydelig del av tidsbruken for prosjektet).

Milepæler (tentative datoer)

30.04.20 - Produkt skal være levert.

15.05.20 - Oppgaven skal være skrevet ferdig, og kun rettskriving skal gjenstå.

20.05.20 - Oppgaven skal leveres

Da gruppen skal jobbe med en "agile" arbeidsmetodikk, settes det pr. nå ingen videre milepæler. Det tas dog hensyn til at sprint-loggen skal legges opp slik at samlet estimert tid legger til rette for at prosjektet blir ferdig i tide. Hvilke oppgaver som prioriteres og hvilke komponenter som blir ferdig til hvilken tid, vil bli avgjort etter behov.

Viktige beslutninger

29.02.20 - Det skal besluttes om gruppen skal fortsette med utarbeiding av egen maskinlæringsmodell framfor å bruke mindre fleksible eksisterende rammeverk.

5.7.2 Styringshjelpemidler

Git - For versjonskontroll og tilbakesporing av kode, samt deling av filer internt.

Confluence - For utforming av hjelpemidler som Gantt-skjema og andre hjelpemidler.

Jira - For utforming av "sprints" og for å holde oversikt over arbeidsoppgaver underveis. For eksempel: Hva gjenstår? Hva er underveis? Hva er gjort?

Scrum - Arbeidsmetodikk

Google Drive - For deling av filer internt.

Microsoft teams - For kommunikasjon med arbeidsgiver.

5.7.3 Utviklingshjelpemidler

PyCharm - Felles IDE for utvikling i Python

Azure - For tjenester som for eksempel hosting av servere.

Raspberry Pi - For sending av opptak av lydfiler, og åpning av dør

Mikrofon - For opptak av lyd, koblet med Raspberry-pi

Verktøy for bruk av AI (Tensorflow / PyTorch)

Databaseverktøy (sannsynligvis PostgreSQL)

5.7.4 Intern kontroll – evaluering

Som følge av bruk av Scrum-metodikken blir oppfølging av framdrift gjort annenhver uke med veiledere. Under stand-up møte på morgenene har medlemmer også mulighet til å evaluere tidsprosjektering av oppgaver man holder på med eller skal begynne på. Et mål er nådd når implementasjon er gjennomført og resultatet er pushet til git uten noen konflikter.

5.8 Beslutninger – beslutningsprosess

Under første møtet med Avento ble vi presentert en relativt åpen oppgave, men parter ble enige om å fokusere på talegjenkjenningsdelen for låssystem i høyere grad, og så heller se på integrering av tidligere prosjekt senere. Det skal også prøves å få selskapets låsleverandør til å åpne et API til bruk for direkte implementasjon i låsen.

Etter å ha hatt møte med veileder ble vi enige om at vi skulle prøve ut forskjellige modeller for maskinlæring og nevrale nettverk, og se om vi kunne oppnå gode resultater, slik at de kan brukes i låssystemet. Gjennom prosjektets gang vil det oppstå behov for å ta beslutninger, både de som er forutsatt nå men enda ikke tatt stilling til, samt andre uforutsette valg. Det vil bli gruppens ansvar, i fellesskap, å komme til avgjørelser for disse valgene. I de tilfellene der det vurderes nødvendig, vil gruppen konsultere med veileder først. Det forutsettes dog at gruppemedlemmene vil kunne utøve skjønn for å avgjøre mindre kritiske valg på egenhånd, i den grad det hjelper å fremskynde arbeidets gang.

6 DOKUMENTASJON

6.1 Rapporter og tekniske dokumenter

Følgende dokumentasjon skal utarbeides for å sørge for at alle parter er på samme side når det gjelder utforming av systemet generelt, samt deler av systemets enkelte komponenter:

- Kravspesifikasjon - For enighet om funksjonalitet som prosjektet inkluderer
- UI Mockups - For design av eventuelle brukergrensesnitt
- UML Diagram - For visning av hvordan systemet i sin helhet fungerer
- Test Plan - For å få en oversikt over hva det vil si at en test er suksessfull
- Deployment diagram – oversikt over de ulike komponentene i prosjektet
- Versjonsliste - Liste over hvilke versjoner av all programvare og utvidelser brukt

I tillegg vil denne forprosjektrapporten fungere som en initiell oppgavebeskrivelse, slik at alle involverte parter har referanse for hva prosjektet skal innebære, hvilke frister som skal overholdes, hvordan arbeidet skal foregå underveis, osv.

Gruppen vil også gjennomføre muntlig rapportering til veileder minimum en gang annenhver uke, eller skriftlig rapportering via mail i tilfeller der et fysisk møte ikke vil være gjennomførbart. En fast ramme for hvor ofte arbeidsgiver vil motta rapport, både skriftlig og muntlig, vil ikke settes. Rapportering til arbeidsgiver vil dog skje så ofte arbeidsgiver måtte ønske. Arbeidsgiver vil også bli informert om, og spurt om å godkjenne, alle valg som faller utenom innholdet som er beskrevet i denne rapporten.

Arbeidsgiver vil ikke ha mulighet for å distribuere ferdig løsning utenom innad i egen bedrift. Det forventes at arbeidsgiver selv vil stille med kompetanse for å vedlikeholde produktet (til den grad det vil være nødvendig) dersom det ønskes brukt.

7 PLANLAGTE MØTER OG RAPPORTER

7.1 Møter

7.1.1 Møter med styringsgruppen

Møte med veileder og biveileder foregår som regel annenhver torsdag fra 11.00 til 11.30 om ikke annet er avtalt.
Møte med arbeidsgiver avtales etter behov. Ellers vil rapportering skje gjennom *Microsoft Teams* eller muntlig når gruppens medlemmer måtte befinne seg i arbeidsgiverens lokaler.

7.1.2 Prosjektmøter

Siden prosjektet følger en scrum-metodikk har vi har stand-up møter før jobbing på prosjekt hver dag. Utover dette skal gruppen ha et mer omfattende møte etter hver sprint, der vi går mer i detalj om hva som har vært bra eller dårlig i denne perioden. Om ikke annet er planlagt, tar dette sted annenhver fredag fra 15.00-16.00, eller etter endt arbeidsdag fredag.

7.2 Periodiske rapporter

7.2.1 Framdriftsrapporter (inkl. milepæl)

Etter hver avsluttet sprint, altså annenhver uke, skal det legges frem en sprint-rapport som oppsummerer den forrige sprinten. I tillegg skal begge gruppens medlemmer kontinuerlig jobbe med en omfattende prosjektrapport, der prosjektets fremgangsmåter, informasjonssamling, resultater og videre drøfting, m.m., diskuteres i detalj.

8 PLANLAGT AVVIKSBEHANDLING

Dersom den foreliggende planen for prosjekt skal endres grunnet uforutsette hendelser, skal begge gruppemedlemmer være enig i endringen. Ved uenighet skal den foreliggende planen gjelde fram til begge parter kommer til enighet.

I tilfelle der gruppemedlemmene kommer til enighet om en endring som avviker fra den opprinnelige planen, vil en slik endring først diskuteres med veileder. Dersom gruppens medlemmer sammen med veileder, kommer fram til at den aktuelle endringen er fordelaktig, vil arbeidsgiver til slutt bli bedt om å gi en endelig godkjenning for endringen.

Ved en eventuell endring vil all dokumentasjon for prosjektet gjennomgås for å sikre at endringen blir gjenspeilet overalt der det er aktuelt. Dokumentasjonen vil også bli gjennomgått for å sikre at den foreslåtte endringen ikke har videre implikasjoner utenom det som ellers måtte være innlysende for gruppens medlemmer.

9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

For å gjennomføre prosjektet trenger gruppen en Raspberry Pi, som både skal sende inn lyd og eventuelt deretter åpne inngangsdøren til Avento. Det trengs ikke en god versjon av denne, men det er fordelaktig om wi-fi er integrert. Oppdragsgiver har allerede en enhet som vi i første omgang skal se på å bruke. Som tilleggsutstyr trenger vi også en mikrofon som kan ta opp lyd. Mikrofonen bør ta opp god nok lyd slik at modellen kan bli trent best mulig.

10 REFERANSER

Vedlegg 1: Kravspesifikasjon

Vedlegg 2: UML-diagram

Vedlegg 3: Deployment diagram

Vedlegg 4: Gantt diagram

Hvordan kan vi trekke ut data fra lydfiler for prosessering:

- https://reference.wolfram.com/language/tutorial/NeuralNetworksAudioAnalysis.html
- https://python-speech-features.readthedocs.io/en/latest/
- https://github.com/keunwoochoi/kapre/
- https://www.researchgate.net/publication/269295516_Speaker_verification_using_kernel-based_binary_classifiers_with_binary_operation_derived_features?fbclid=IwAR1Uagjb5zftHeg9XwXwDoqndS54R2aCdQlqqz_2TGZwRWGLAQg0nuDmk_A
- https://medium.com/@jonathan_hui/speech-recognition-feature-extraction-mfcc-plp-5455f5a69dd9

Eksisterende løsninger:

- https://azure.microsoft.com/nb-no/services/cognitive-services/
- https://cloud.google.com/speech-to-text/docs/multiple-voices

Database for forskjellige opptak av lyd:

- https://research.google.com/audioset/

Rammeverk:

- https://www.tensorflow.org/
- https://pytorch.org/

7.2 Vedlegg 2 - Kravspesifikasjon

Kravspesifikasjon

Målutgivelse	
Epic	
Dokumentstatus	UTKAST
Dokumenteier	Klaus Dyvik
Designer	
Utviklere	
Kvalitetssikring	

Mål

Lage en løsning som gjør det enkelt for Avento sine ansatte å få tilgang til arbeidslokalene gjennom talegjenkjenning.
Systemet må være sikkert, slik at de sikkerheten på arbeidsplassen ikke minsker med noen betydelig grad.
Systemet må også være enkelt å bruke, så det i prinsippet ikke bare tilbyr en ekstra angrepsoverflate, men også tjener en funksjon for de ansatte hos Avento.

Bakgrunn og strategisk passform

Avento AS ønsker å gjøre det enklere for de ansatte å få tilgang til kontorlokalene. De har foreslått en løsning der man tar i bruk talegjenkjenning, slik at ansatte kan åpne inngangsdøren ved bruk av stemme.
Det er viktig at systemet er enkelt å bruke, da de ansatte skal ha insentiv for å bruke denne teknologien fremfor eksisterende teknologi med RFID-chip og pin-kode.
Samtidig er det viktig at systemet gjøres sikkert, til den grad ikke-registrerte brukere ikke skal kunne få tilgang - både ved å bruke sin egen stemme, men også gjennom mer utspekulerte fremgangsmåter (som f.eks. å bruke et opptak av stemmen til en autorisert bruker).
Denne kravspesifikasjonen dekker alle behovene for et slikt system, alt fra implementasjon av selve teknologien for talegjenkjenning til systemer for administrative funksjoner og registrering.

Antakelser

- Vi antar at Avento er tålmodige og klar over at det kan være en periode der produktet blir kjørt som et testprosjekt og ikke er stabilt.
- Vi antar at ikke-registrerte brukere vil prøve å misbruke systemet for å få tilgang.

Funksjonelle krav

#	Tittel	Brukerhistorie	Viktighet	Notater	Mål
1	Autorisering ved tale	Brukere skal kunne bruke sin stemme for å bli autorisert, dersom brukeren er registrert i systemet.	Må ha	Viktig at dette gjelder for begge kjønn i alle aldre	Brukere av tjenesten skal kunne komme inn i bedriften sine lokaler med en lav falsk-negativ rate
2	Sikker sperre for ikke-registrerte brukere	Personer som ikke er registrert i systemet skal ikke ha mulighet til å bli autorisert.	Må ha		Falsk-positiv raten skal være nær null
3	Oversikt over bruk	Administrator skal ha oversikt over bruk av tjenesten	Må ha	Nettside	Administrator har en oversikt bruk av tjenesten og hvor godt den yter
4	Sperring av tjeneste	Administrator skal kunne skru av og på tjenesten	Fint å ha	På nettside for administrator	Tjenesten kan bli skrudd av over internett.
5	Registrering av ny bruker	Administrator skal kunne registrere nye brukere	Må ha	Evt. skal administrator kunne "invitere" brukere slik at de kan registrere seg selv.	Ny bruker kan gjøre #1 etter registrering.

#	Tittel	Brukerhistorie	Viktighet	Notater	Mål

1	Sikkerhet	Alle deler av systemet må være tilstrekkelig sikret for angrep fra personer med onde hensikter. Dette gjelder spesielt at "false-positive" raten skal være tilnærmet null, men også tilstrekkelig sikring av serveren og administrative funksjoner. Sikkerhetskritisk data som lagres i databaser eller lignende skal lagres i et format slik at det ikke er mulig å lese data selv med full tilgang til databasen.	Må ha	Det vil selvfølgelig ikke være mulig å teste og sikre mot alle mulige angrep - men ved å bruke allerede etablerte prinsipper for sikkerhet og behandling av brukerdata, vil dette hjelpe på lang vei i å holde de fleste angripere ute.	Tjenesten skal være tilstrekkelig sikret mot angrep fra brukere med onde hensikter.
2	Samtidighet og kapasitet	Tjenesten skal streve til å fungere for alle ansatte hos bedriften. Samtidig antas det at kun en person vil forsøke å låse seg inn med stemmegjenkjenning av gangen. Løsningen skal ta høyde for eventuelle økninger i antallet ansatte hos Avento.	Fint å ha	Selv om tjenesten skal prøve å fungere på mange personer kan det hende at dette ikke i bunn og grunn ikke er mulig, derav fint å ha.	Tjenesten skal erstatte allerede eksisterende løsninger hos bedriften.
3	Ytelse	Komponentene i systemet skal kommunisere raskt med hverandre og fungere problemfritt	Må ha	Vi ser til å utnytte siste teknologi for å sikre at systemet fungerer på best måte	Hele systemet skal være bygget opp på en ordentlig måte for å hindre konflikter og sikre god ytelse.
4	Pålitelighet	Tjenesten skal autorisere registrerte brukere og motsatt for ikke autoriserte brukere.	Må ha		Ingen brukere uten tilgang skal komme inn ved bruk av stemmen deres.
5	Vedlikehold	Systemet skal kunne kjøre av seg selv, men ingen (minimal) inngripen fra bedriften som tar i bruk systemet.	Må ha	Det tas høyde for at systemet kan ha feil som ikke har bitt seg merke i utviklingsfasen.	Systemet skal kun være en utvidelse av bedriftens sikkerhet, ikke en utvidelse av bedriftens ansvarsområder. Bedriten skal ikke trenge å gjøre videre utvikling av systemet etter at løsningen har blitt installert i bedriftens lokaler.
6	Brukervennlighet	Talegjenkjenningen i seg selv skal fungere raskt slik at det er hensiktsmessig å bruke tjenesten framfor andre autoriseringsmetoder (nøkkel etc.)	Må ha		Løsningen vil bli tatt i bruk av de ansatte hos bedriften, fremfor fortsatt bruk av eksisterende løsninger.
7	Dokumentering	Alle aspekter ved produktet skal være nøye dokumentert, slik at det er lagt godt til rette for videre utvikling dersom det er ønskelig. God dokumentasjon gjør det også enklere å vedlikeholde produktet ved behov.	Må ha		Produktet skal leveres med omfattende dokumentasjon.

Brukersamhandling og utforming

- UML- Diagram
- UI - Mockup
- Forprosjektrapport

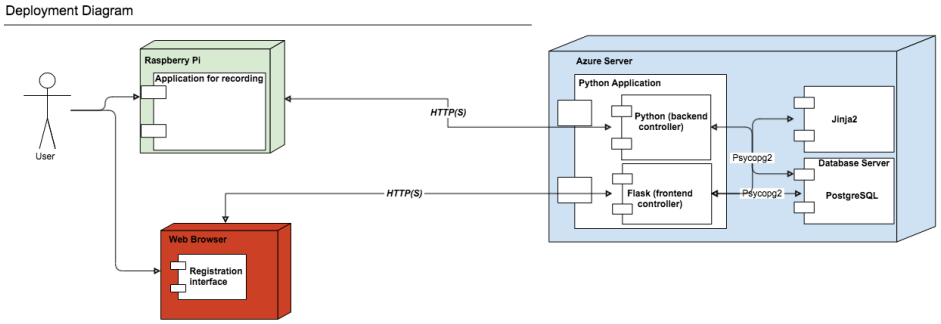
Spørsmål

Nedenfor er en liste over spørsmål som skal tas stilling til som følge av dette kravdokumentet:

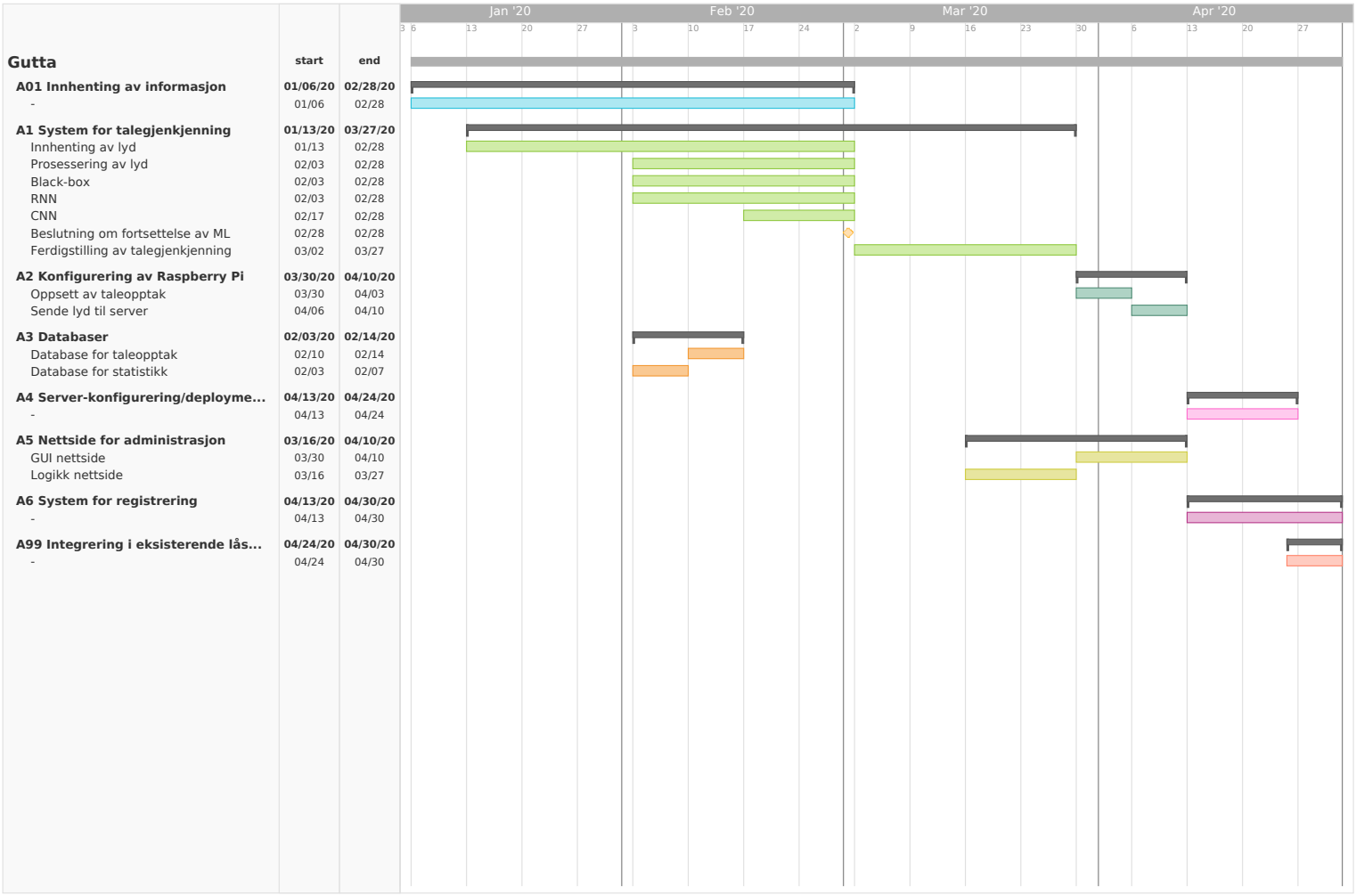
Spørsmål	**Utfall**
Hvor godt fungerer autorisering av personer?	Må trene maskin bedre for brukere som er registrert i systemet
Hvor godt klarer maskinen å detektere personer som ikke skal ha tilgang til tjenesten	Må trene maskin bedre for å skille personer som er registrert eller ikke
Klarer maskinen å skille mellom stemme og et taleopptak?	Enten gjøre endringer eller rådgive om å ikke bruke tjenesten utenfor bedriftens arbeidstider.
Fungerer maskinen like godt på alle aldre?	Trene maskinen for flere aldersgrupper
Fungerer maskinen like godt på begge kjønn?	Trene maskinen mer for det utsatte kjønn.

7.3 Vedlegg 3 - Deployment Diagram

Deployment Diagram



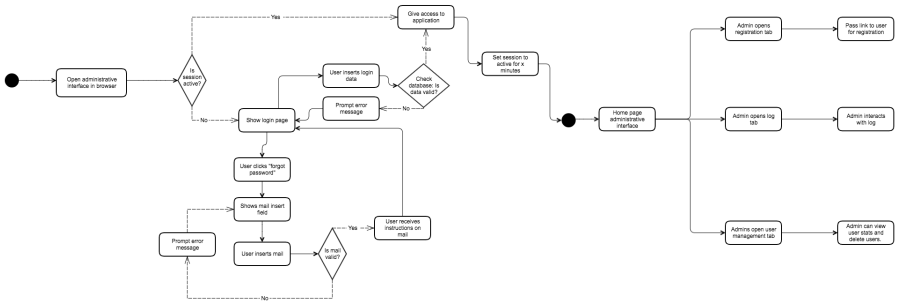
7.4 Vedlegg 4 - Gantt-Skjema

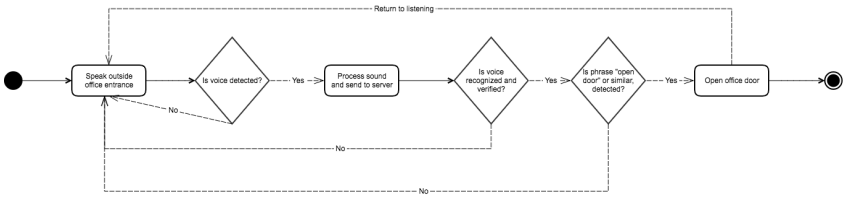


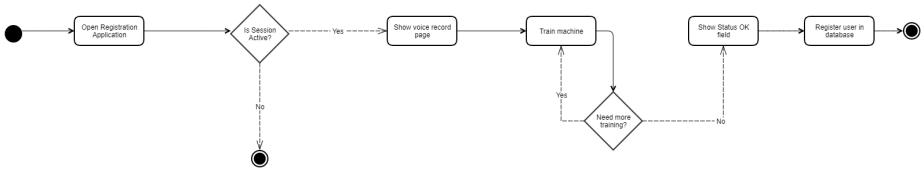
Gutta	start	end
A01 Innhenting av informasjon	**01/06/20**	**02/28/20**
-	01/06	02/28
A1 System for talegjenkjenning	**01/13/20**	**03/27/20**
Innhenting av lyd	01/13	02/28
Prosessering av lyd	02/03	02/28
Black-box	02/03	02/28
RNN	02/03	02/28
CNN	02/17	02/28
Beslutning om fortsettelse av ML	02/28	02/28
Ferdigstilling av talegjenkjenning	03/02	03/27
A2 Konfigurering av Raspberry Pi	**03/30/20**	**04/10/20**
Oppsett av taleopptak	03/30	04/03
Sende lyd til server	04/06	04/10
A3 Databaser	**02/03/20**	**02/14/20**
Database for taleopptak	02/10	02/14
Database for statistikk	02/03	02/07
A4 Server-konfigurering/deployme...	**04/13/20**	**04/24/20**
-	04/13	04/24
A5 Nettside for administrasjon	**03/16/20**	**04/10/20**
GUI nettside	03/30	04/10
Logikk nettside	03/16	03/27
A6 System for registrering	**04/13/20**	**04/30/20**
-	04/13	04/30
A99 Integrering i eksisterende lås...	**04/24/20**	**04/30/20**
-	04/24	04/30

7.5 Vedlegg 5 - UML Diagram

UML-Diagram


Open administrative interface in browser
Is session active?
Yes
No
Give access to application
Yes
User inserts login data
Check database: is data valid?
No
Prompt error message
Show login page
User clicks "forgot password"
Shows mail insert field
User inserts mail
Is mail valid?
Yes
No
User receives instructions on mail
Prompt error message
Set session to active for x minutes
Home page administrative interface
Admin opens registration tab
Pass link to user for registration
Admin opens log tab
Admin interacts with log
Admins open user management tab
Admin can view user stats and delete users.


Return to listening
Speak outside office entrance
Is voice detected?
Yes
No
Process sound and send to server
Is voice recognized and verified?
Yes
No
Is phrase "open door" or similar, detected?
Yes
No
Open office door


Open Registration Application
Is Session Active?
Yes
No
Show voice record page
Train machine
Need more training?
Yes
No
Show Status OK field
Register user in database

7.6 Vedlegg 6 - Sprintrapporter

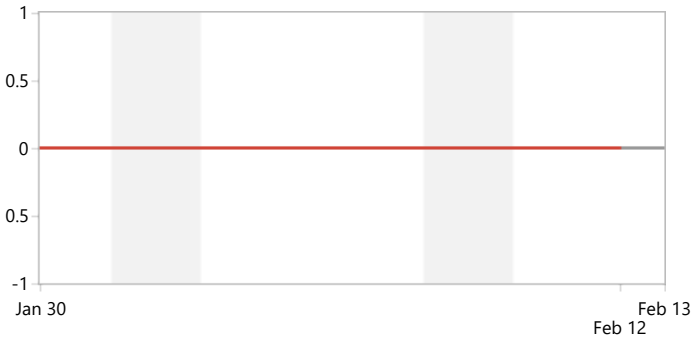
TAL board

Sprint Report Switch report

Sprint1

Closed Sprint, started by Fredrik Waaler, ended by Klaus Dyvik 30/Jan/20 9:20 AM - 12/Feb/20 10:06 AM View linked pages

Find information about CNN and RNN, learn basic info about signal-processing for sound - this will hopefully give us a solid foundation when we start b...



Status Report

* Issue added to sprint after start time

Completed Issues
View in Issue navigator

Key	Summary	Issue Type	Priority	Status	Story Points (-)
TAL-30 *	Database for taleopptak	Story	Major	DONE	-
TAL-31 *	Database for admin-side	Story	Major	DONE	-

Issues Not Completed
View in Issue navigator

Key	Summary	Issue Type	Priority	Status	Story Points (-)
TAL-24 *	Prosessering av lyd	Story	Major		-
TAL-26 *	RNN	Story	Major		-
TAL-27 *	CNN	Story	Major		-
TAL-33 *	Nettside for administrasjon	Story	Major		-

Issues Removed From Sprint
View in Issue navigator

Key	Summary	Issue Type	Priority	Status	Story Points (-)
TAL-23 *	Innhenting av lyd	Story	Major		-

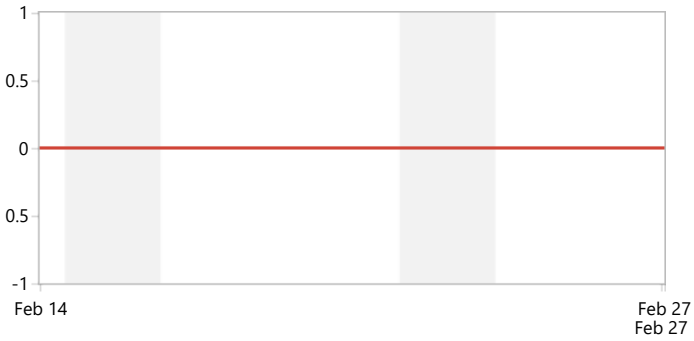
TAL board

Sprint Report Switch report

Sprint2

Closed Sprint, started by Fredrik Waaler, ended by Fredrik Waaler 14/Feb/20 11:19 AM - 27/Feb/20 12:47 PM View linked pages

Produce models that allows us to confidently asses whether we should keep creating our own ML-models.



Status Report

Completed Issues
View in Issue navigator

Key	Summary	Issue Type	Priority	Status	Story Points (-)
TAL-24	Prosessering av lyd	Story	Major	DONE	-

Issues Not Completed
View in Issue navigator

Key	Summary	Issue Type	Priority	Status	Story Points (-)
TAL-26	RNN	Story	Major		-
TAL-27	CNN	Story	Major		-

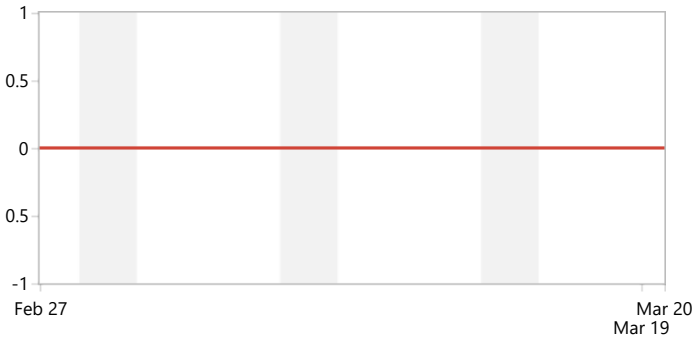
TAL board

Sprint Report Switch report

Sprint3

Closed Sprint, started by Klaus Dyvik, ended by Fredrik Waaler 27/Feb/20 2:58 PM - 20/Mar/20 9:08 AM View linked pages

To collect data from website and test it out on ML models, as well as start working on smaller things in paralell



Status Report

* Issue added to sprint after start time

Completed Issues
View in Issue navigator

Key	Summary	Issue Type	Priority	Status	Story Points (-)
TAL-98 *	Nettside for opptak	Story	Major	DONE	-

Issues Not Completed
View in Issue navigator

Key	Summary	Issue Type	Priority	Status	Story Points (-)
TAL-26 *	RNN	Story	Major		-
TAL-27 *	CNN	Story	Major		-
TAL-33 *	Nettside for administrasjon	Story	Major		-
TAL-34 *	System for registrering	Story	Major		-

Issues Removed From Sprint
View in Issue navigator

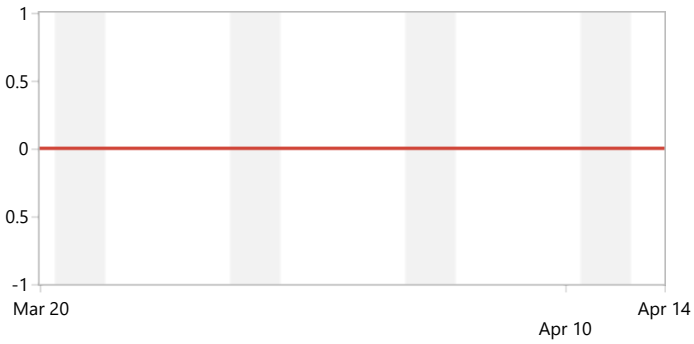
Key	Summary	Issue Type	Priority	Status	Story Points (-)
TAL-23 *	Innhenting av lyd	Story	Major		-

TAL board

Sprint Report Switch report

Sprint4

Closed Sprint, started by Fredrik Waaler, ended by Fredrik Waaler 20/Mar/20 9:16 AM - 14/Apr/20 8:13 AM View linked pages



Status Report

Issues Not Completed
View in Issue navigator

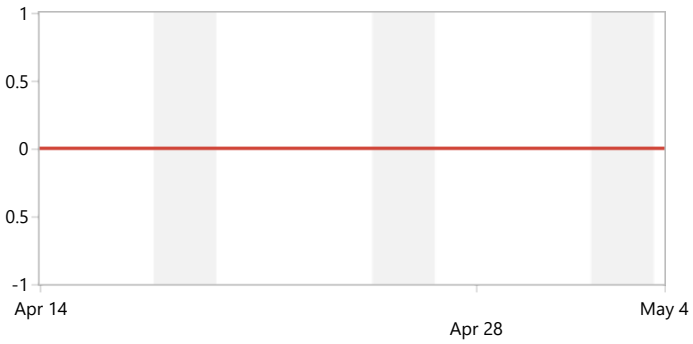
Key	Summary	Issue Type	Priority	Status	Story Points (-)
TAL-23	Innhenting av lyd	Story	Major		-
TAL-26	RNN	Story	Major		-
TAL-27	CNN	Story	Major		-
TAL-28	Oppsett av taleopptak	Story	Major		-
TAL-33	Nettside for administrasjon	Story	Major		-
TAL-34	System for registrering	Story	Major		-

TAL board

Sprint Report Switch report

Sprint5

Closed Sprint, started by Fredrik Waaler, ended by Fredrik Waaler 14/Apr/20 8:18 AM - 04/May/20 8:14 AM View linked pages



Status Report

Completed Issues
View in Issue navigator

Key	Summary	Issue Type	Priority	Status	Story Points (-)
TAL-27	CNN	Story	Major	DONE	-
TAL-34	System for registrering	Story	Major	DONE	-

Issues Not Completed
View in Issue navigator

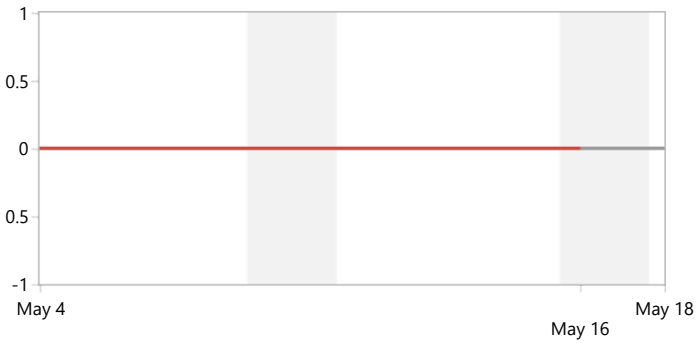
Key	Summary	Issue Type	Priority	Status	Story Points (-)
TAL-26	RNN	Story	Major		-
TAL-28	Oppsett av taleopptak	Story	Major		-
TAL-29	Sende lyd til server	Story	Major		-
TAL-32	Server deployment	Story	Major		-
TAL-33	Nettside for administrasjon	Story	Major		-

TAL board

Sprint Report Switch report

Sprint6

Closed Sprint, started by Fredrik Waaler, ended by Fredrik Waaler 04/May/20 8:18 AM - 16/May/20 11:10 AM View linked pages



Status Report

* Issue added to sprint after start time

Completed Issues
View in Issue navigator

Key	Summary	Issue Type	Priority	Status	Story Points (-)
TAL-23	Innhenting av lyd	Story	Major	DONE	-
TAL-26	RNN	Story	Major	DONE	-
TAL-28	Oppsett av taleopptak	Story	Major	DONE	-
TAL-29	Sende lyd til server	Story	Major	DONE	-
TAL-32	Server deployment	Story	Major	DONE	-
TAL-33	Nettside for administrasjon	Story	Major	DONE	-
TAL-35	Integrering i eksisterende låsesystem	Story	Major	DONE	-
TAL-57	Håndtering av autorisasjon	Story	Major	DONE	-
TAL-109	Sikre kommentering på all kode	Story	Major	DONE	-
TAL-110	Rapportskriving	Story	Major	DONE	-
TAL-116 *	Lage demovideo	Story	Major	DONE	-

7.7 Vedlegg 7 - SQL

```sql
-- -----------------------------------------------------
-- Schema AventoSR
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS AventoSR;
CREATE SCHEMA IF NOT EXISTS AventoSR;

-- -----------------------------------------------------
-- Table UserInfo
-- -----------------------------------------------------
DROP TABLE IF EXISTS UserInfo CASCADE;

CREATE TABLE IF NOT EXISTS UserInfo (
  email VARCHAR(255) PRIMARY KEY NOT NULL,
  name VARCHAR(255) NOT NULL,
  reg_date TIMESTAMP NOT NULL,
  valid_until TIMESTAMP NOT NULL
);

-- -----------------------------------------------------
-- Table AdminUsers
-- -----------------------------------------------------
DROP TABLE IF EXISTS AdminUsers CASCADE;

CREATE TABLE IF NOT EXISTS AdminUsers (
  email VARCHAR(255) PRIMARY KEY NOT NULL,
  password VARCHAR(255) NOT NULL
);


-- -----------------------------------------------------
-- Table Logging
-- -----------------------------------------------------
DROP TABLE IF EXISTS Logging CASCADE;
CREATE TABLE IF NOT EXISTS Logging (
  log_date TIMESTAMP NOT NULL,
  status INT NOT NULL,
  prob DECIMAL(9,8) NOT NULL
  constraint valid_status
      check (status <= 1),
  PRIMARY KEY (log_date)
);
```

```sql
-- ----------------------------------------------------------
-- Table RegistrationRecordings
-- ----------------------------------------------------------
DROP TABLE IF EXISTS RegistrationRecordings CASCADE;
CREATE TABLE IF NOT EXISTS RegistrationRecordings (
  email VARCHAR(255) NOT NULL,
  reg_recording VARCHAR NOT NULL,
  CONSTRAINT MAIL_FK
    FOREIGN KEY (email)
    REFERENCES UserInfo(email)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
;


DROP TYPE IF EXISTS TokenType CASCADE;
CREATE TYPE TokenType as ENUM('normal', 'admin', 'password');
-- ----------------------------------------------------------
-- Table RegistrationRecordings
-- ----------------------------------------------------------
DROP TABLE IF EXISTS Tokens CASCADE;
CREATE TABLE IF NOT EXISTS Tokens (
  token_value VARCHAR(255) NOT NULL,
  token_type TokenType NOT NULL,
  email VARCHAR(255) NOT NULL,
  expiry TIMESTAMP NOT NULL
);
```

```sql
--- Takes a email and a password and creates a record in the AdminUsers table.
DROP FUNCTION IF EXISTS add_admin;
CREATE OR REPLACE FUNCTION add_admin(p_email VARCHAR, password VARCHAR) RETURNS void AS
$BODY$
    INSERT INTO AdminUsers
    VALUES (p_email, password);
$BODY$ LANGUAGE sql;



--- Takes a admin-email and returns the related data of any such admin
DROP FUNCTION IF EXISTS get_admin;
CREATE OR REPLACE FUNCTION get_admin(p_email VARCHAR) RETURNS TABLE (email VARCHAR, password VARCHAR) AS
$BODY$
    SELECT
        *
    FROM AdminUsers
    WHERE email = p_email;
$BODY$
LANGUAGE sql;

--- Changes the password of the admin with the given email to new_password
DROP FUNCTION IF EXISTS change_admin_password;
CREATE OR REPLACE FUNCTION change_admin_password(p_email VARCHAR, new_password VARCHAR) RETURNS VOID AS
$BODY$
    UPDATE AdminUsers
    SET password = new_password
    WHERE email = p_email;
$BODY$
LANGUAGE sql;

--- Changes the administrators email
DROP FUNCTION IF EXISTS change_admin_email;
CREATE OR REPLACE FUNCTION change_admin_email(p_old_email VARCHAR, p_new_email VARCHAR) RETURNS VOID AS
$BODY$
    UPDATE AdminUsers
    SET email = p_new_email
    WHERE email = p_old_email;
$BODY$
LANGUAGE sql;

--- Deletes the admin-user by email
DROP FUNCTION IF EXISTS delete_admin;
CREATE OR REPLACE FUNCTION delete_admin(p_email VARCHAR) RETURNS VOID AS
$BODY$
    DELETE
    FROM AdminUsers
    WHERE email = p_email;
$BODY$
LANGUAGE sql;

--- Retrieves the validation of the token supplied
DROP FUNCTION IF EXISTS get_token_expiry;
CREATE OR REPLACE FUNCTION get_token_expiry(p_token_value VARCHAR, p_token_type TokenType) RETURNS TABLE(expiry TIMESTAMP) AS
$BODY$
    SELECT expiry FROM tokens
    WHERE token_type = p_token_type AND token_value = p_token_value;
$BODY$
LANGUAGE sql;


```

```sql
--- Takes an email, a name and a reg-date. Creates a record in the user-table.
DROP FUNCTION IF EXISTS add_user;
CREATE OR REPLACE FUNCTION add_user(p_email VARCHAR, p_name VARCHAR, p_reg_date TIMESTAMP, p_valid_until TIMESTAMP) RETURNS VOID AS
$BODY$
    INSERT INTO UserInfo
    VALUES (p_email, p_name, p_reg_date, p_valid_until);
$BODY$
LANGUAGE sql;


--- Retrieves a user-record by email
DROP FUNCTION IF EXISTS get_user;
CREATE OR REPLACE FUNCTION get_user(p_email VARCHAR) RETURNS TABLE (email VARCHAR, name VARCHAR, reg_date TIMESTAMP, valid_until TIMESTAMP) AS
$BODY$
    SELECT *
    FROM UserInfo
    WHERE email = p_email
    ORDER BY email;
$BODY$
LANGUAGE sql;


--- Retrieves detailed user info for all users (reg user-info + reg_status)
DROP FUNCTION IF EXISTS get_detailed_user_info;
CREATE OR REPLACE FUNCTION get_detailed_user_info() RETURNS TABLE (email VARCHAR, name VARCHAR, reg_date TIMESTAMP, reg_rec_count BIGINT, valid_until VARCHAR) AS
$BODY$
    SELECT email, name, reg_date, COALESCE(count, 0), CAST(valid_until AS VARCHAR) FROM (SELECT * FROM UserInfo UI
    LEFT JOIN (SELECT email as email3, COUNT(email) FROM RegistrationRecordings GROUP BY email ) AS entry_count ON entry_count.email3 = UI.email) as detailed_user_info ORDER BY 1
$BODY$
LANGUAGE sql;


--- Delete user-record with corresponding email
DROP FUNCTION IF EXISTS delete_user;
CREATE OR REPLACE FUNCTION delete_user(p_email VARCHAR) RETURNS VOID AS
$BODY$
    DELETE
    FROM UserInfo
    WHERE email = p_email;
$BODY$
LANGUAGE sql;


--- Creates a log-record with corresponding email, date-time, recording and status
DROP FUNCTION IF EXISTS add_log;
CREATE OR REPLACE FUNCTION add_log(p_status INTEGER, p_prob DECIMAL(9,8)) RETURNS VOID AS
$BODY$
    INSERT INTO Logging
    VALUES (now()::timestamp(0), p_status, p_prob);
$BODY$
LANGUAGE sql;


--- Deletes the log-record with corresponding timestamp
DROP FUNCTION IF EXISTS delete_log;
CREATE OR REPLACE FUNCTION delete_log(p_log_date TIMESTAMP) RETURNS VOID AS
$BODY$
    DELETE
    FROM Logging
    WHERE log_date = p_log_date;
$BODY$
LANGUAGE sql;


```

```
--- Retrievs all the data in the log
DROP FUNCTION IF EXISTS get_log;
CREATE OR REPLACE FUNCTION get_log() RETURNS TABLE(log_date TIMESTAMP, status INTEGER, prob DECIMAL(9,8)) AS
$BODY$
    SELECT *
    FROM Logging
    ORDER BY log_date;
$BODY$
LANGUAGE sql;


--- Creates a record in the RegistrationRecords table, with given email and recording
DROP FUNCTION IF EXISTS add_reg_record;
CREATE OR REPLACE FUNCTION add_reg_record(p_email VARCHAR, p_recording_path VARCHAR) RETURNS VOID AS
$BODY$
    INSERT INTO RegistrationRecordings
    VALUES (p_email, p_recording_path);
$BODY$
LANGUAGE sql;


--- Gets all registration recordings
DROP FUNCTION IF EXISTS get_reg_recordings;
CREATE OR REPLACE FUNCTION get_reg_recordings(p_email VARCHAR) RETURNS TABLE (email VARCHAR, reg_recording VARCHAR) AS
$BODY$
    SELECT *
    FROM RegistrationRecordings
    WHERE email = p_email;
$BODY$
LANGUAGE sql;

--- Checks if the user has associated registration recordings
DROP FUNCTION IF EXISTS has_reg_records;
CREATE OR REPLACE FUNCTION has_reg_records(p_email VARCHAR) RETURNS TABLE (value BOOLEAN) AS
$BODY$
    SELECT (CASE  WHEN reg_count >= 1 THEN True ELSE False END)
    FROM (SELECT COUNT(email) as reg_count, email FROM registrationrecordings GROUP BY 2) as "registrations_counted"
    WHERE email = p_email;
$BODY$
LANGUAGE sql;

--- Deletes any registration-recordings associated with the given user
DROP FUNCTION IF EXISTS delete_reg_records_for_user;
CREATE OR REPLACE FUNCTION delete_reg_records_for_user(p_email VARCHAR) RETURNS VOID AS
$BODY$
    DELETE FROM RegistrationRecordings
    WHERE email = p_email;
$BODY$
LANGUAGE sql;


--- Inserts the specified token with the specified type in the token table
DROP FUNCTION IF EXISTS add_token;
CREATE OR REPLACE FUNCTION add_token(p_token_value VARCHAR, p_token_type TokenType, p_email VARCHAR, p_expires TIMESTAMP) RETURNS VOID AS
$BODY$
    --- Delete any such token before issuing a new one
    DELETE FROM Tokens WHERE email = p_email AND token_type = p_token_type;
    --- Then issue a new one
    INSERT INTO Tokens
    VALUES (p_token_value, p_token_type, p_email, p_expires);
$BODY$
LANGUAGE sql;

```

```sql
--- Retrieves the token with the specified value
DROP FUNCTION IF EXISTS get_token_by_value_and_type;
CREATE OR REPLACE FUNCTION get_token_by_value_and_type(p_token_value VARCHAR, p_token_type TokenType) RETURNS TABLE(token_value VARCHAR, token_type TokenType, email VARCHAR, expiry TIMESTAMP) AS
$BODY$
    SELECT *
    FROM Tokens
    WHERE token_value = p_token_value AND token_type = p_token_type;
$BODY$
LANGUAGE sql;


--- Retrieves the email that belongs to the token of given type and value
DROP FUNCTION IF EXISTS get_email_by_token_and_type;
CREATE OR REPLACE FUNCTION get_email_by_token_and_type(p_token_value VARCHAR, p_token_type TokenType) RETURNS TABLE(email VARCHAR) AS
$BODY$
    SELECT email
    FROM Tokens
    WHERE token_value = p_token_value AND token_type = p_token_type;
$BODY$
LANGUAGE sql;

--- Checks if the user has any active tokens of the given type
DROP FUNCTION IF EXISTS get_token_by_mail_and_and_type;
CREATE OR REPLACE FUNCTION get_token_by_mail_and_and_type(p_token_type TokenType, p_email VARCHAR)
RETURNS TABLE(token_value VARCHAR, token_type TokenType, email VARCHAR, expiry TIMESTAMP) AS
$BODY$
    SELECT *
    FROM Tokens
    WHERE email = p_email AND token_type = p_token_type;
$BODY$
LANGUAGE sql;

--- Deletes the token of given value and given type
DROP FUNCTION IF EXISTS delete_token_by_value_and_type;
CREATE OR REPLACE FUNCTION delete_token_by_value_and_type(p_token_value VARCHAR, p_token_type TokenType) RETURNS VOID AS
$BODY$
    DELETE FROM Tokens
    WHERE token_type = p_token_type AND token_value = p_token_value;
$BODY$
LANGUAGE sql;

--- Gets the valid TokenTypes
DROP FUNCTION IF EXISTS get_token_types;
CREATE OR REPLACE FUNCTION get_token_types() RETURNS TABLE (token_values VARCHAR) AS
$BODY$
    SELECT unnest(enum_range(NULL::TokenType))::text as token_values;
$BODY$
LANGUAGE sql;


--- Retrieves the email of users from UserInfo that has expired
DROP FUNCTION IF EXISTS get_expired_users;
CREATE OR REPLACE FUNCTION get_expired_users() RETURNS TABLE (email VARCHAR) AS
$BODY$
    SELECT email FROM UserInfo WHERE valid_until < NOW();
$BODY$
LANGUAGE sql;

```

7.8 Vedlegg 8 - Readme for server

Redme Server Deployment

Install psql and postgres by following this tutorial:
https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-18-04

- Run sudo -u postgres psql to access the psql shell
- Create supplied SQL tables
- Create supplied SQL functions
- Add admin user to use (Use PasswordHandler.py to encrypt password before insert)

Follow tutorial https://www.digitalocean.com/community/tutorials/how-to-deploy-a-flask-application-on-an-ubuntu-vps
With exception:

- Run "sudo apt-get install libapache2-mod-wsgi python3-dev" instead of "sudo apt-get install libapache2-mod-wsgi python-dev", as the project needs Python3.
- Install pip3, by running "sudo apt-get install python3-pip" instead of "sudo apt-get install python-pip".
- Change default python and pip for system to python3 and pip3 by running:
  sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.6 1
  sudo update-alternatives --install /usr/bin/pip pip /usr/bin/pip3
- Install the virtualenv without sudo, because mod_wsgi will run without sudo permission.
- Clone git-repo "https://github.com/fredrikwaaler/AdminWebsite» into /var/www/<ProjectName>/<ProjectName>
- Use pip install –r requirements.txt to install all python dependencies
- Add the settings.py file in /var/www/<ProjectName>/<ProjectName> with the following values:
  - EMAIL
  - EMAIL_PASS
  - DB_HOST
  - DB_USER
  - DB_DATABASE
  - DB_PASSWORD
  - BASE_URL
  - WAVS_FOLDER
- Run sudo apt-get install libpq-dev as this is needed for librosa
- Change the paths at the following places:
  - Settings.py path in CfgHandler.py
  - Google resources key path in SpeechToText.py
  - Templates path in Mailer
  - FTPFolder path on "do_prediction" in __init__
  - Noise folder path and AudioBase path in AudioHandler
- Change the settings for AudioBase, FTPFolder and Settings.py, so that the application is allowed to read and write to these locations by running "sudo chmod 777 <PathToFolders/Files>"

Install Certbot to enable HTTPS, and configure for the application by following this tutorial:
https://certbot.eff.org/lets-encrypt/ubuntubionic-apache

When certbot is installed, add the following to the first to lines of the Apache config file (/etc/apache2/<ProjectName>.conf):

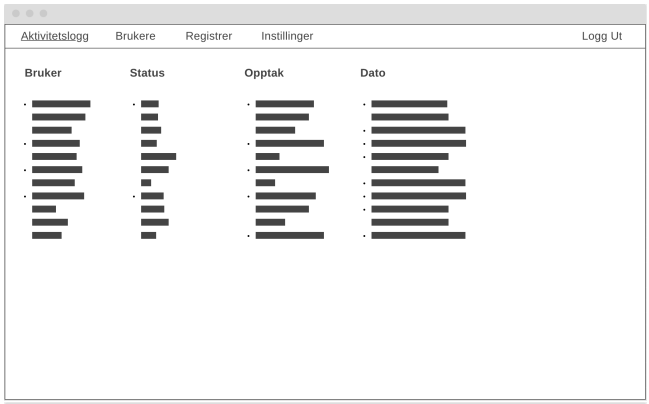
- WSGIPythonHome /var/www/<PathToVirtualEnv>
- WSGIPythonPath /var/www/<PathToFolderWith__init.py>

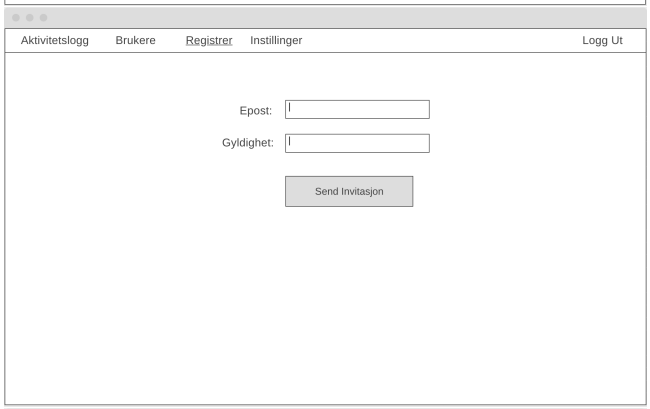
Add the same to SSL-config file at same location.

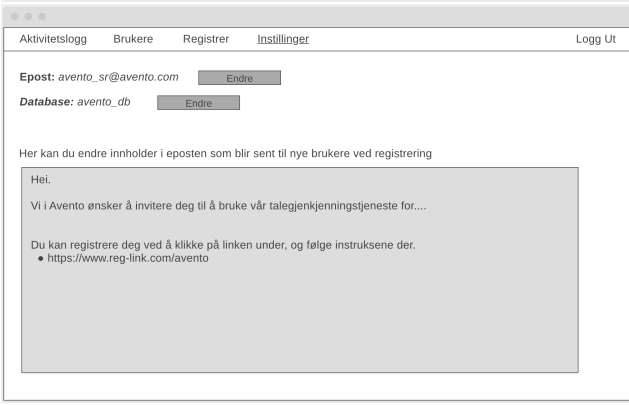
This will tell mod_wsgi where to look for the project files, and what python interpreter to use.

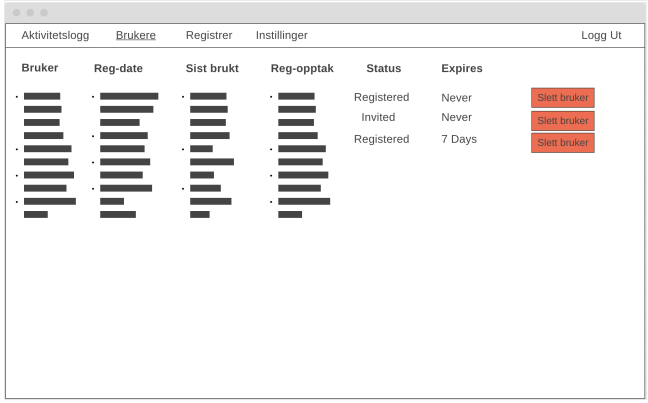
Lastly, run “sudo service apache2 restart”. Application should be running on server.

7.9 Vedlegg 9 - Wireframes









7.10 Vedlegg 10 - Flowchart Github

VoiceRecogniton

master origin/master origin/HEAD Cleaned Voice Recognition part	4 May 2020 12:47	kdyvik <klaus.dyvi	3dd9df90
merge	28 Apr 2020 9:59	kdyvik <klaus.dyvi	8845b440
merge	28 Apr 2020 9:56	kdyvik <klaus.dyvi	76d051bc
Added outline for system on Raspberry Pi	28 Apr 2020 9:54	kdyvik <klaus.dyvi	62aca187
Added requirements.txt file	27 Apr 2020 11:23	fredrikwaaler <frec	570612fc
Fixed merge conflict	27 Apr 2020 11:11	fredrikwaaler <frec	4ed024d9
Added commentary	27 Apr 2020 11:07	fredrikwaaler <frec	a780de9e
ResNet transfer learning ++	21 Apr 2020 15:43	kdyvik <klaus.dyvi	6bcf93ee
Merge branch 'master' of https://github.com/fredrikwaaler/Gutta	14 Apr 2020 12:49	kdyvik <klaus.dyvi	2020777d
current working dir after testing different datasets	14 Apr 2020 12:48	kdyvik <klaus.dyvi	a7c107e7
Admin web page logic done for now. Except tts check, ml-retrain on reg and email content edit	26 Mar 2020 13:18	fredrikwaaler <frec	7face5d4
Hallo	24 Mar 2020 12:38	fredrikwaaler <frec	332c0df8
Added app_handler, configparser, user & admin reg on website, updated log, settings page (not finished),	24 Mar 2020 12:36	fredrikwaaler <frec	8bed865c
Delete settings.py	19 Mar 2020 14:19	fredrikwaaler <368	4feb54de
Updated users-page to reflect changes discussed in meeting	18 Mar 2020 16:03	fredrikwaaler <frec	bb4e5f99
merge conflict	18 Mar 2020 15:34	kdyvik <klaus.dyvi	656390c9
temp	18 Mar 2020 15:33	kdyvik <klaus.dyvi	18145ed6
Merge branch 'master' of https://github.com/fredrikwaaler/Gutta	18 Mar 2020 15:31	fredrikwaaler <frec	8a9319bf
Fixed problem on noise generation where noise was to long	18 Mar 2020 15:31	fredrikwaaler <frec	25f49400
Added visualization file	18 Mar 2020 15:03	kdyvik <klaus.dyvi	94faf8c8
Attempted fix on AudioEq so that audio starts at random location	18 Mar 2020 14:54	fredrikwaaler <frec	d244288c
Merge conflicts	18 Mar 2020 10:55	kdyvik <klaus.dyvi	83aaf936
Added more automation in model process	18 Mar 2020 10:37	kdyvik <klaus.dyvi	79bf9bd0
Merge fix	18 Mar 2020 10:15	fredrikwaaler <frec	3bfa9f29
Some changes	18 Mar 2020 10:14	fredrikwaaler <frec	6f94e76d
added sound files to gitignore	17 Mar 2020 10:21	kdyvik <klaus.dyvi	abaab03c
cleaned	27 Feb 2020 14:11	kdyvik <klaus.dyvi	cb6b8ab'
Added folder to Microphone App	27 Feb 2020 14:05	kdyvik <klaus.dyvi	8d25a985
Created folder for microphone app	27 Feb 2020 14:00	kdyvik <klaus.dyvi	fc8a07df
merge conflicts	26 Feb 2020 12:27	kdyvik <klaus.dyvi	b69ed6e9
some minor testing issues	26 Feb 2020 12:23	kdyvik <klaus.dyvi	00ba9f74
Normalization and confusion matrix	26 Feb 2020 11:52	fredrikwaaler <frec	30a8f0da
C	21 Feb 2020 12:52	fredrikwaaler <frec	1cb08fd5
MergeConflict because of local changes, not important	21 Feb 2020 12:47	fredrikwaaler <frec	2ef8ac72
removed ap	21 Feb 2020 12:39	kdyvik <klaus.dyvi	ba2aa82f
delete ap	21 Feb 2020 12:24	kdyvik <klaus.dyvi	162c254d
Merge branch 'master' of https://github.com/fredrikwaaler/Gutta	21 Feb 2020 11:52	kdyvik <klaus.dyvi	e3235051
Added new data in 'VoiceRecogntion/PredictionData'	20 Feb 2020 11:33	fredrikwaaler <frec	de7436ac
Refinments and dataset	20 Feb 2020 10:06	kdyvik <klaus.dyvi	8b23b27c
Merge branch 'master' of https://github.com/fredrikwaaler/Gutta	19 Feb 2020 15:27	kdyvik <klaus.dyvi	af9b2b33
test	19 Feb 2020 15:27	kdyvik <klaus.dyvi	14a24a55
Added drawer for confusion matrixes, and tons of noises in resources	19 Feb 2020 15:06	fredrikwaaler <frec	56736f00
Merge branch 'master' of https://github.com/fredrikwaaler/Gutta	19 Feb 2020 13:11	kdyvik <klaus.dyvi	2c100d65
er	19 Feb 2020 13:11	kdyvik <klaus.dyvi	cebb784c
Added extra noise-generated files for original recordings	19 Feb 2020 12:46	fredrikwaaler <frec	0c142c6b
Automated noise generation for datasets, + automatic labeling for datasets	18 Feb 2020 12:00	fredrikwaaler <frec	fbcde8df
Finished logic for admin-page. Included mailer for registration - content of mail need editing	18 Feb 2020 11:58	fredrikwaaler <frec	23fef6a9
added noise wav's	14 Feb 2020 14:04	kdyvik <klaus.dyvi	7081c3ef
Separated command detection from audio recorder and cleaned up	14 Feb 2020 13:25	kdyvik <klaus.dyvi	c0d27873
Audio features and simple cnn	14 Feb 2020 9:24	kdyvik <klaus.dyvi	d24cbfe9
Started with logic for admin pages - login, manage-userbase, log	5 Feb 2020 12:55	fredrikwaaler <frec	2a1f9f50
this is a test to make sure everything works	27 Jan 2020 14:49	kdyvik <klaus.dyvi	01f3204b
first commit	27 Jan 2020 14:41	fredrikwaaler <frec	3b67787f

AdminWebsite

master origin/master origin/HEAD Added styling for mobile use	15 May 2020 15:15	kdyvik <klaus.dyvi	691c288
added loader for user reg. lacks styling	15 May 2020 14:39	kdyvik <klaus.dyvi	d200437
Merge branch 'master' of https://github.com/fredrikwaaler/AdminWebsite	14 May 2020 12:47	kdyvik <klaus.dyvi	846cb4a
Removed ID from AH	14 May 2020 12:47	kdyvik <klaus.dyvi	5e3d8cf
Removed test code from AudioHandler	14 May 2020 12:23	fredrikwaaler <frec	dcc6b03
Fixed ah issue	14 May 2020 12:21	kdyvik <klaus.dyvi	e837cce
Fixed win_length issue	14 May 2020 12:18	kdyvik <klaus.dyvi	712635d
Fixed ah issue	14 May 2020 12:16	kdyvik <klaus.dyvi	b451396
Restricted log entries to seconds instead of millis	14 May 2020 12:13	fredrikwaaler <frec	78833cf
Removed win_length and 44100 Hz mic restriction	14 May 2020 11:27	fredrikwaaler <frec	4bda66e
Removed email from log-table as it is not returned on predictions	14 May 2020 11:26	fredrikwaaler <frec	b1115d1
Changed __init__ to accomodate path-storing of reg recordings	12 May 2020 10:24	fredrikwaaler <frec	83154c5
merge ah	12 May 2020 10:13	kdyvik <klaus.dyvi	0a5bc85
Added merge noise	12 May 2020 10:11	kdyvik <klaus.dyvi	d30190d
__init__ adapted for file-path specific voice-rec	8 May 2020 15:23	fredrikwaaler <frec	0ea87f7
Updated __init__ to contain VOICE_REC flag	6 May 2020 11:35	fredrikwaaler <frec	f022672
Moved guttakeys.json to VoiceRecognition	5 May 2020 15:34	fredrikwaaler <frec	cf31383
Updated commentary for all files	5 May 2020 15:22	fredrikwaaler <frec	4645546
Merge branch 'master' of https://github.com/fredrikwaaler/AdminWebsite	5 May 2020 13:43	fredrikwaaler <frec	0d3196c
Added support for merging audio with noise files	5 May 2020 11:21	kdyvik <klaus.dyvi	5961fca
Implemented Voice Recognition software	4 May 2020 11:45	kdyvik <klaus.dyvi	e4f62f3
Merge branch 'master' of https://github.com/fredrikwaaler/AdminWebsite	30 Apr 2020 10:25	fredrikwaaler <frec	b2ef7eb
Added requirements and google keys	30 Apr 2020 10:25	fredrikwaaler <frec	4db68b8
added log entry route	30 Apr 2020 10:09	kdyvik <klaus.dyvi	bfa5dac
Added Raspberry Pi variables in settings	29 Apr 2020 13:50	kdyvik <klaus.dyvi	105c2e0
Added support to send files over FTP to Raspberry Pi	29 Apr 2020 12:19	kdyvik <klaus.dyvi	ca6ac3c
utkast administratornettside	28 Apr 2020 14:16	kdyvik <klaus.dyvi	7e5b4ae
First Commit	27 Apr 2020 11:36	fredrikwaaler <frec	42bf7ca

MicrophoneApp

origin/master origin/HEAD Halla	5 Mar 2020 11:16	fredrikwaaler <frec	83fd398
Added recorder2 as main route	5 Mar 2020 11:02	fredrikwaaler <frec	e6d887a
Merge branch 'master' of https://github.com/fredrikwaaler/MicrophoneApp	5 Mar 2020 10:49	fredrikwaaler <frec	ba11fc3
Attempted fix for ajax on mobile by nocache	5 Mar 2020 10:49	fredrikwaaler <frec	9e18287
master 4↓ Merge branch 'master' of https://github.com/fredrikwaaler/MicrophoneAp	5 Mar 2020 10:35	kdyvik <klaus.dyvi	ce897fa
fixed blob issue	5 Mar 2020 10:35	kdyvik <klaus.dyvi	fc8ed52
Fixed defect ajax request for recorder2	5 Mar 2020 10:24	fredrikwaaler <frec	37328ea
Added route for recorder2	5 Mar 2020 10:08	fredrikwaaler <frec	b64c6b5
Revert again	4 Mar 2020 20:02	fredrikwaaler <frec	bc96eb2
Revert to recorder1 for test purposes	4 Mar 2020 19:56	fredrikwaaler <frec	0b9aee4
b	4 Mar 2020 19:06	fredrikwaaler <frec	3283000
Bug Fix	4 Mar 2020 18:37	fredrikwaaler <frec	25d2df0
Bug. Created recorder in stopRecording, now moved to startRecording	4 Mar 2020 18:18	fredrikwaaler <frec	91744a1
aa	4 Mar 2020 18:16	fredrikwaaler <frec	4a8b512
Bug fix	4 Mar 2020 18:13	fredrikwaaler <frec	56bb176
Attempted fix for lag on recoring by making usermedia global	4 Mar 2020 18:09	fredrikwaaler <frec	ecac7a9
Bug fix	4 Mar 2020 17:59	fredrikwaaler <frec	160a8cc
Bug fix to toggle start and stop buttons	4 Mar 2020 17:54	fredrikwaaler <frec	147d2a8
Removed blocker for apple devices	4 Mar 2020 17:22	fredrikwaaler <frec	c87fcd3
Removed blocker for apple devices	4 Mar 2020 17:22	fredrikwaaler <frec	d60fdfd
Added support for iOS	4 Mar 2020 17:19	fredrikwaaler <frec	8afde5c
removed flask-limiter	4 Mar 2020 16:07	kdyvik <klaus.dyvi	e45bafa
added app.py	4 Mar 2020 16:05	kdyvik <klaus.dyvi	30b7512
Experimental recorder.js features	4 Mar 2020 16:03	kdyvik <klaus.dyvi	0a76262
css for unsupported device	4 Mar 2020 14:49	kdyvik <klaus.dyvi	4ed7843
Merge branch 'master' of https://github.com/fredrikwaaler/MicrophoneApp	4 Mar 2020 14:39	kdyvik <klaus.dyvi	e88c501
new message for user	4 Mar 2020 14:39	kdyvik <klaus.dyvi	22d6b8b
Merge fix	4 Mar 2020 14:35	fredrikwaaler <frec	16fc0f2
Incorporated ip into file-identidier	4 Mar 2020 14:33	fredrikwaaler <frec	656e10d
Added check for iOS devices	4 Mar 2020 14:15	kdyvik <klaus.dyvi	671dfa8
Merge branch 'master' of https://github.com/fredrikwaaler/MicrophoneApp	4 Mar 2020 11:10	fredrikwaaler <frec	b6ab6aa
Specified path to /var/www/...	4 Mar 2020 10:34	fredrikwaaler <frec	fed1e4b
Dont ask for mic access before last page	3 Mar 2020 15:32	fredrikwaaler <frec	23355ef
A	3 Mar 2020 12:50	fredrikwaaler <frec	151f018
Added ./ to log file	3 Mar 2020 12:24	fredrikwaaler <frec	662c226
changed app to init, added try-except for logging	3 Mar 2020 12:21	fredrikwaaler <frec	5aca405
Merge branch 'master' of https://github.com/fredrikwaaler/MicrophoneApp	3 Mar 2020 12:10	fredrikwaaler <frec	aadbd55
Added folder creator for oggs and wavs	3 Mar 2020 12:10	fredrikwaaler <frec	b53322f
fixed shading issues	2 Mar 2020 15:09	kdyvik <klaus.dyvi	a126360
fixed check for input to text field	2 Mar 2020 15:04	kdyvik <klaus.dyvi	1185425
Website prototype finished	2 Mar 2020 14:18	kdyvik <klaus.dyvi	b35fbe2
Split css and js into separate files	2 Mar 2020 12:43	kdyvik <klaus.dyvi	d4a3ccb
merge conflicts	2 Mar 2020 12:22	kdyvik <klaus.dyvi	97b6c17
some extra css	2 Mar 2020 12:19	kdyvik <klaus.dyvi	3673524
Final js for logic	2 Mar 2020 12:18	fredrikwaaler <frec	d202e50
Opptakswidget complete	28 Feb 2020 16:03	fredrikwaaler <frec	958e008
Skeleton of website	28 Feb 2020 14:45	kdyvik <klaus.dyvi	04cad6e
Skeleton of website	28 Feb 2020 14:43	kdyvik <klaus.dyvi	e85388f
Can now do three recordings, sent to server and stored	28 Feb 2020 14:41	fredrikwaaler <frec	4738bed
Added input fields to recorder.html	27 Feb 2020 14:26	fredrikwaaler <frec	224d34a
Init app	27 Feb 2020 14:22	fredrikwaaler <frec	fe2d476
Init repository	27 Feb 2020 14:13	fredrikwaaler <frec	1a41687

RPiSystem

master origin/master Changed sample rate	15 May 2020 11:41	kdyvik <klaus.dyvi	67e7e8b
Added microphone tester 3	14 May 2020 19:52	kdyvik <klaus.dyvi	92fb3e9
Added microphone tester 2	14 May 2020 19:51	kdyvik <klaus.dyvi	1b3fb43
Added microphone tester	14 May 2020 19:47	kdyvik <klaus.dyvi	5bdf743
Added microphone tester	14 May 2020 19:37	kdyvik <klaus.dyvi	9aed8f8
Added setup to test all diode states	8 May 2020 11:01	kdyvik <klaus.dyvi	62d935a
Commit to test green diode output	8 May 2020 10:58	kdyvik <klaus.dyvi	072f152
Commit to test green diode output	8 May 2020 10:54	kdyvik <klaus.dyvi	cf89969
Commit to test green diode output	8 May 2020 10:53	kdyvik <klaus.dyvi	b2f5922
Commit to test green diode output	8 May 2020 10:51	kdyvik <klaus.dyvi	0ce0fb8
Commit to test green diode outpu	8 May 2020 10:50	kdyvik <klaus.dyvi	e1ebdb9
Commit to test green diode output	8 May 2020 10:46	kdyvik <klaus.dyvi	2c6d039
Commit to test green diode output	8 May 2020 10:45	kdyvik <klaus.dyvi	0ba8dd9
Commit to test green diode output	8 May 2020 10:36	kdyvik <klaus.dyvi	b5fa34e
fixed sftp	4 May 2020 13:42	kdyvik <klaus.dyvi	52230c1
RPi-system without IO and working FTP	4 May 2020 11:20	kdyvik <klaus.dyvi	5423702
removed reference to visualizer	29 Apr 2020 13:44	kdyvik <klaus.dyvi	f4ec920
Removed references to package	29 Apr 2020 13:44	kdyvik <klaus.dyvi	f352b04
Added requirements.txt	29 Apr 2020 13:09	kdyvik <klaus.dyvi	0964acf
Added FTPfolder to gitignore	29 Apr 2020 11:31	kdyvik <klaus.dyvi	1067646
Removed FTP folder init file	29 Apr 2020 11:29	kdyvik <klaus.dyvi	c870964
Added folder for ftp	29 Apr 2020 11:29	kdyvik <klaus.dyvi	d12fc37
Added the rest of the system	28 Apr 2020 15:02	kdyvik <klaus.dyvi	8579e87
Added gitignore	28 Apr 2020 15:02	kdyvik <klaus.dyvi	e13af43

Fredrik Ferdinand Waaler, Klaus Dyvik

Talegjenkjenning for låsesystem

NTNU
Kunnskap for en bedre verden