

2. Semester, It-teknologi

Projekt Tryghedssystem

Gruppe 2

Forfattere:

Simon H. Kristensen (SHK)

Henrik B. Bruun (HB)

Stian Ø. Falck (SØF)

Henrik Jolin-Laursen (HJL)

Vejledere: Kaj Norman Nielsen, Lasse Kaae og Jonas Wehding

Indsendelsesdato: 06-12-2024

Antal tegn: 92.740

Resumé

Mange ældre oplever utryghed i hjemmet, især på grund af tricktyverier eller stalking, hvilket kan føre til social isolation. For at adressere denne udfordring har vi udviklet en prototype på et tryghedssystem, der anvender teknologi for at øge sikkerheden i disse situationer.

Systemet inkluderer et kamera med ansigtsdetektion og genkendelse, der automatisk identificerer besøgende. Optagelser af ikke genkendte personer, lagres som MP4-filer lokalt og kan tilgås via en webserver, hvilket gør det muligt for brugere at overvåge besøg, som dokumentation.

Vi har udarbejdet et detaljeret Gantt-skema og en WBS for at strukturere projektet og have et overblik over tidsstyringen. Derudover har vi idéer til fremtidige forbedringer, herunder implementering af tovejs-lyd- og videokommunikation gennem webrtc, som vil gøre det muligt at interagere med besøgende uden at åbne døren.

Vores projekt er gennemført med en induktiv tilgang, hvor data og tests har bidraget til løbende forbedringer. Den nuværende prototype lægger fundamentet for et komplet tryghedssystem, der vil øge sikkerheden for ældre eller udsatte borgere.

Indholdsfortegnelse

Resumé	2
Indholdsfortegnelse	3
1. Indledning (HB)	7
2. Metoder og værktøjer (HJL).....	8
2.1 Induktiv metodisk tilgang under projektet. (HJL)	8
2.2 Værktøjer (WBS, Gantt, kodningsprogrammer mm.) (HJL)	8
3. Virksomheden (HB)	9
5. Moduler	11
5.0 RPI 4B (HJL).....	11
5.0.1 - Overordnet moduldiagram (HJL, SØF)	13
5.1 Video og Ansigtsgenkendelse (HB)	14
5.1.1 Modul Design	14
5.1.2 Modul implementering	21
5.1.3 Formel modultest	23
5.2 HC-SR501 PIR Sensor modul (HJL).....	25
5.2.1 Python PIR Modul (HJL)	26
5.2.2 Modul implementering (HJL)	27
5.2.3 Formel modultest (HJL)	27
5.3 Database & Web (SHK, HJL)	31
5.3.1 Modul Design	32
5.3.2 Modul implementering	33
5.3.3 Formel modultest	36
5.4 Main & Mastersoftware (SHK, HB, HJL)	36
5.4.1 Modul Formål og Design	36
5.4.2 Modul implementering	37
5.4.3 Formel modultest	38
5.4.4 Videreudvikling	39
5.5 Lyd optagelse og afspilning – ‘audio_handling’ (SØF)	39
5.5.1 Modul design	40
5.5.2 Formel modultest	45
5.5.3.1 - Opsætning af default lydenheder og test af optagelse gennem terminal...45	45

5.5.3.2 - test af PyAudio til optagelse og lagring af lyd.....	47
5.5.4 Refleksioner	53
5.6 WebRTC samtale software og signaling server (<i>SØF</i>).....	53
 5.6.1 WebRTC og signaling	54
 5.6.2 Modul design.....	55
 5.6.2.1 Signaling server.....	55
 6.5.2.2 WebRTC klient	62
 5.6.3 Formel modultest.....	75
 5.6.3.1 WebRTC klient	75
 5.6.4 Refleksioner	87
6. Bevis (HB, SØF, HJL).....	88
 6.1 Samlet system (HB).....	88
 6.2 Audio_handling (SØF).....	88
 6.3 WebRTC klient og Signaling server (SØF).....	89
 6.3.1 Signaling server	89
 6.3.2 WebRTC klient.....	89
 6.4 Bevis for succesfuld test af PIR Sensor	90
7. Projektledelse (HJL).....	91
 7.1 PBS. Fuldkommen produktliste (SØF, HJL, SHK, HB):.....	91
 7.2 WBS. Faseopdelt (HJL, HB, SØF, SHK).....	93
 7.3 Gannt Skema (HJL).....	97
 7.3.1 Første udkast. 14.11.2024	97
 7.3.2 Gannt Skema. Endelig version.....	98
 7.3.3 Gannt. Indsnævring af omfang og justering af tidsplan. (HJL).....	98
 7.4 Relations diagram med kritisk sti (SØF)	98
8. Perspektiv / refleksioner (SHK)	99
9. Konklusion (HB, SØF, SHK, HJL).....	100
10. Bilag og tillæg	101
 Bilag 1 – Moduldiagram (SØF, HJL)	101
 Bilag 2. Sensor Python Modul (HJL)	102
 Bilag 3. Succesfuld Sensor Test Video. (HJL).....	102
 Bilag 4. Gannt skema. (HJL)	103
 Bilag 5. Youtube video af Web-aktivering af LED.(HJL).....	103

Bilag 6 – Model træning og mappestruktur	104
6.1 Mappestruktur træningsmodel.....	104
6.2 LBPH træner.....	104
6.3 Kamera kode	108
Bilag 7. Database.....	111
7.1 – Tabeloverblik	111
Bilag 8. Web	113
8.1 – Home.....	113
8.2 – Library.....	114
8.3 – Open Door	115
8.4 - Udkast af Live-Feed.....	116
8.5 – Delete Video	117
Bilag 9 – Div. Kode	118
9.1 – app.py	118
9.2 – uploadvideo.py.....	121
9.3 – master.py.....	122
9.4 – main.py	122
9.5 – startup.sh.....	123
9.6 – env_startup.service	123
9.7 – Testvideo af uploadvideo & app.py	123
Bilag 10 – audio_handling (SØF).....	123
10.1 Flowcharts	124
10.2 Modultest kode.....	129
Modultest 1	129
Kode til audio_handling bevis	131
10.3 Audio_handling kode	132
Bilag 11 – Signaling server (SØF)	137
11.1 Endelig signaling server kode	137
Bilag 12 - WebRTC klient (SØF).....	141
12.1 Kode til test af WebRTC klient.....	141
12.2 Endelig WebRTC klient kode	142
Bilag 13 – Blokdiagram over PIR Sensorens sammenhæng i systemet. (HJL).....	149
Bilag 14 – Sensor Videoer.....	149

11.1 Litteraturliste:149

1. Indledning (HB)

Mange ældre føler sig utrygge i deres hjem, især med den stigende forekomst af tricktyverier, hvor fremmede udgiver sig for at være f.eks. hjemmehjælp. Det skaber bekymring blandt ældre og kan føre til at de oplever en dalende livskvalitet, da de bruger for meget energi på at bekymre sig om der nu er hjælp at hente, hvis de skulle blive konfronteret med en tricktyv. Vi har derfor valgt at udvikle et tryghedssystem, der kan give ældre større tryghed i eget hjem. Ligeledes kan det også anvendes af andre udsatte borgere, som f.eks. stalkingofre.

Det er vigtigt at understrege, at dette er vores første iteration og vi har skabt en prototype, som viser mulighederne og potentialet i et system, der integrerer kameraer, ansigtsgenkendelse og fjernstyret adgang til egen hoveddør, men vi er meget bevidste om, at et rigtigt sikkerhedssystem af denne kaliber vil kræve langt mere end, hvad vi har kunnet nå indenfor projektets tidsfrist. For at gøre det til en fuldt funktionel løsning, ville det kræve støtte fra en større organisation som f.eks. G4S eller en anden sikkerhedsfaglig organisation.

Grunden til at vi nævner en stor organisation, er at et sådant system vil kræve løbende opdateringer - især i relation til ændringer i GDPR-reglerne og nye krav til datasikkerhed. Derudover er der behov for en form for supportcenter, som kan hjælpe med de fejl, den vedligeholdelse og de softwareopdateringer, der måtte være nødvendige. Sådanne krav ligger uden for, hvad vi som en mindre projektgruppe kan løfte. Til gengæld har vi gjort os tanker om, hvordan systemet kan designes, så det ville være interessant for en sikkerhedsfaglig organisation som G4S at arbejde videre med.

Vi har lavet en tænkt situation, hvor vi forestiller os, at G4S kunne tage vores prototype og videreudvikle den til et færdigt produkt. Systemet kunne inkludere funktionalitet som et to-vejs-kommunikationssystem, som vi i opgaven har udviklet ideerne til et miljø til, samt livefeed, hvor der ligeledes er mulighed for, at man kan dele live video med sine pårørende eller en værge. Vi har fokuseret på at vise, hvordan ansigtsgenkendelse kan skabe værdi, og hvordan systemet potentielt kan hjælpe ældre og udsatte i deres hverdag.

Selvom vi i denne iteration har begrænset os til basal funktionalitet, ser vi store muligheder i det her system. For eksempel kunne fremtidige versioner integrere funktionalitet som faldregistrering, der sender alarm til pårørende eller hjælpere, hvis en ældre falder, eller en funktionalitet, der kan advare om, en dement borger er ved at forlade hjemmet uden opsyn.

Vi ser vores prototype som første skridt i en konceptmodel, der viser, hvad der er muligt og som kan danne grundlag for videreudvikling. Det her projekt handler ikke kun om at lave et færdigt produkt, men om at vise, hvordan teknologi kan gøre en reel forskel i hverdagen for ældre, udsatte borgere og deres pårørende. Derfor håber vi at store aktører som G4S vil kunne se potentialet

2. Metoder og værktøjer (HJL)

2.1 Induktiv metodisk tilgang under projektet. (HJL)

Den induktive metode er en undersøgelsesmetode, hvor man benytter indsamlede data, observationer og erfaringer. Ud fra disse forsøger man gennem analyse, at danne og identificere mønstre og tendenser - og herigennem opnår man ny erfaring - som kan lede til generelle konklusioner. Ud fra disse generelle konklusioner kan man altså træffe beslutninger ud fra et erfaringsbaseret grundlag.

Under arbejdet med projektet har vi gjort brug af en induktiv tilgang i forsøget på, at være mere systematiske i vores tilgang til fejlfinding når vi er løbet på problemer med f.eks. kode, integrering af systemer eller andre problemer.

Ved hjælp af den induktive tilgang har vi gennem tests fået data og erfaring og brugt disse i forsøget på at løse evt. problemer, fejl eller mangler. Målet har ikke været at bekræfte en hypotese som man oftest gør ved den induktive tilgang, men har derimod været at benytte den opnåede erfaring og resultater fra tests til bedre at forstå og derved forbedre projektet og produktet som helhed.

2.2 Værktøjer (WBS, Gantt, kodningsprogrammer mm.) (HJL)

Under styringen og planlægningen af projektet har vi i gruppen gjort brug af projektstyringsværktøjerne PBS (Product Breakdown Structure), for at lave vores WBS (Work Breakdown Structure). Vi har ligeledes udformet og gjort brug af et Gantt skema. Disse værktøjer har givet et bedre overblik over hele projektet samt givet os muligheden for at strukturere vores tid, uddeletere arbejdsopgaver og prioritere de vigtigste heraf.

Vi har ligeledes gjort brug af kodningsprogrammerne PyCharm og Visual Studio Code (VSC) som del af udviklingen og test af kode til projektet.

Produktet har 2 separate databaser, disse er en lokal database på RPI4B og en Postgres cloud database til metadata. Til benyttelse af den lokale linux database har der været benyttet WinSCP og PuTTY som ekstern forbindelse til RPI4B, hvilket har givet adgang til database og ændring af dokumenter. Til hosting og administration af clouddatabasen er aiven.io blevet benyttet til hosting, hvortil PgAdmin bliver brugt til administration af denne.

3. Virksomheden (HB)

G4S ønsker, at vi udvikler et innovativt og brugervenligt sikkerhedssystem, der er særligt tilpasset til at skabe tryghed for ældre borgere og personer, der oplever utryghed, for eksempel på grund af tricktyverier og stalking. Derfor skal vi have designet en løsning, der kombinerer avanceret teknologi med enkel betjening og mulighed for at involvere nærmeste pårørende.

1. Kamera med ansigtsgenkendelse

Systemet skal inkludere et kamera ved hoveddøren med integreret ansigtsgenkendelse, baseret på den nyeste teknologi. Kameraet kan identificere både kendte og ukendte ansigter og sender en notifikation til brugeren, når nogen nærmer sig døren.

2. Kommunikation via smartphone

For at sikre let adgang og tryghed skal systemet give brugeren mulighed for at kommunikere med besøgende gennem sin egen telefon. Med tovejs-lyd- og videokommunikation kan brugeren se og tale med den besøgende, uden at skulle åbne døren.

3. Fjernstyret dørlås

Med et enkelt tryk kan brugeren låse døren op via appen, når brugeren føler sig tryg ved besøgende. Dette giver fleksibilitet og gør det muligt for brugeren at åbne døren fra sin smartphone, hvilket er både praktisk og sikkert.

4. Mulighed for netværk af pårørende og venner

Vi ved, hvor vigtigt det er at have en pålidelig støtte i nærmeste omgangskreds. Derfor har vi et ønske om at udvikle en løsning, hvor pårørende, venner og familiemedlemmer kan tilknyttes systemet som ekstra brugere. Disse personer kan modtage notifikationer, få adgang til livekameraet og hjælpe brugeren, hvis der opstår en utryg situation.

5. Nød- og historikfunktioner

Systemet inkluderer en nødknap i appen, som gør det muligt hurtigt at kontakte pårørende eller alarmcentralen, hvis der skulle opstå en akut situation. Derudover gemmes alle besøg og interaktioner med systemet i en log, så brugeren og deres netværk altid kan få et overblik over, hvem der har været ved døren.

Hos G4S ser vi dette system som mere end blot et sikkerhedsprodukt. Det er en løsning, der skaber tryghed og sikrer, at ældre og udsatte borgere kan føle sig sikre i deres eget hjem. Vi ønsker at gøre hverdagen mere overskuelig og sikker – og give vores kunder mulighed for at bevare deres selvstændighed med støtte fra deres netværk.

4. Krav (HJL, SØF)

Kravene til projektet er udstedt af studiet og dets undervisere. Herunder er der blevet udstedt en officiel teknisk rapportskabelon som følges ifm. dokumentation af projektet. Denne skabelon dikterer bl.a., at rapporten skal være op til 40 sider, hvilket er tilsvarende 96000 tegn.

Herunder ses ligeledes kravspecifikationen, som er godkendt af vejleder Jonas Wehding.

Kravspecifikation (SHK, HB, SØF, HJL)

Projektnavn	Tryghedssystem
Formål	At udvikle et tryghedsskabende system, der tilgodeser trygheden hos private borgere, der har et ønske herom. Uden at komme i vejen for juridiske regler som GDPR og almene lovovertrædelser i relation til opbevaring af data og overvågning.
Omfang	<p>Omfanget af produktet skal fungere således, at systemet skal kunne: Streame og optage både lyd og video samt lagring af disse. Systemet skal ligeledes kunne genkende brugeren/ejeren og dens nære, og kunne tilgås gennem et webbaseret interface.</p> <p>Projektet skal gøre brug af omfattende projektledelse og munde ud i en færdig prototype af et produkt, samt en dertilhørende teknisk rapport.</p> <p>Derudover skal systemet kunne lave alarmkald både ved brug af lokal alarm samt til vagtcentral, og låse/låse op for indgangen til boligen.</p>
Tidsplaner og frister	Projekt og rapport afsluttes d. 3. December. Afleveringsfrist d. 6. December. <u>Projektafslutning v. eksamen d. 5-7. Januar.</u>
Risikohåndtering	Taget i betragtning at projektet er styret og ført udelukkende af studerende er der en reel risiko for, at tidsplanen ikke holder som først antaget. Dette vil kunne resultere i, at det kan være en nødvendighed at nedskalere omfanget af projektet.

Interessentkrav, Ressourcekrav, og andre ekskluderede segmenter som normalt er inkluderet i en kravspecifikation er ikke inkluderet på baggrund af vejledning fra AER.

5. Moduler

5.0 RPI 4B (HJL)

En Raspberry Pi er en lille computer, hvilken kan bruges til at lave forskellige computer- og IoT-projekter.

RPI 4B er omdrejningspunktet for hele produktet- og dermed til dels ligeledes projektet. RPI 4B er en forkortelse for Raspberry Pi 4 Model B, og er en opgradering af det Raspberry produkt, der tidligere har været benyttet på uddannelsen, en Raspberry Pi Pico.

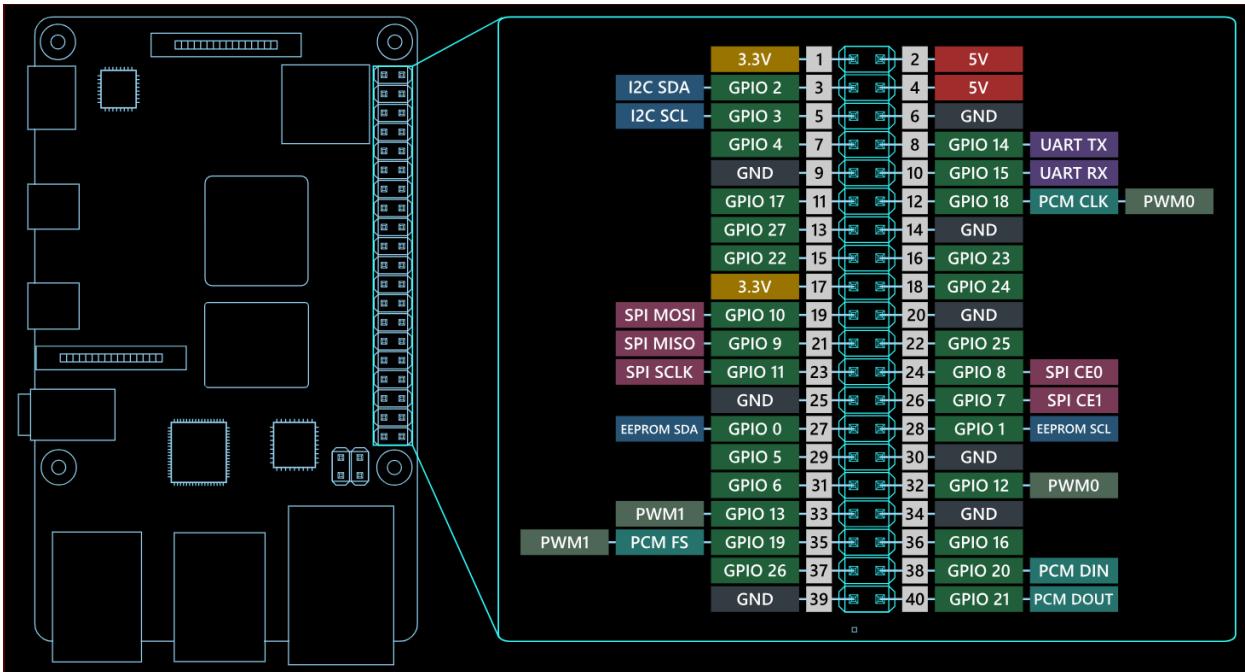
RPI 4B har en højere ydeevne, flere porte, flere tilslutningsmuligheder og er dermed mere fleksibel i sin anvendelse end sine forgængere. Den er baseret på en BCM2711 chip, som har en 4-kernet processor. Denne model af RPI har ligeledes mere ram end forgængerne. RPI 4B er kompatibel med bl.a. 4K og VMware.

Som nævnt ovenfor, har RPI 4B et udvidet antal porte og tilslutningsmuligheder. Herunder:

- 2 USB 3.0 porte
- 2x USB 2.0 porte
- Ethernet Port
- Wi-Fi mulighed gennem Wi-Fi 802.11ac.
- Bluetooth
- 2 Micro-HDMI porte.
- 1 RPI kamera port
- 1 RPI display port
- GPIO Pins

Modellen kræver en 5V strømforsyning. Denne kan skabes gennem en USB-C indgang fra f.eks. en bærbar computer.

Et overblik over Pin-layoutet på Raspberry Pi'en ses på billedet nedenfor:



Begrænsninger

RPI 4B har ingen intern lagring, hvilket gør, at den afhænger af eksterne harddiske eller micro SD kort hvis der skal lagres data lokalt.

Modellen har ligeledes intet dedikeret operativsystem.

Workarounds

Modellens begrænsninger har medført, at der i gruppen har været brug for at finde diverse løsninger til at arbejde uden om begrænsningerne.

Vi har i gruppen forsøgt at arbejde omkring modellens begrænsninger ved bl.a. at gøre brug af et cloud- og lokalt baseret databasesystem for at kunne håndtere større mængder data, siden denne model af RPI ikke supportere intern lagring. For at få lokale lagring til projektet har vi valgt at gøre brug af et SD kort.

For nemmere at kunne arbejde med produktet har vi ligeledes gjort brug af tredjepart operativsystemet Raspberry PI OS Desktop 64-bit. Dette operativsystem bliver benyttet med grundelsen at Desktop OS er kompatibelt med det valgte hardware i produktet.

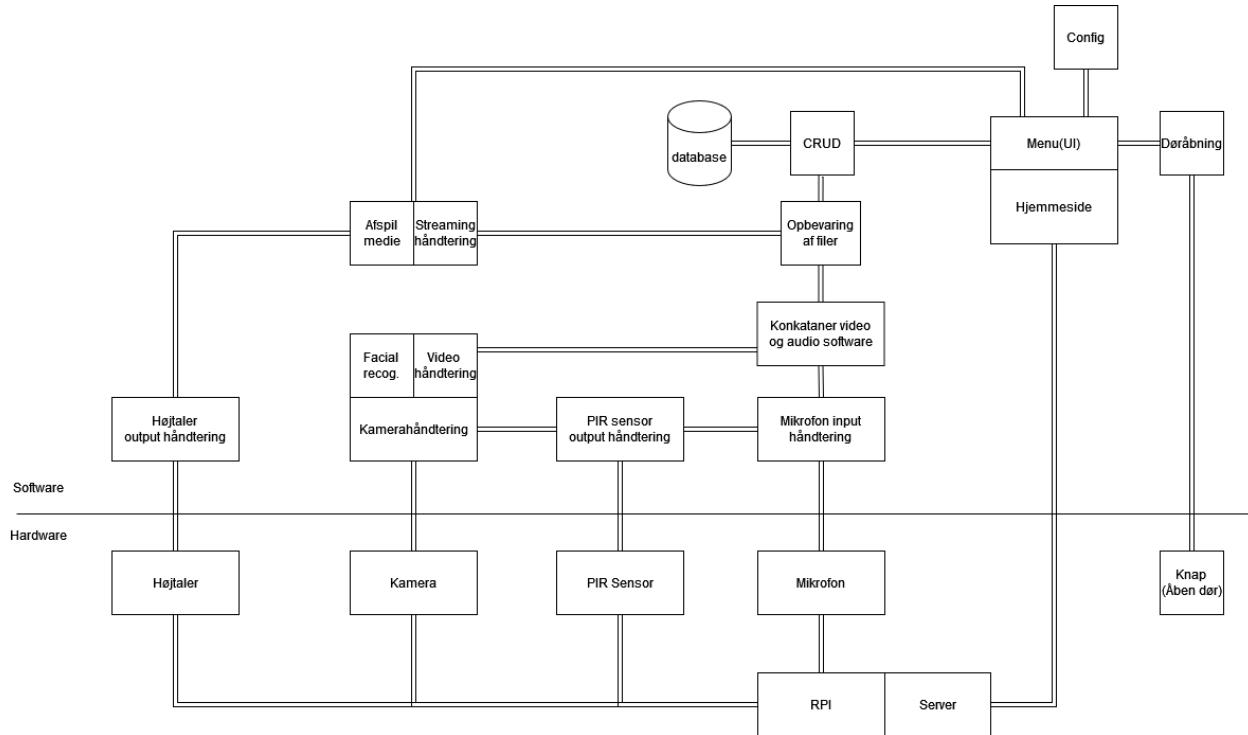
For yderligere information, se databladet¹.

¹ [Raspberry Pi 4 Model B Datasheet](#)

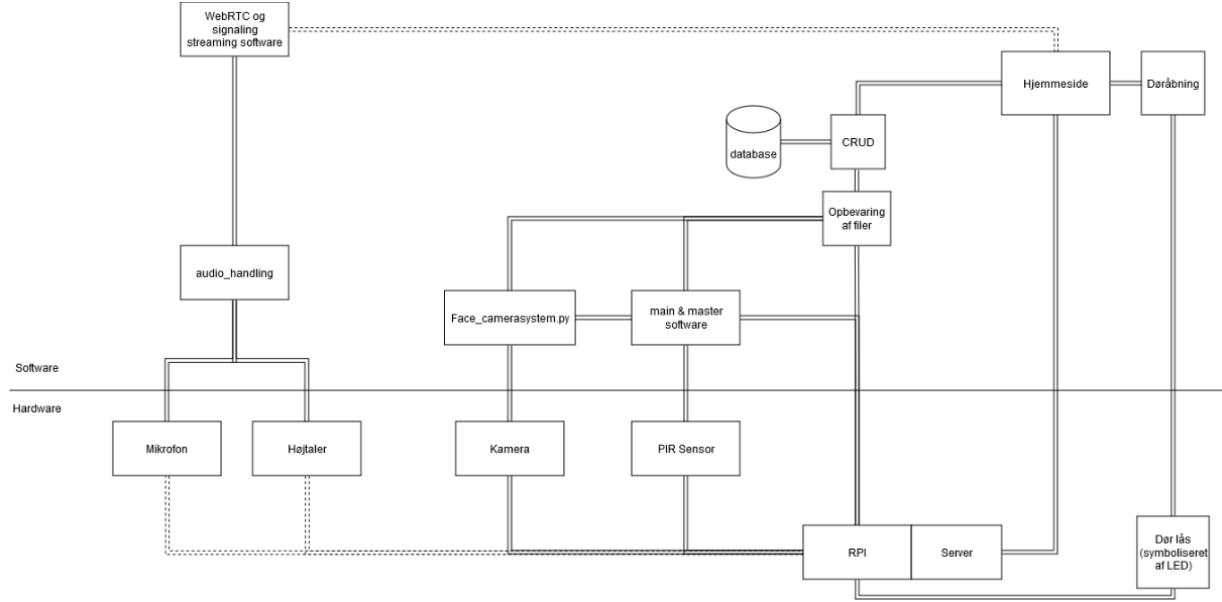
5.0.1 - Overordnet moduldiagram (HJL, SØF)

HJL udarbejdede et overordnet moduldiagram, som SØF senere udvidede for bedre at kunne skelne mellem hardware of software. Diagrammet viser systemet som helhed, og er udarbejdet jf. vores PBS, således der tages højde for samtlige under-produkter.

Dette diagram er benyttet som værktøj til udvikling af systemet, for at sikre, at hvert modul kan interagere korrekt med dets relaterede moduler, samt at unødvendige features ikke tilføjes produktet. I følgende afsnit opdeles disse og forklares uddybende.



Herunder er den endelige version af vores overordnede modul-diagram, udviklet af SØF og HB, som bygger videre på den oprindelige version af HJL. De stiplede linjer symboliserer WebRTC-signaling og audio_handling modulernes fremtidige tilslutning til systemet.



5.1 Video og Ansigtsgenkendelse (HB)

5.1.1 Modul Design

For at komme i gang er det essentielt at definere begreberne ansigtsdetektion og ansigtsgenkendelse, så der opnås en grundlæggende forståelse, inden man bevæger sig videre til modeltræning.

Ansigtsdetektion refererer til processen med at identificere og lokalisere ansigter i et billede eller en videostream. Dette opnås typisk ved at træne en model baseret på Haar-kaskader.

Ansigtsgenkendelse går et skridt videre og forsøger at identificere eller verificere en persons identitet baseret på ansigtet. Denne proces benytter metoden Local Binary Pattern Histogram (LBPH), som analyserer tekstur- og pixelmønstre.

Ved at træne og finjustere begge processer korrekt sikrer man, at systemet kan detektere og genkende ansigter med høj præcision. Det er afgørende at opsætte et solidt træningsmiljø for ansigtsgenkendelsesmodellen og teste den grundigt, inden den implementeres. Dette minimerer risikoen for falske positiver og falske negativer.

Fremgangsmåde for træning af en ansigtsgenkendelsesmodel:

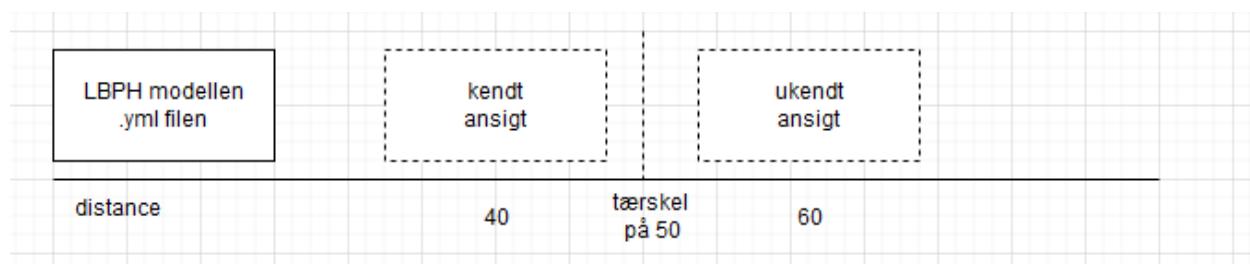
I bilag mappestruktur træningsmodel, kan man se mappestrukturen og i bilag LBPH træner kan man finde alle de python koder, der her omtales for at få den .yml fil, der ansigtsgenkender.

- Start med at tage omkring 1000 billeder af personen, der skal registreres i ansigtsgenkendelsen. Gem disse billeder i mappen positive_folder.
- Saml derefter 1000 billeder uden ansigter, og placér dem i mappen negative_folder, så forholdet mellem positive og negative billeder er 1:1. Dette sikrer en balanceret træning af modellen.
- Sørg for at hente filen haarcascade_frontalface_default.xml fra GitHub. Denne fil bruges som grundlag for ansigtsdetektion.
- Brug Python-programmet picture_taker.py, der automatisk tager billeder af en person og gemmer dem i mappen faces_folder.
- Brug programmet Negative_generator.py til at generere forskellige billeder ud fra 10 originale billeder, som placeres i mappen negative_images. De genererede billeder gemmes automatisk i negative_folder.

Modeltræning:

Brug Python-programmet model_trainer.py til at træne modellen på de indsamlede billeder. Resultatet bliver en .yml-fil, der gemmes som akmodel.yml (se bilag for mappestruktur).

Test modellen på både kendte og ukendte ansigter ved hjælp af programmet recognize_tester.py.



LBPH bruger distancen som en metode til at sammenligne to histogrammer, for at afgøre hvor ens de to er. I ansigtsgenkendelse betyder det, hvor ens to ansigter er og derved kan modellen give en vurdering af om to ansigter er ens – jo mindre distance, jo mere ens er to ansigter.

```

if distance < 50: # Tærskel for genkendelse
    print(f"Genkendt ID: {label} med sikkerhed: {round(distance, 2)}%")
else:
    print("Ukendt ansigt")

```

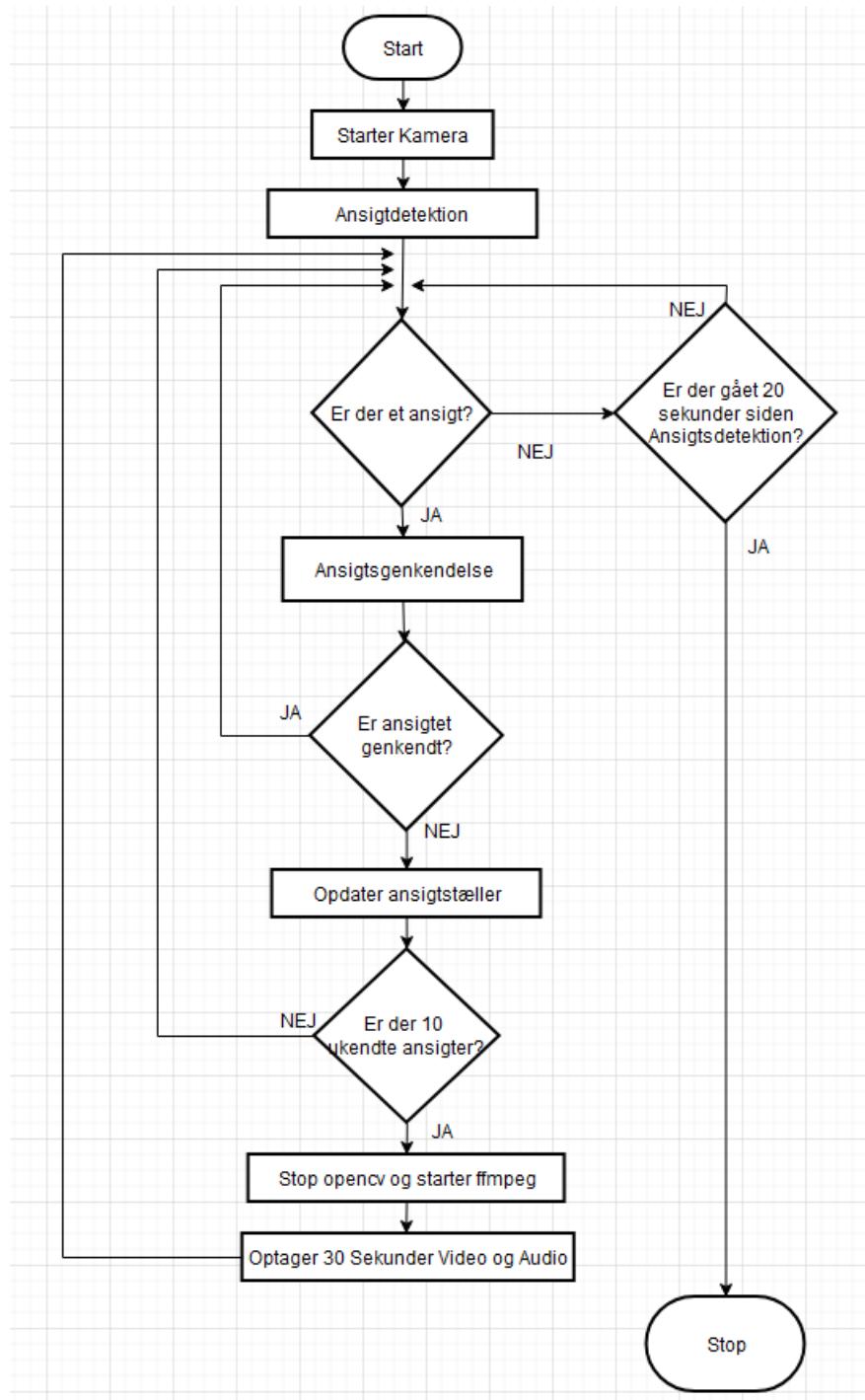
Justér LBPH-modellens distancevariabel for at optimere systemets præstation:

- Lave værdier betyder at modellen er sikker på, at ansigtet matcher en personen, hvis de er lave nok til at være indenfor den programmerede tærskel.

- Høje værdier betyder at ansigtet klassificeres som ukendt. Det vil sige at distancen fra modellen til det kameraet læser er langt.

Python programmerne til denne fremgangsmåde ligger under bilaget "Modeltræning" og der ligger også overvejelser omkring mappe struktur.

Flowdiagram af face_camerasytem.py



Start

Processen begynder med en startbetningelse, der bliver sendt fra PIR sensoren til RPI4B som sender signalet til kameraet.

Kameraaktivierung

```
while True:  
    frame, faces = face_detector.detect_faces(camera)
```

Koden tænder kameraet og begynder at analysere billeder for at opdage ansigter.

Ansigtsgenkendelse

```
if faces is not None and len(faces) > 0:  
    results = face_recognizer.recognize_faces(frame, faces)
```

Efter kameraaktivierung forsøger programmet at identificere, om der findes et ansigt i kameraets synsfelt.

- Hvis et ansigt detekteres, fortsætter processen til ansigtsgenkendelse.
- Hvis ikke, evalueres, om der er gået 20 sekunder siden sidste detektion.

Timeout på 20 sekunder

- Hvis der ikke detekteres et ansigt inden for 20 sekunder, stopper processen, og kameraet nulstilles til en ny overvågningscyklus gennem PIR sensor aktivierung.

Ansigtsgenkendelse

```
for result in results:  
    label = str(result["label"])  
    distance = result["distance"]  
    print(f"Label: {label}, distance: {distance:.2f}%)")
```

Når et ansigt er detekteret, analyserer programmet ansigtet og forsøger at genkende det ved hjælp af den forud trænede LBPH model

- Hvis ansigtet genkendes, nulstilles tælleren for ukendte ansigter, og processen vender tilbage til ansigtsdetektion.

- Hvis ansigtet ikke genkendes, opdateres tælleren for ukendte ansigter

Tæller for ukendte ansigter

```
if label == "Unknown":  
    unknown_frame_count += 1  
    print(f"Ukendt ansigt detekteret i {unknown_frame_count}/{unknown_threshold} frames.")  
else:  
    unknown_frame_count = 0
```

For hvert ukendt ansigt opdateres en tæller. Denne tæller holder styr på, hvor mange ukendte ansigter der er blevet observeret i træk.

- Hvis tælleren når en tærskelværdi på 10 ukendte ansigter i træk, aktiveres optagelse.
- Hvis tærsklen ikke nås, fortsætter systemet med at overvåge.

Optagelse af video og lyd

```
if unknown_frame_count >= unknown_threshold:  
    print("Starter optagelse på grund af vedvarende ukendt ansigt.")  
    output_file = os.path.join(output_dir, f"recording_{datetime.now().strftime('%Y-%m-%d_%H-%M-%S')}.mp4")  
    camera.release()  
    recorder.record_video_with_audio(output_file, recording_duration)  
    camera = cv2.VideoCapture(0)  
    unknown_frame_count = 0
```

Når tærskelværdien for ukendte ansigter nås, stopper systemet midlertidigt kameraovervågningen og starter en optagelse ved hjælp af FFmpeg.

Optagelsen varer i 30 sekunder og inkluderer både video og lyd, som gemmes for videre analyse.

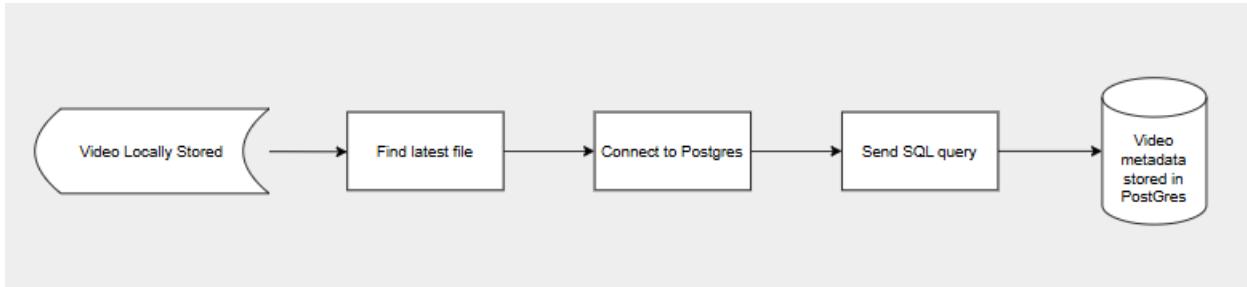
Stop

```
while datetime.now() < end_time:  
    ret, frame = camera.read()  
    if not ret:  
        break
```

Efter optagelsen afsluttes processen, og programmet vender tilbage til den oprindelige tilstand for at genoptage overvågning gennem pir sensoren og sker der intet efter 20 sekunder bryder programmet ud af mit modul.

Hele koden kan findes i bilag 13.3

Upload af video til database:



Find latest file:

- Finder den seneste fil som er blevet lavet i et forudbestemt directory, ved brug af os.

Connect to Postgres:

- Skaber forbindelse til Postgres database ved brug af prekonfigureret DB Config

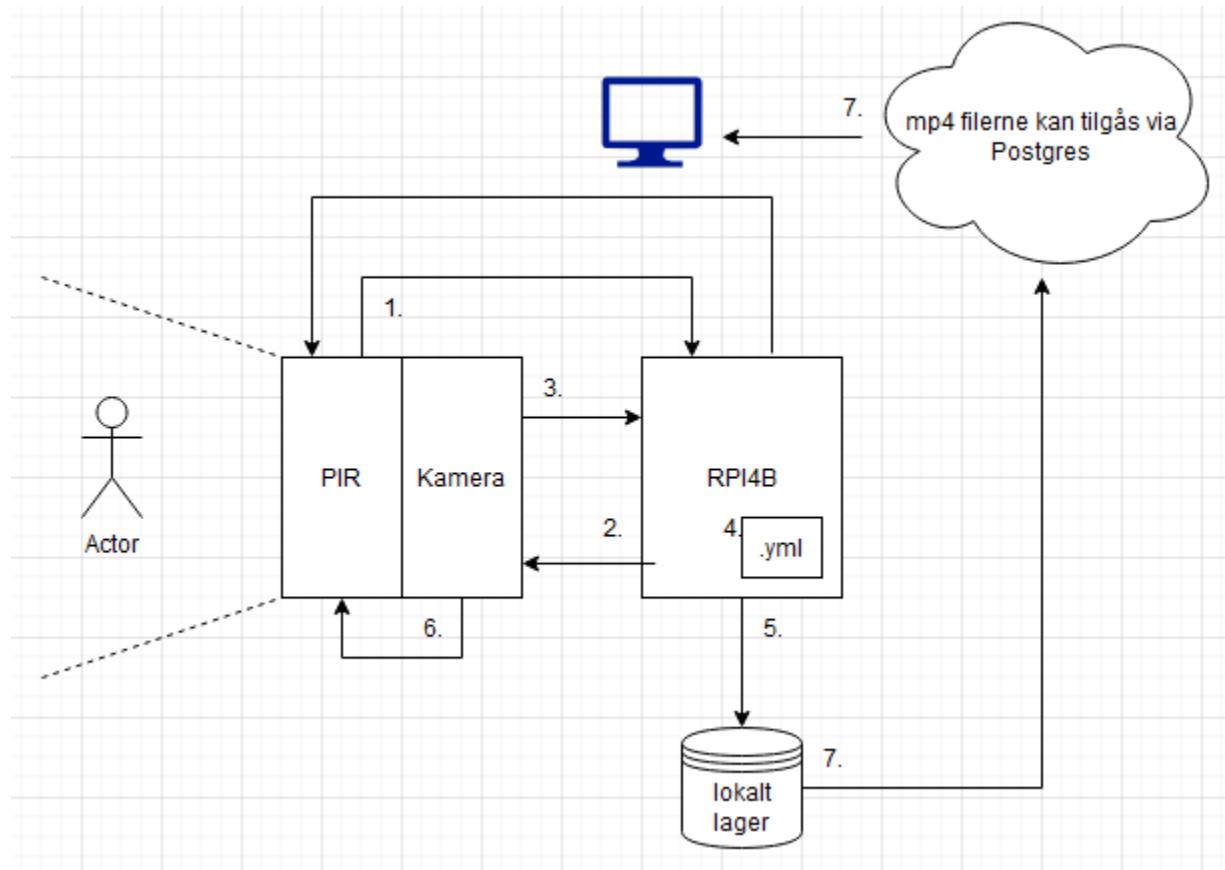
Send SQL query:

- Sender en konfigureret SQL query til Postgres indeholdende metadata om specifik fil vha. os.
2

² Bilag 9.2 – uploadvideo.py

5.1.2 Modul implementering

Modulet er et led i hele sikkerhedssystemet og det vil i det følgende blive beskrevet ud fra den kontekst modulet er implementeret. Den visuelle præsentation ses i sin helhed herunder og dernæst kommer beskrivelsen på den.



1. PIR sensor aktiveres af bevægelse og sender signalet til RPI4B
2. RPI4B modtager signalet fra PIR sensoren og sender signal til kameraet om, at det skal starte.
3. Kameraet sender billeder til RPI4B, som behandler billederne i relation til koden. Hvis ansigtet genkendes, sender programmet i sin nuværende form, et distanceprint som ligger under 50, med de enkelte falske negative, som der kan komme med en buffer på 10.
4. Hvis kameraet sender billeder som projektets nuværende model i .yml filen ikke kan genkende, det vil sige at den overstiger distanceprint på 50 frames og bufferen på 10 frames, så aktiverer RPI4B ffmpeg modulet, der starter optagelse og samler mp4 og mp3 i en mp4 fil
5. Efter 30 sekunders optagelse, lægger programmet den ffmpeg skabte mp4 fil i en lokal database på RPI4Bs SD kort.

6. Er der ikke mere at detektere for kameraet, så bryder programmet ud af face_camerasystem.py programmet og vender tilbage til master.py og igen venter på feedback fra PIR sensoren.

7. brugeren kan nu tilgå den ffmpeg skabte mp4 fil, som ligger på det lokale lager via Postgres serveren.

Opsætning af Postgres-database

For at kunne lagre metadata tilhørende videooptagelser blev en Postgres-database oprettet og konfigureret ved brug af aiven.io og PgAdmin. Forbindelsen hertil bliver etableret via Psycopg2. Tabellen blev oprettet via et simpelt SQL-script og indeholder følgende felter:

- **Video ID:** En unik nøgle, der letter søgning og administration af data.
- **Filename:** Navnet på videofilen, hvilket gør det muligt at identificere den specifikke fil.
- **Created_at:** En tidsstemppling der angiver, hvornår filen blev oprettet i systemet.

Selve opsætningen blev udført uden nævneværdige udfordringer og medfører en struktureret lagring af relevante data. De valgte felter blev designet med henblik på at understøtte systemets behov, herunder nem adgang til filernes placering og muligheden for effektiv administration via det unikke ID.

Vedligeholdelse og opretholdelse af databasen foretages via PgAdmin, hvilket giver en grafisk brugergrænseflade til administration af tabellen.

Ved hjælp af denne database kan metadata nemt anvendes som referencer til at finde og administrere filer gennem operativsystemet (os-bibliotek).³

Implementation af uploadvideo.py (SHK):

I forbindelse med at uploade den specifikke videooptagelses metadata til PostGres database er der brug for 3 forskellige python-biblioteker. De 3 python-biblioteker består af:

OS (Dette skal bruges til at kunne tilgå de specifikke filer, hvorfra vores metadata afstammer)

- *Psycopg2 (Dette er for at kunne skabe forbindelse til Postgres database, samt kunne sende SQL querlys hertil)*
- *Datetime (Dette bruges til at anvende et timestamp for tidspunktet af upload til database)*

Den kronologiske proces foregår ved at der som nævnt bliver importeret de 3 ovenstående biblioteker. For at kunne gøre brug af Postgres database startes der med oprettelse af en DB Config, så alle de nødvendige login/informationer om databasen bliver angivet og kan benyttes til etablering af forbindelse til den specifikke Postgres database.

³ Bilag 7.1 - Tabeloverblik

Derefter bliver der gjort brug af os-biblioteket, brugen af denne skal benyttes til at finde den seneste fil i enhedens directory. Først benyttes `os.listdir` for at angive det directory, som skal benyttes til at finde den givne .mp4 fil (`f.endswith`). Dernæst findes den seneste optagelse ved brug af `max`, `lambda-key` og `os.path.getctime`, hvortil den fil med den højeste value af `getctime` bliver valgt som `latest_file`. Den `key` som bliver brugt i dette tilfælde er `lambda`, og den hjælper med at vise hvilken parameter `max` skal kigge efter, hvilket er `getctime`.

Dernæst etableres der forbindelse til Postgres database ved hjælp af `psycopg2 & DB Config`, hvorefter der bliver indsendt en query bestående af metadata fra den valgte fil.⁴

5.1.3 Formel modultest

Hele den formelle modultest ligger på dette link: <https://youtu.be/z0fyJ8KGEjo>

Variablen i koden er rettet til `distance` i stedet for `confidence`, som det vises i videoen, `distance` skal forstås som afstanden fra den indkodede models data i .yml filen i relation til de billeder kameraet opfanger. Jo mindre `distance` der er, jo tættere er billedet på modellen inden for `threshold`, som er defineret på 50. Det er eget ansigt (HB) som er kodet ind i modellen og den viser en `distance` på < 50

Label: 1, Distance	44.24%
Label: 1, Distance	47.91%
Label: 1, Distance	46.88%
Label: 1, Distance	40.50%
Label: 1, Distance	36.22%
Label: 1, Distance	37.94%
Label: 1, Distance	33.73%

Hvis `threshold` bliver > 50 så tæller programmet op til 10 frames for ikke at optage ved en falsk negative, men det kræver 10 konsekutive negative for at aktivere kameraet. På billedet kan man se, når kameraet ændrer sig til et ukendt ansigt:

Ukendt ansigt detekteret i 8/10 frames.
Label: Unknown, distance 56.94%
Ukendt ansigt detekteret i 9/10 frames.
Label: Unknown, distance 53.40%
Ukendt ansigt detekteret i 10/10 frames.
Startet optagelse på grund af vedvarende ukendt ansigt.

⁴ Bilag 9.2 – uploadvideo.py

Herefter begynder kameraet at sende billeder til optagelse.

```
frames= 33 fps= 31 q=29.0 size= 21
frames= 48 fps= 31 q=29.0 size= 21
frames= 63 fps= 31 q=29.0 size= 21
frames= 79 fps= 31 q=29.0 size= 21
frames= 93 fps= 31 q=29.0 size= 21
frames= 109 fps= 31 q=29.0 size= 21
frames= 125 fps= 31 q=29.0 size= 21
frames= 140 fps= 31 q=29.0 size= 21
```

Efter 20 sekunder uden ansigt afsluttes programmet.

(error) үзүүлэхийн эхийн талбай нь 5.

Test af uploadvideo.py:

Første iteration af uploadvideo.py havde til forskel at ved valg af fil blev der benyttet en statisk directory reference, hvilket betyder, at den kun kan tage den angivne fil og uploade den til Postgres-databasen. Dette gjorde python programmet for afhængigt af koden, til at bestemme hvilken fil der skulle benyttes til videresendelse. Herudover var der også unødvendigt meget metadata, som blev tilsendt til Postgres uden efterbehov af dette stykke metadata (filepath). Første test fungerede som det skulle, men ville ikke være repræsentativt for, hvordan den endelige kode skulle fungere, og blev herefter justeret.

Efter justering af ovenstående problemstillinger, blev det besluttet at løsningen for at finde den rigtige fil, altid ville være at finde den nyeste oprettet video. Dette kunne både gøres som at den henter en fil fra et specifikt directory, såsom `/static/videos`, men dette er der ikke behov for da den nyeste oprettede fil altid vil være filen som skal uploades til databasen. Samtidigt med ovenstående handling, blev der også fjernet det unødvendige metadata, så der undgås at bruge processorkraft på dette selvom det er minimal spildkraft. Test af ovenstående fungerede optimalt og det ønskede output var som forventet.⁵

⁵ Bilag 9.7 – Testvideo af uploadvideo.py

5.2 HC-SR501 PIR Sensor modul (HJL)

HC-SR501 Funktionalitet

HC-SR501 sensoren er en PIR (passiv infrarød sensor). Nedenfor er en gennemgang af de vigtigere af sensorens egenskaber og funktioner. Informationen kommer fra sensorens datablad⁶.



- Device Initialization. Sensoren tager et minut om at starte op. I denne tid kan sensoren outputte falske signaler.
- Sensoren opfanger ændringer i infrarød stråling (bevægelse) i en 110 graders vinkel, med en rækkevidde på op til 7 meter.
- Rækkeviden er justérbar fra 3 til 7 meter.
- Sensorens output er enten HIGH eller LOW. HIGH betyder, at sensoren opfanger bevægelse. Når output er LOW er sensoren i dvale. Dvaletilstanden er sensorens "default" mode, hvor den afventer en ændring i infrarødt signal.
- Time Delay. Denne kan justeres til at være mellem 3 sekunder og 5 minutter. Time Delay betyder, at efter sensoren har detekteret en ændring i infrarød stråling og derfor outputter HIGH, forbliver outputtet HIGH i den mængde tid som er angivet ved justeringen.
- Sensoren er som udgangspunkt i "Single Trigger Mode" hvilket betyder, at Time Delay begynder så snart en ændring i infrarød stråling opfanges.

Sensoren er til projektet opsat og justeret således, at Time Delay er sat til minimum (~3 sekunder). Grunden til dette er, at vi udelukkende skal bruge det umiddelbare signal at der er opfanget bevægelse, da dette signal skal tænde for kameraet og resten af tryghedssystemet. Herefter bør softwaren tage over ift. hvad der skal foretages herefter.

⁶ [Datablad for HC-SR501 Sensor](#)

5.2.1 Python PIR Modul (HJL)

For et overblik over hele koden, se bilag⁷.

På afleveringstidspunktet er koden endt med at være kortet ned til et minimalt antal linjer. Derfor er ovenstående kode ikke endt med at være den version, som er implementeret i det endelige produkt på afleveringsdagen.

Formålet med PIR sensoren og det dertilhørende python modul er, at sensoren skal fungere som en aktivator til kameraet og andet software. Eftersom sensoren er en *passiv* sensor, er strømkravet hertil væsentlig lavere end hvis et kamera skulle aflæse aktivitet foran brugerens dør.

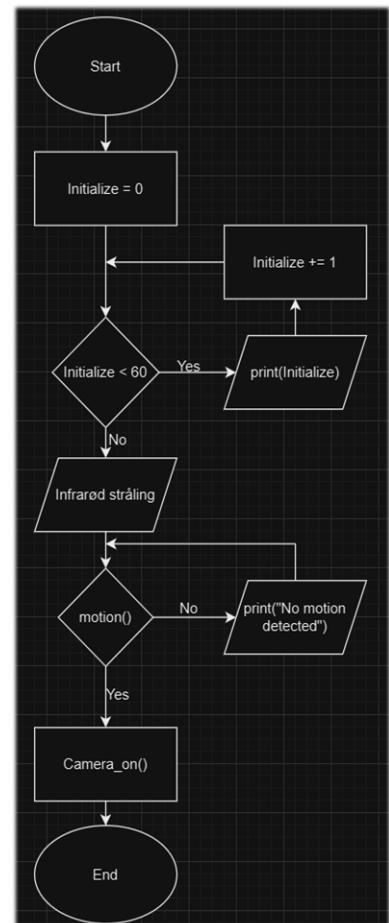
Hermed ses det ligeledes i designet af python koden til sensoren, at det endelige formål er at kalde en aktiveringsfunktion til kameraet.

Initialize variablen som ses i flowchartet er til for at sikre, at sensoren får 60 sekunder til at starte korrekt op og herved ikke outputter falske signaler⁸.

IPO-Chart, Sensor.

Input	Processing & Storage	Output
Ændring i Infrarødt signal	<ul style="list-style-type: none">- Sensoren opfanger ændringer i infrarød stråling.- Når signalet er højt er der en ændring i infrarød stråling (bevægelse) foran sensoren.	<ul style="list-style-type: none">- Motion(): Tænd-Kamera funktionen kaldes.- No_motion(): Når PIR ikke ser bevægelse er den i dvale.

Sensor Modul kode 1. udkast



⁷ Bilag 2, Sensor Python Modul.

⁸ Se afsnit 5.2 vedr. Device Initialization.

5.2.2 Modul implementering (HJL)

Pr. d. 3/12 er modulet er implementeret på en sådan vis, at det er integreret direkte i "Master" (styrings) softwaren. Koden til sensoren sørger for, at når sensoren opfanger en ændring i Infrarød stråling, så tændes kamera'et. Se Master Styringsmodul for yderligere.

5.2.3 Formel modultest (HJL)

Test 1.0. Første test.

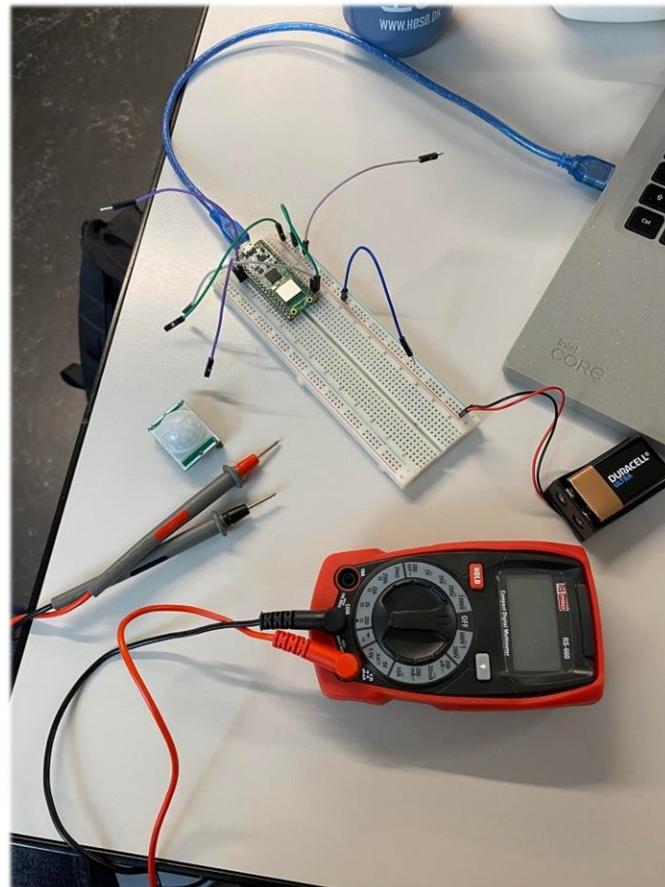
19/11/2024

Efter at have brugt dagene forinden på at designe og udvikle første udkast til koden, så første test således ud.:

Der blev gjort brug af HC SR501 sensoren, en raspberry pi pico W 2022, et breadboard og et 9V batteri.

Batteriet var antaget nødvendigt idet undertegnede på tidspunktet var af opfattelsen, at RPI Pico W 2022 ydede 3,3 V og sensoren kræver mindst 5,0 V.

Forinden testen kontrollerede undertegnede, at opsætningen stemte overens med kravene i databladet for HC SR501 sensoren. Databladet findes i rapportens bilag⁹.



⁹ [Datablad for HC SR501 PIR Sensor – Hyperlink](#)

Efter flere forsøg var det ikke muligt at få et brugbart output fra sensoren. Sensorens output skiftede aldrig fra LOW til HIGH. Dette ses på billedet til højre. Billedet viser terminalen fra VSC.

```
>>>
Initializing. Please wait 60 seconds...
Sensor ready to detect motion. Press Ctrl+C to exit.
Sensor state: 0
```

Undertegnede fik herefter hjælp fra Torben Egelund Rauff til at isolere fejlen. Herigennem fandt vi frem til, at koden virkede – idet koden kunne registrere ændringer i spændingen. Herudfra blev det konkluderet, at HC SR501 sensoren var defekt, hvorfor en ny blev bestilt.

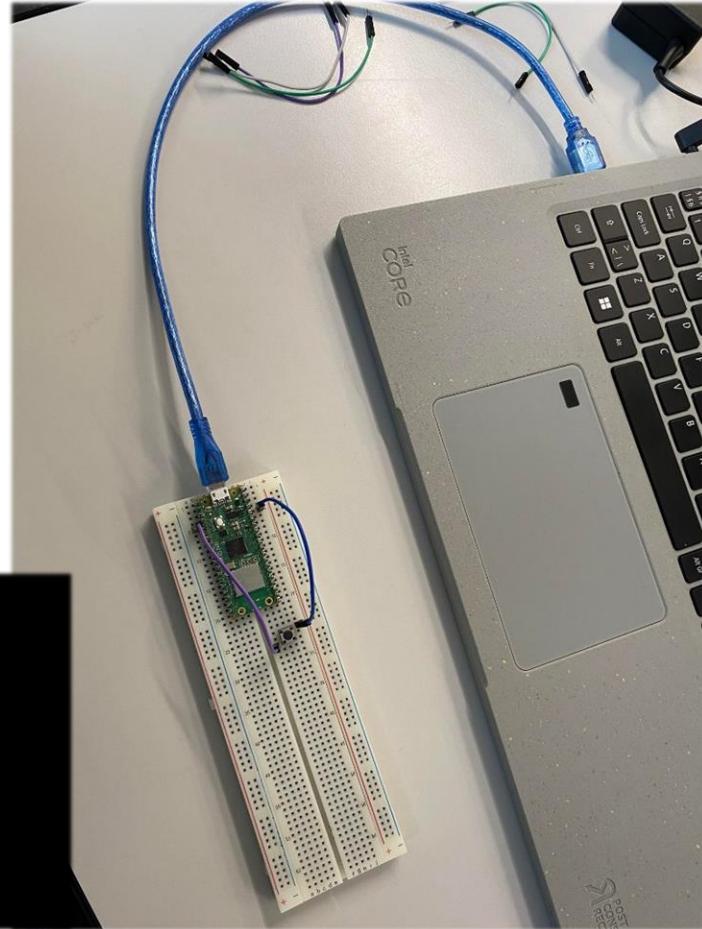
Test 1.1 Knap-test.

19.11.2024

For at simulere at koden korrekt kunne registrere ændringen i states, opsatte undertegnede en "Knap-test" som set på billedet til højre. Koden blev redigeret således at den kunne håndtere en fysisk knap.

Testen viste, at koden korrekt kan aflæse ændringen i state. Dette ses i udklippet fra terminalen på billedet nedenfor.

```
>>>
Initializing. Please wait 60 seconds...
Sensor ready to detect button press. Press Ctrl+C to exit.
Button state: 0
Button state: 0
Button state: 0
Button pressed! Motion detected!
Button state: 1
Motion detected!
Button state: 0
Button pressed! Motion detected!
Button state: 1
Motion detected!
Button state: 0
Button state: 0
```



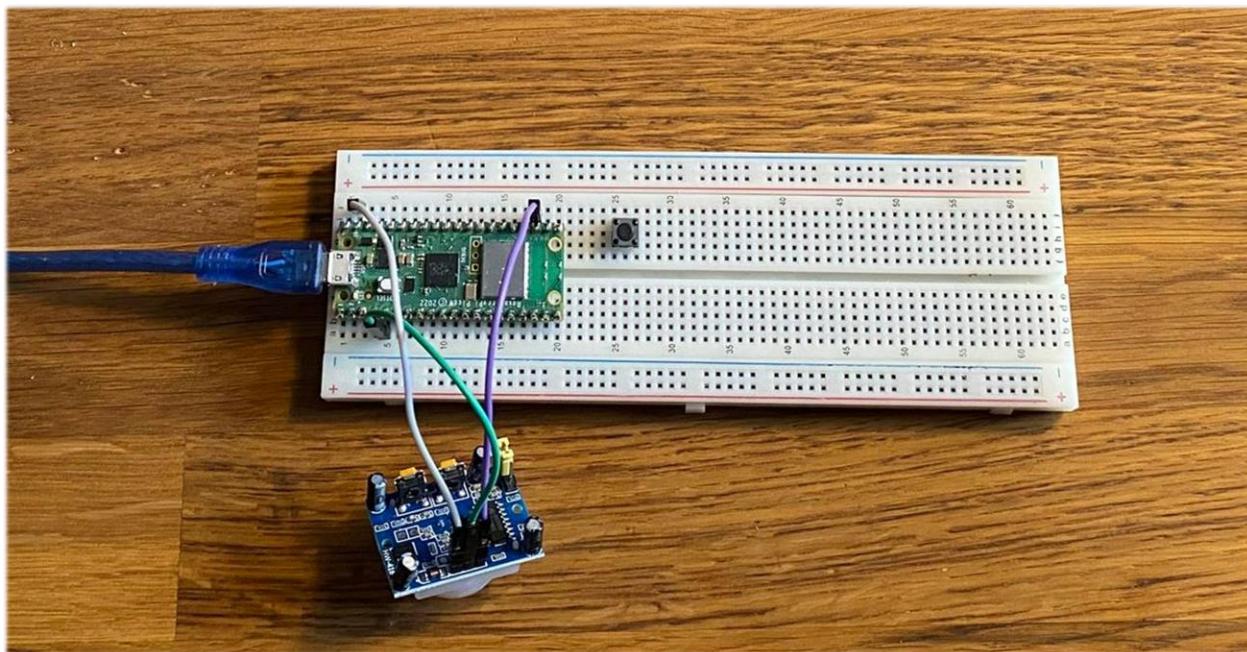
Test 2. Succesfuld test.

21.11.2024

For at se video test, klik på følgende hyperlink: [Link¹⁰](#)

Denne test blev foretaget efter jeg modtog den nye sensor, efter den tidligere blev fundet defekt.

Opsætningen til denne test var som følger.: Sensoren fik strøm fra en bærbar PC gennem USB'en i Pico'ens VSYS Pin. Sensorens output Pin er tilknyttet Pin 3, og sensoren er afslutningsvis tilknyttet GND.



I videoen ses det, at sensoren og koden korrekt registrerer denne bevægelse. Hver gang jeg bevægede min venstre hånd ind foran sensoren, blev bevægelsen opfanget, hvilket ses i form af stub-test printet "Motion Detected" i Terminalen i Visual Studio Code.

Sensoren er opsat således, at Time Delay'en er blevet sat til minimum, hvilket er ca. 3 sekunder. Dette betyder, at der skal gå den angivne mængde tid efter et HIGH utfør, før sensoren kan aflæse ny bevægelse. Dette er grunden til, at undertegnede i videoen venter med at skabe bevægelse med et mellemrum på et par sekunder og ikke konstant fører hånden foran sensoren.

Koden var til testens formål redigeret således at den i stedet for at kalde en aktiveringsfunktion til kamera'et, blev der printet en meddelelse i Terminalen for at vise resultatet som set i den vedhæftede [video](#).

Overgang til RPI 4B fra RPI Pico

¹⁰ Bilag 3. Successful sensor test video.

Ovenstående tests blev udført på en RPI Pico, grundet gruppens begrænsende antal af RPI 4B. Da undertegnede skulle teste sensorens funktion på RPI 4B skulle der små ændringer til i koden, for at koden var kompatibel til Raspberry Pi 4B. Denne nye version af koden fejlede som udgangspunkt, og det var først med hjælp fra vejleder Jonas Wehding, at sensoren kunne opfange en ændring i Input på samme vis som da sensoren var tilsluttet en RPI Pico.

5.3 Database & Web (SHK, HJL)

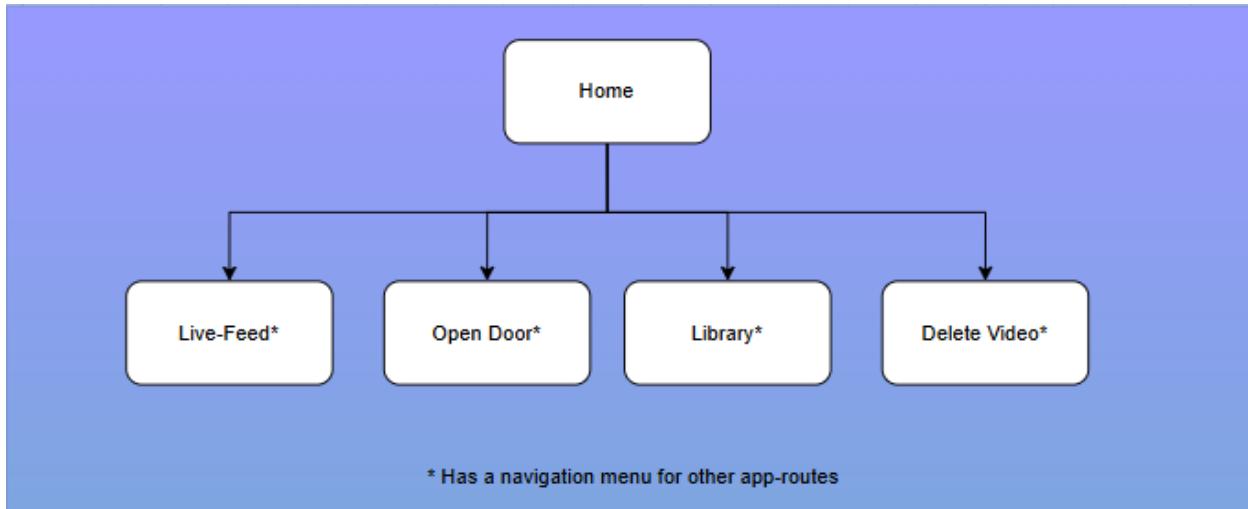
Oversigt, folderstruktur (HJL).¹¹

```
.
├── __pycache__/
│   └── face_camerasytem.cpython-311
├── bin/
│   ├── activate
│   ├── activate.csh
│   ├── activate.fish
│   ├── activate.ps1
│   ├── f2py
│   ├── flask
│   ├── pip
│   ├── pip3.11
│   ├── python
│   ├── python3
│   └── python 3.11
├── env/
│   └── static/
│       └── videos
├── include/
│   └── python3.11
├── lib/
│   └── python 3.11/
│       └── site-packages/
│           ├── distutils_hack
│           ├── blinker
│           ├── click
│           ├── cv2
│           ├── ffmpeg
│           ├── flask
│           ├── flask-3.1.0.dist-info
│           ├── fureture
│           ├── jinja2
│           ├── libfuturize
│           ├── libpasteurize
│           ├── markupsafe
│           ├── numpy
│           ├── opencv_python.libs
│           ├── past
│           ├── pip
│           ├── pkg_Resources
│           ├── psycopg2
│           ├── RPi
│           ├── RPi.GPIO-0.7.1.egg-info
│           ├── setuptools
│           └── werkzeug
├── static/
│   ├── images/
│   │   └── logo
│   ├── styles/
│   │   └── stylemedfoto
│   └── videos/
│       └── 2024-12-04_09-16-39
├── templates/
│   ├── bibliotek
│   ├── deletevideo
│   ├── home
│   ├── livefeed
│   └── opendoor
├── akmodel.yaml
├── app.py
├── env.startup.service
├── face_camerasytem.py
├── haarcascade_frontalface_default.xml
├── main.py
├── master.py
├── pir.py
├── pyvenv.config
├── startup.sh
└── test.py
└── uploadvideo.py
```

[View the source](#)

¹¹ [URL til oversigt over folderstrukturen.](#)

Site map:



5.3.1 Modul Design

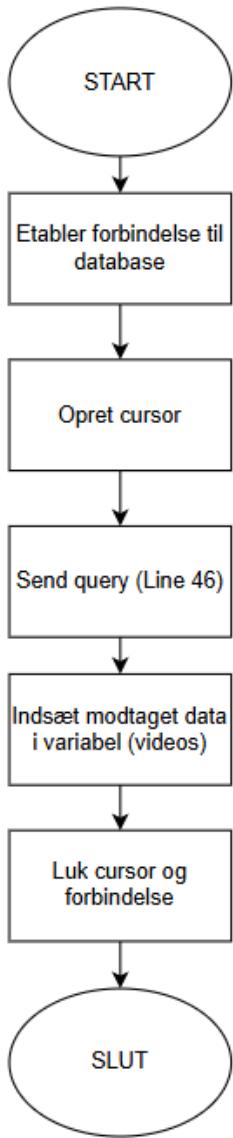
HTML/Flask moduldesignet er blevet opbygget på baggrund af de kriterier, som der stilles til produktets egenskaber. Produktets egenskaber som er relevant for HTML består af:

- Home / Welcomesite
- Library (Visning og afspilning af optagelser)
- Live-Feed (Placeholder til evt. Live-Feed)
- Open-Door (Visuelt output af døråbningsfunktion)
- Delete Video (Fjernelse af video fra database & RPI)

Alle disse egenskaber er blevet taget i betragtning i udarbejdelsen af HTML Front & Backend.

Kravet til designet bestod af at man skulle kunne navigere sig rundt i hver eneste route, f.eks. at det er muligt at tilgå "Library" fra "Delete Video". Et yderligere krav til hjemmesiden er også at det skulle være brugervenligt, og let at overskue som bruger. Derfor har designet forholdet sig til at bruge let navigationsbar, samt ved så vidt muligt at have gjort brug af en fælles css-style, så der er en rød tråd mellem alle routes/HTML.

5.3.2 Modul implementering



Som nævnt i ovenstående afsnit, afsnit 5.5.1, bliver der benyttet flask som host til projektets website. Flask er blevet benyttet på baggrund af dens nemme syntaks, og at undertegnet allerede har dannet sig erfaring med flask fra tidligere projekter. Flask er et micro-web-framework, hvilket efter opsættelse hoster en server på den specifikke host og dens tilkoblede netværk. Dette gør det muligt for andre på samme netværk at kunne tilgå denne hjemmeside og benytte dens html-funktionalitet som bruger. I flask bliver der gjort brug af html, og disse templates bliver hentet/benyttet ved hjælp af Jinja2, som er indbygget i flask-modulet.

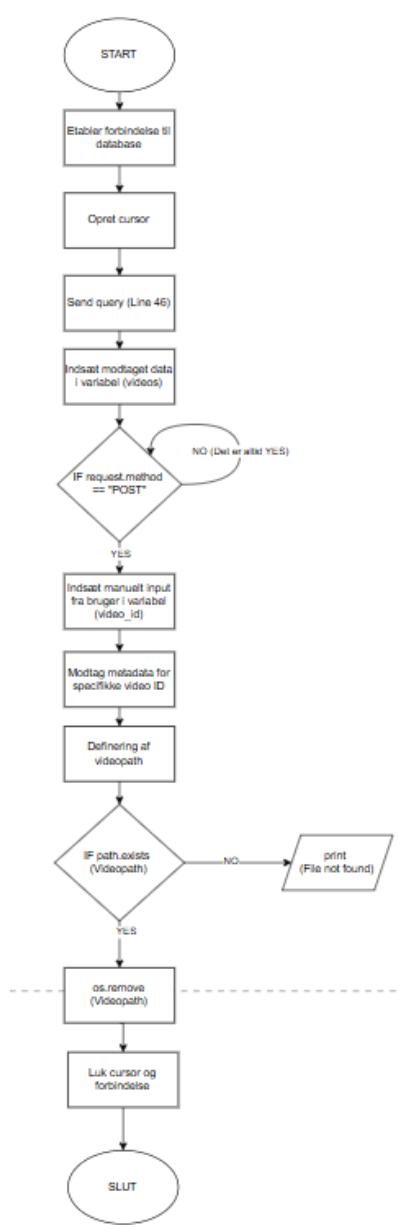
I hjemmesidemodulet findes der 6 forskellige app-routes, som hver især returnerer sin helt egen HTML side.

Første app-route består af en hjemmeskærm. Dens formål er ganske enkelt at vise en navigationslinje som giver links til andre funktioner/html.

Næste app-route er et database-bibliotek (Library). Hertil er der oprettet forbindelse til Postgres database, bestående af metadata som er videreført fra RPI4B. Her benyttes psycopg2 til at skabe forbindelsen, samt oprette en cursor, hvilket giver mulighed for at kunne tilsende en SQL Query. Dette metadata som returneres via. SQL Query bliver lagt på en variabel med navnet videos, hvorefter HTML trækker det lagrede data fra denne variabel (videos). Herefter bliver der lavet en visuel tabel i library.html, hvilket selv varierer ud fra antallet af poster, som videregives fra Postgres. I denne tabel vises der 3 forskellige headers; "Video ID", "File Name", "Hyperlink". Disse 3 headers er til for at give brugeren overblik over de filer, som både ligger på RPI4B og Postgres database. I den tredje header er der en href; som viderefører brugeren til næste app-route. Flowchart af denne kan ses på venstre side.

Brugeren bliver videreført til denne specifikke app-route, hvilket er play_video. Play_video har den overordnede funktion at skulle kunne lave/videreføre hyperlinks, som gør det muligt for den lokalt lagret video på RPI4B at blive afspillet over dette specifikke hyperlink. Dette specifikke hyperlink bliver lavet med information fra HTML-tabellen, som stammer af metadataen fra postgres databasen. Her bliver hyperlinket oprettet ved hjælp af filens navn (filename), som bliver placeret til slut i app-routen. For at få fat i denne video skal vi gøre brug af modulet "OS". OS hjælper med at skabe forbindelse/styre RPI4B's styresystem. Dette skal benyttes for at man kan trække det data, som er lokalt laget på RPI4B i form af .mp4 filer, hvilket endeligt gør det en mulighed at afspille disse. Der bliver dernæst foretaget en statisk søgning i et bestemt directory i

RPI4B OS (/static/videos), hvert tilfælde brugeren får brug for at skulle afspille en video. Når denne statiske søgning bliver foretaget, sker der kort efter en undersøgelse om den specifikke fil, med det bestemte filnavn, findes i det statiske directory. Hvis den efterspurgte fil findes i mappen, bliver den sendt fra directory og bliver returneret til den bestemte URL, hvilket kan afspille selvsamme video. Hvis filen derimod ikke findes i mappen, returnerer den en enkelt error-message indeholdende at filen ikke kunne findes.¹²



Brugeren skal selvfølgelig og have mulighed for at slette en video fra listen, hvis denne unødvendigt er blevet optaget og lagret. Dette bliver gjort for både at spare lokalt lager, men også for at spare plads på postgres databasen. Dertil er app-routen “delete_video” blevet gjort tilgængelig. I denne app-route gøres der brug af både OS og Psycopg2 modulerne.

Psycopg2-modulet benyttes til at der igen skal laves en SQL Query til postgresdatabasen. OS-modulet bliver i modsætning til ovenstående app-route brugt til at fjerne fremfor at hente en videofil. Brugeren får til start vist en oversigt over de angivne filer der findes i postgres og RPI4B, netop ved hjælp af psycopg2. Her gives bruger mulighed for at indtaste et manuelt input af “Video ID”, som ønskes slettes.

Begrundelsen for at have benyttet video ID som variabel for at slette, er grundet at filnavne kommer til at bestå af lange timestamps, hvilket vil være besværligt at bruge som manuelt input i submit-formen. Dette manuelle input er blevet opsat som en “input-form” i deletevideo.html, så bruger får mulighed for at udføre app-routens funktion.

Bruger har nu besluttet sig for at slette en given video og submitter formen med at trykke på “delete” submit knappen. Dette igangsætter fastsættelse af variablen “video_id”, som svarer til det ID som brugeren har valgt at slette. Denne variabel benyttes videre til at finde det filnavn i postgres, som video ID matcher til, for at den specifikke kan vælges og benyttes som sin egen variabel kaldet “result”. Dette “result” variabel skal bruges i fjernelsen af filen på RPI4B, ved brugen af modulet OS. Efter fastsættelse af denne variabel bruger vi en lignende opsætning

af modulet OS, som tidligere brugt i “play_video” app-routen.

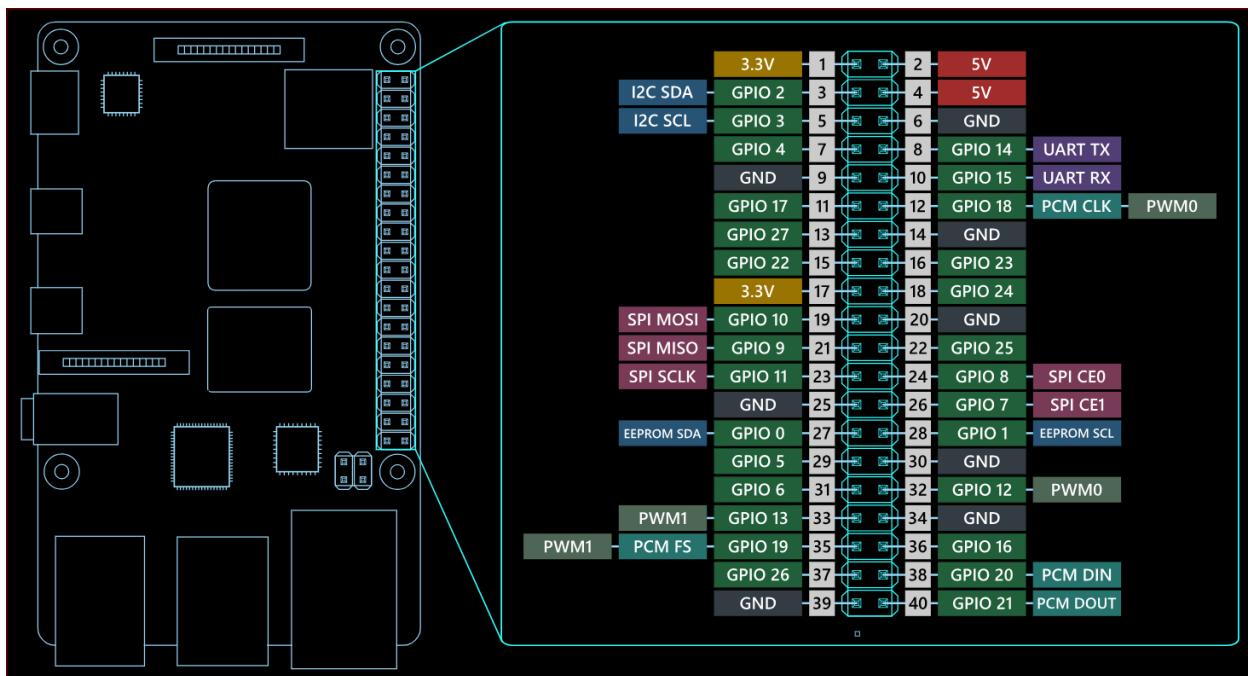
¹² Bilag 9.7 – Testvideo af uploadvideo og app.py – Kapitel: ”Visning af video”

Der bliver undersøgt af OS om der i den specifikke directory (/static/videos) indeholder en fil med det valgte filnavn. Filen bliver fjernet ved hjælp af OS' remove funktion, og er dermed blevet fjernet fra det lokale lager på RPI4B og dernæst fjernet ved postgres database gennem en SQL Query. Flowchart over routen delete video findes på ovenstående side¹³.

LED Døråbning (HJL)

Vi har under projektet måtte indse, at der grundet tidspres ikke kan indfries det fulde omfang i form af, at låse en dørlås op ved hjælp af et elektronisk signal sendt fra hjemmesiden gennem Raspberry Pi'en.

For at simulere denne "døråbning", har vi i stedet opsat en test. Vi har tilføjet en blå LED, og sat denne til Pin GPIO 17 samt GND(39) på RPI'en. Den blå LED skal simulere en elektronisk dørlås.



Testen er til skal vise, at brugeren gennem hjemmesiden kan tænde og slukke LED'en – og herved simulere at låse op for, og låse en dør.

For at implementere denne funktion, er der tilføjet GPIO.setup til app.py. Ligeledes er der under app.routen "/opendoor" samt opendoor.html-templetten tilføjet, at når man interagere med "Open Door" knappen, så kaldes skifter LED'en tilstand.

¹³ Samlet kode for Database & Web kan findes respektivt i bilagsafsnit 7 & 8

For at se koden for dette, se .html "Opendoor" og app.py filerne.

Hermed har vi forsøgt at simulere en "døråbning" vha. af en ekstern LED. Tryk på hyperlink for youtube video af en positiv test af aktivering af LED'en gennem web-applikationen.

[Hyperlink](#)

5.3.3 Formel modultest

I begyndelsesfasen af HTML og Flask opsætning, blev der begyndt med at få styr på den enkle opsætning af flask. Ved test af dette forekom der ingen funktionelle eller visuelle problemer. Ved oprettelse af style.css, som skulle gengives i alle app-routes, opstod der mindre problemer, men som blev løst relativt hurtigt.

Der er efterfølgende blevet løbende opdateret i de forskellige app-routes, her forekom der både visuelle og funktionelle fejl. De funktionelle fejl opstod som oftest af at der fandtes syntaksfejl i opsætningen af app.py kode, specifikt i "play_video" app-routen. Dette gjorde at den specifikke video som skulle afspilles, ikke kunne "fetches" af flask modulet. Syntaksfejlene blev hurtigt løst og dermed fungerede app-routen og videoafspilning optimalt efterfølgende.

Visuelle fejl opstod i HTML opsætningen og dette havde betydning for brugervenligheden. Første iteration af HTML kode, bestod af postgres tabel som blev oprettet med for mange headers, hvor disse headers var inkluderet: Video ID, Filename, Filepath, Created at.

Det blev ved test hurtigt åbenlyst at der ikke var nødvendigt med disse 2 headers: Created at & Filepath. Informationen for hvornår filen blev lavet, ville blive samlet i filnavnet inden upload af video til databasen, hvilket ville betyde at vi ville have den ens information i samme tabel. Det blev tydeligt at der ikke var behov for en filepath, da dette ikke var brugbar viden for slutbruger, samtidig med at flask modulet ikke havde behov for denne filepath, da en statisk givet directory var givet på forhånd i flask-modulet.

5.4 Main & Mastersoftware (SHK, HB, HJL)

I dette modul vil der blive fortalt om opsætning, implementering og test af det samlede styringssystem. I dette afsnit bliver der foretaget system/integrationstest, da dette modul er sammenhængende med hele systemet.

5.4.1 Modul Formål og Design

Master modulet har til opgave at fungere som et styringsmodul, der "samler trådene" for vores forskellige moduler. Modulet overvåger systemets funktioner, herinde den passive infrarøde sensor, som skal opfange en ændring i infrarød stråling – en person – hvilket aktiverer resten af systemet.

Master modulet har til opgave at fungere som det centrale kontrol- eller styringspunkt der sikrer, at de forskellige software- og hardwarekomponenter samarbejder som tiltænkt og på en struktureret måde.

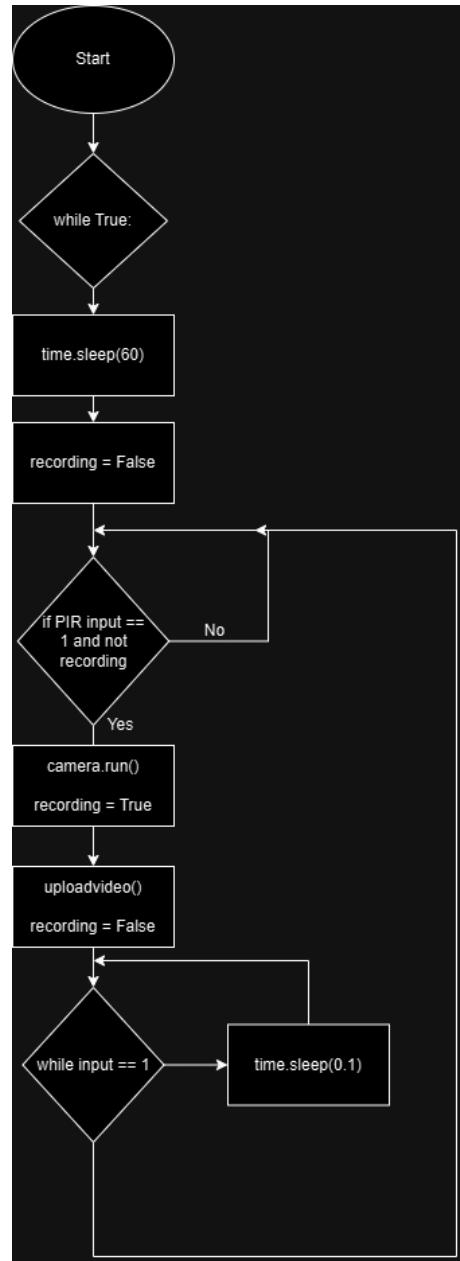
I designet af softwaren er der prioriteret en struktur, hvor der er en klar opdeling mellem main og master softwaren. Main-softwaren er bevidst holdt så simpel som muligt og fungerer udelukkende som en indgang, der kalder master softwaren¹⁴. Master-softwaren, hvilket er implementeret i filen master.py, fungerer derimod som en central styringsenhed, der forbinder de forskellige moduler i systemet.¹⁵

5.4.2 Modul implementering

Implementeringen af softwaredesignet har været fokuseret på at samle alle de nødvendige komponenter på RPI 4B og sikre en korrekt opsætning af master softwaren. For at understøtte systemets funktionalitet har vi etableret et Virtuelt Miljø som er Python-baseret, hvor alle relevante biblioteker og afhængigheder er installeret. Dette inkluderer RPi.GPIO til hardwarestyring, OpenCV til ansigtsgenkendelse, psycopg2 til databaseinteraktion og ffmpeg til videooptagelse mm.

Integration af de forskellige moduler har været en central del af denne implementering.

PIR-sensoren er opsat med GPIO-konfiguration i BCM-mode, hvilket muliggør registrering af bevægelse gennem signaler. Ansigtsgenkendelsen er implementeret med OpenCV og anvender både en Haar-cascade og en LBPH-model for at skelne mellem kendte og ukendte ansigter. Hvis et ukendt ansigt detekteres, starter systemet en videooptagelse via FFmpeg, hvor video og lyd bliver lagret som en MP4-fil. Efter optagelsen overføres filen til en database med tilhørende metadata som filnavn og tidsstempel.



¹⁴ Bilag 9.4 – main.py

¹⁵ Bilag 9.3 – master.py

Under implementeringen er der lagt vægt på at minimere fejl og sikre robusthed i softwaren. GPIO-cleanup er for eksempel integreret for at undgå problemer med pin-konfiguration, når programmet afsluttes. Derudover anvendes try/except til at håndtere uforudsete fejl, såsom manglende hardwaretilslutning eller databaseforbindelser. Denne tilgang sikrer, at softwaren er stabil og tilpasset til fremtidige mulige problemer.¹⁶

Implementering af Startup Script

Implementeringen af startup scriptet har været en vigtig opgave for at sikre, at systemet kan fungere selvstændigt ved opstart uden behov for manuelle input. Løsningen blev opbygget omkring to centrale komponenter: et Bash-script kaldet startup.sh og en systemd-service env_startup.service. Sammen danner de en automatiseret process, der starter både masterprogrammet (main.py) og webserveren (app.py) i hver sin terminal.

startup.sh blev designet til at aktivere det virtuelle miljø og starte de nødvendige Python-scripts. For at sikre, at begge programmer kunne køre parallelt uden interferens, blev de startet i separate terminaler. Denne tilgang var essentiel for at håndtere de forskellige funktioner i systemet, som både kræver realtidsovervågning af bevægelser og drift af en webserver.

Systemd-servicen, env_startup.service, blev implementeret for at automatisere processen ved opstart. Den sørger for, at startup.sh kører korrekt som en del af systemets opstart. Vi sikrede os, at servicen blev tilpasset det grafiske miljø, da det var nødvendigt for at understøtte åbningen af terminalvinduer via Ixterminal kommandoen.¹⁷¹⁸¹⁹

5.4.3 Formel modultest

Efter færdiggørelsen af en første version af master softwaren blev der udført en række tests for at afprøve systemets funktionalitet og identificere fejl. Disse tests dækkede både software- og hardwarekomponenter for at sikre en helhedsorienteret validering.

Under testforløbet blev der udført flere funktionelle tests for at sikre, at master softwaren korrekt håndterede interaktionen mellem de forskellige moduler. Dette inkluderede blandt andet test af, at videooptagelser blev startet korrekt, og at upload til databasen blev gennemført uden fejl. Et vigtigt fokusområde var at sikre, at metadata som filnavn og tidsstempel blev lagret korrekt i databasen.

Hardwaretesten omfattede verificering af PIR-sensorens funktionalitet, hvor vi stødte på problemer med forkert pin-konfiguration, hvilket førte til fejlaflæsninger. Dette blev løst ved at justere til den korrekte GPIO-pin. Ligeledes opstod der problemer med ansigtsgenkendelse i

¹⁶ Bilag 9.3 – master.py

¹⁷ Bilag 9.4 – main.py

¹⁸ Bilag 9.5 – startup.sh

¹⁹ Bilag 9.6 – env_startup.service

starten, hvor en for høj distance på ansigtet resulterede i manglende detektion af ukendt ansigt. Dette blev udbedret ved at justere parameterne for detektion. Desuden opdagede vi under de første tests, at FFmpeg ikke var korrekt konfigureret, hvilket blokerede videooptagelsen. Ved at justere video device (" -i ", "/dev/video0") og audio device (" -i ", "alsa" og " -i ", "plughw3,0") i FFmpeg kommandoen, stødte vi på et nyt problem – Linux tager imod stereolyd, men kameraets mikrofon kan kun sende mono, så vi opkonfigurerede igen FFmpeg kommandoen til at tage imod mono lyd (" -ac ", "i ")

De gennemførte tests har været essentielle for at sikre, at systemet fungerer som tiltænkt. Endvidere er en kontinuerlig proces mellem at teste og dokumentere nødvendig for at optimere sit færdige produkt og opdage sammenhænge mellem virkeligheden og det nedskrevne.

Test af startup script

Under testforløbet stødtes der på flere udfordringer, som krævede justeringer i både scriptet og servicen. En af de primære problemer opstod, fordi det virtuelle miljø ikke blev aktiveret korrekt. Dette skyldtes en fejl i stien til miljøet, som blev rettet ved at verificere og opdatere den specifikke placering i scriptet..

En anden udfordring var relateret til grafiske miljøvariabler, som ikke blev indlæst korrekt af systemd-servicen. Dette resulterede i fejl, hvor terminaler ikke kunne åbnes. Problemet blev løst ved at specificere nødvendige miljøvariabler som *DISPLAY* og *XDG_RUNTIME_DIR* i servicefilen. Endelig krævede scriptet en tilpasning for at sikre, at det forblev aktivt og ventede på afslutning, i stedet for at stoppe umiddelbart efter opstart.

De nævnte test og dertil rettelser betyder at systemet nu fungerer, og opstarten sker som forventet.

5.4.4 Videreudvikling

Ved at have centraliseret styringen af systemet i ét Master modul holder vi muligheden åben for på nemmere vis, at kunne videreudvikle på det samlede produkt fremover. Ved fra start at have designet og senere udviklet modulet efter et fleksibelt og modulært princip, understøttes implementering af yderligere funktioner herved på nemmere vis i fremtiden.

5.5 Lyd optagelse og afspilning – ‘audio_handling’ (SØF)

Eftersom systemet de første uger benyttede sig af et kamera uden mikrofon, var det nødvendigt at have en måde at kunne optage audio på parallel med video. Video og lyd skulle så samles efter endt optagelse, og lagres, eller sendes sideløbende over en live peer-to-peer forbindelse. Systemet skulle også kunne afspille lyd, så det kunne fungere som en slags intercom. Denne håndtering af lyd in- og output skulle software-modulet ‘audio_handling’ stå for..

For at løse denne opgave på et proof-of-concept-niveau benyttede vi en Edutige mikrofon med 3.5mm jack output, som er tilkoblet en Antlion Audio USB-A adapter som audio input. Til vores output benyttedes et sæt Logitech Z200 højtalere med ekstern strømforsyning. Mikrofo

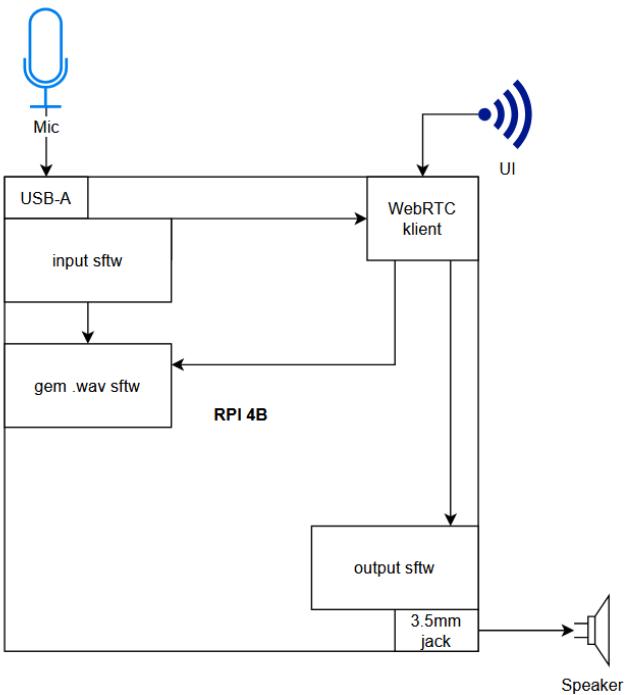
nen og højtalerne blev bestemt især pga. deres tilgængelighed, idet vi med disse løsninger ikke skulle bekymre os om at læse I2S output, håndtere DAC eller ADC eller forstærke output til højtalerne. I stedet kunne vi fokusere på at behandle in- og output-signaler således de kunne lagres og streames efter behov.

Audio_handling blev et færdigudviklet modul, i stand til at håndtere audio in- og output, men endte med ikke at blive integreret i denne iteration af slutproduktet. Eftersom audio_handling har opfyldt et krav vi satte i starten af projektet, og indtil projektets afslutning blev forsøgt integreret som en del af to-vejs kommunikation er modulet inkluderet i rapporten.

5.5.1 Modul design

Overordnet om modulet

Som det fremgår af figuren herunder består modulet på et overordnet niveau af en mikrofon, en højtalere, en Raspberry Pi samt software til at håndtere input og output i forbindelse med lagring og læsning af filer. Derudover findes også software til live streaming af filer, idet transmittering af live-data skal håndteres anderledes end hvis det blot skal lagres eller læses. I modulet benyttes især to vigtige libraries til python: PyAudio og Wave.



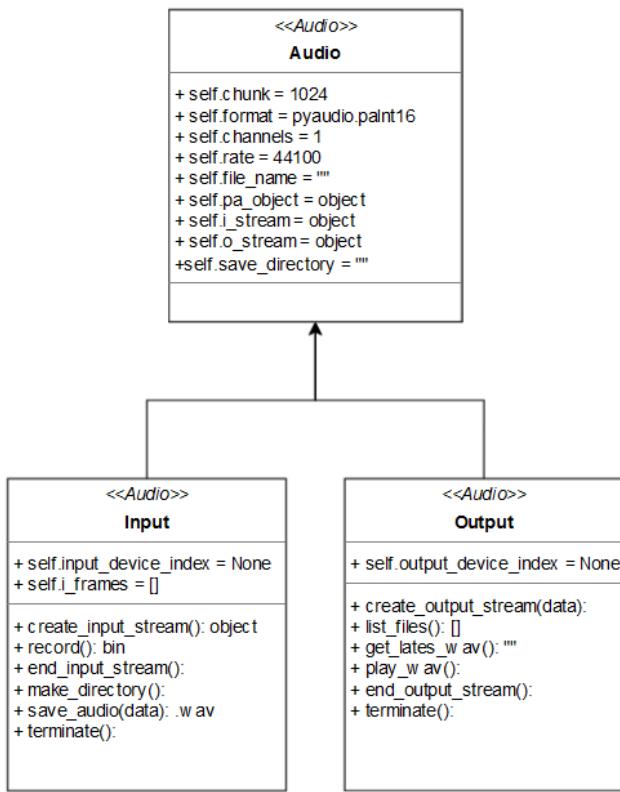
Nogle vigtige nøglepunkter:

- Systemet skal kunne optage og lagre lydfiler til senere afspilning gennem UI
- Systemet skal kunne læse lagrede medier og videregive disse til UI
- Systemet skal ikke selv afspille lagrede medier. Selve afspilning af lagrede medier skal ske gennem UI.
- Systemet skal kunne optage live data og afsende dette til en bruger
- Systemet skal kunne modtage og afspille live data fra en bruger

Som det fremgår af den understregede tekst ovenover er dette moduls primære funktion at optage og afspille lyd. Det skal dog kunne indgå i et tæt samarbejde med vores UI og et Real Time Communication modul, for at håndtere live data gennem en direkte P2P forbindelse.

For at sikre overskuelighed, modularitet og fleksibilitet er audio_handling modulet skrevet ved brug af OOP. En yderligere årsag er blot for øvelsens og interessens skyld.

UML diagrammet herunder beskriver modulets klasser: 'Audio', 'Input' og 'Output', samt deres attributter og metoder.



Audio

Overklassen, 'Audio' er her brugt til at tildele attributter, som 'Input' og 'Output' deler.

At skrive og læse lyd er en tung proces, i hvert fald hvis hele lydfiler skal skrives og læses i én omgang. Derfor deles filerne op i 'chunks', som lader software læse en lille bid af lyd ad gangen.

Chunks er samlinger af frames, som igen er samlinger af bytes. Af UML-diagrammet ovenover fremgår det, at chunks har en størrelse på 1024, altså indeholder en chunk 1024 frames. Størrelsen af disse frames er bestemt af lydfilens format samt antallet af spor i lydfilen - altså om filen er mono, stereo eller en form for surround-kompatibel. I dette tilfælde bruger vi kun mono-lyd, idet vi kun har én mikrofon tilgængelig, det sparer ressourcer, og stereo-lyd ikke er relevant ifht. to vejs kommunikation.

Formatet på en lydfil bestemmer hvor mange bytes en frame består af. I dette tilfælde er 16-bit format valgt, hvilket betyder at én frame er 2 bytes pr. spor.

Tre andre vigtige attributter i ‘Audio’ er ‘pa_object’, ‘i_stream’ og ‘o_stream’.

Pa_object bruges af både ‘Input’ og ‘Output’ sub-klasserne til at terminere instanser af disse objekter gennem terminate() metoderne, således at de ressourcer, der bruges på objekterne kan frigives når ikke længere de er nødvendige.

‘I_stream’ og ‘o_stream’ er instanser af PyAudio objekter, såkaldte ‘streams’ som data skrives til eller læses fra²⁰. ‘i_stream’ står således for ‘input stream’, og ‘o_stream’ for ‘output stream’. Disse burde være ‘Input’ og ‘Audio’ specifikke, idet et input ikke behøver en output stream, men undertegnede er blot en ølle, dødelig 2. semester studerende, der kunne ikke få programmet til at virke uden denne opsætning. Disse streams oprettes af hhv. create_input_stream() og create_output_stream() , og lukkes af end_input/output_stream() metoderne under ‘Input’ og ‘Output’ klasserne

Se flowcharts og kode i ‘**Bilag 10 - audio_handling**’

Input

‘Input’ har to attributter: ‘input_device_index’, som, når sat til None, angiver at programmet benytter sig af den default input-enhed. Det er så op til os at bestemme hvilken enhed, der benyttes som default af systemet.

‘i_frames’ er en liste over frames fra input-enheden. Under optagelse, via record_to_file(), lægges chunks af frames løbende over i denne liste. Når filen skal gemmes lukkes input streamen af end_input_stream(), og disse frames konkatineres derefter, og skrives over til én samlet .wav fil af save_audio()²¹.

Se flowcharts og kode i ‘**Bilag 10 - audio_handling**’

Output

‘output_device_index’ = None sætter output streamen til at spille lyd via den default lyd-udgangs enhed. Output streams oprettes af create_output_stream(), og afspilning af lagrede filer initieres af play_wav(), som skal have en filsti. List_files() og get_latest_wav() er legacy metoder fra test-

²⁰ <https://people.csail.mit.edu/hubert/pyaudio/docs/#class-pyaudio-stream>

²¹ <https://docs.python.org/3/library/wave.html#wave-write-objects>

fasen. Når afspilning er slut kan output streamen lukkes og termineres via end_output_stream() og terminate().

Se flowcharts og kode i '**Bilag 10 - audio_handling**'

5.5.2 Formel modultest

I begyndelsen af projektet blev audio_handling testet på en headless installation af Raspberry Pi OS på en Raspberry Pi, hvilket denne del af modultesten vil beskrive. I store træk er processen den samme med en Raspberry OS Desktop installation, og kan oversættes 1:1 gennem terminalen.

5.5.3.1 - Opsætning af default lydenheder og test af optagelse gennem terminal

Først blev default audio ind- og udgange sat op, ved at undersøge hver enheds card og device nummer:

Tilgængelige lydenheder:

Afspilning

```
gruppe2@raspberrypi:/etc $ aplay -l
**** List of PLAYBACK Hardware Devices ****
card 0: Headphones [bcm2835 Headphones], device 0: bcm2835 Headphones [bcm2835 Headphones]
  Subdevices: 8/8
  Subdevice #0: subdevice #0
  Subdevice #1: subdevice #1
  Subdevice #2: subdevice #2
  Subdevice #3: subdevice #3
  Subdevice #4: subdevice #4
  Subdevice #5: subdevice #5
  Subdevice #6: subdevice #6
  Subdevice #7: subdevice #7
card 1: vc4hdmi0 [vc4-hdmi-0], device 0: MAI PCM i2s-hifi-0 [MAI PCM i2s-hifi-0]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 2: vc4hdmi1 [vc4-hdmi-1], device 0: MAI PCM i2s-hifi-0 [MAI PCM i2s-hifi-0]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
gruppe2@raspberrypi:/etc $
```

Optagelse

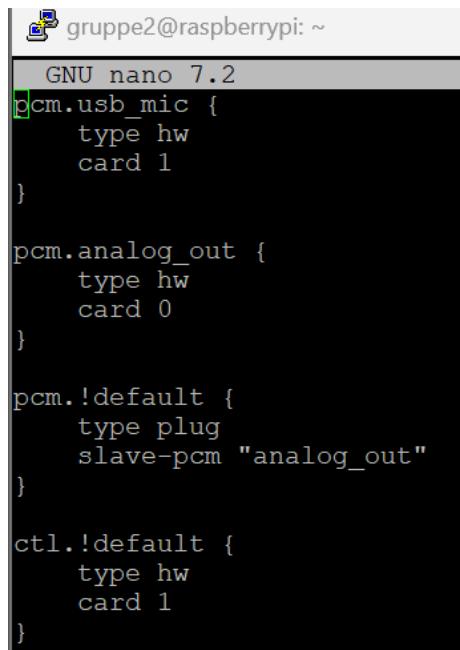
```
gruppe2@raspberrypi:~ $ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 004: ID 0d8c:002b C-Media Electronics, Inc. Antlion USB adapter
Bus 001 Device 003: ID 03f0:0024 HP, Inc KU-0316 Keyboard
Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
gruppe2@raspberrypi:~ $ arecord -l
**** List of CAPTURE Hardware Devices ****
card 3: adapter [Antlion USB adapter], device 0: USB Audio [USB Audio]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
gruppe2@raspberrypi:~ $
```

Her fremgår det af “aplay -l”, at afspilningsenheden Headphones [bcm2835 Headphones] har card 0 og device 0. “lsusb” viser at usb-a converteren til mikrofonen er forbundet (antlion USB adapter), og “arecord -l” lister denne optagelsesenhed med card 3 og device 0.

Disse cards og devices angives i `asound.conf`, så de indstilles som default devices:

Asound.conf:

BEMÆRK!! Dette screengrab er fra et tidligere tidspunkt hvor de to enheder havde andre card numre, mic havde card 1, og højttaler havde card 0. Disse synes at ændre sig efter genstart af Raspberry Pi'en, og kan risikere at skulle sættes op for hver genstart. Ikke desto mindre vil systemet genkende enhederne som default såfremt de rigtige cards angives i `asound.conf`



```
gruppe2@raspberrypi: ~
GNU nano 7.2
pcm.usb_mic {
    type hw
    card 1
}

pcm.analog_out {
    type hw
    card 0
}

pcm.!default {
    type plug
    slave-pcm "analog_out"
}

ctl.!default {
    type hw
    card 1
}
```

Efter opsætning af default devices er vi klar til at teste om disse genkendes gennem følgende kommandoer:

```
gruppe2@raspberrypi:~ $ arecord -D plughw:1,0 -f cd test.wav
Recording WAVE 'test.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
^CAborted by signal Interrupt...
gruppe2@raspberrypi:~ $ aplay test.wav
```

Denne test var succesfuld, og resultatet kan høres [i denne youtube short](#) (<https://youtube.com/shorts/bePsHWpeuz8?feature=share>).

5.5.3.2 - test af PyAudio til optagelse og lagring af lyd

Herefter tests hvorvidt PyAudio og Wave kan arbejde sammen i et python script, så lyd kan optages, gemmes, indlæses og afspilles. Til dette blev et simpelt script skrevet, som først instansierer en input- og output stream med de nødvendige parametre (det var dog kun input streamen, der blev brugt):

```
# Audio settings
chunk = 1024
format = pyaudio.paInt16
channels = 1
rate = 44100
output_file = "recorded_audio.wav"
```

```
# Åben USB mikrofon som input stream
input_stream = p.open(format=format,
                      channels=channels,
                      rate=rate,
                      input=True,
                      input_device_index=None) # Brug default input
```

Herefter oprettedes en frames-liste, og scriptet blev sat til at optage i 5 sekunder:

```
# Opbevaring af optagede chunks
frames = []

try:
    for i in range(0, int(rate / chunk * 5)):
        data = input_stream.read(chunk)
        frames.append(data)
```

Her er lavet lidt matematik for at angive hvor mange chunks der optages på 5 sekunder. Dette var nødvendigt. Ved brug af time.sleep(1) til tidstagnning ville scriptet optage 1 chunk, vente 1 sekund, og så optage 1 chunk mere, hvilket over 5 sekunder resulterede i en *meget* kort optagelse.

Herefter lukkes input og output streams:

```
finally:
    print("Finished recording...")

    # Stop og luk streams
    input_stream.stop_stream()
    input_stream.close()
    output_stream.stop_stream()
    output_stream.close()
```

Og lydfilen gemmes, med samme parametre som input streamen, og PyAudio instansen termineres:

```

# Gem optaget audio

# Definer hvilken fil, der skal skrives til, og hvilken type data, der skal skrives til den
wf = wave.open(output_file, mode: 'wb')
# Sæt antal kanaler på lydfil (mono)
wf.setnchannels(channels)
# Sæt sample-width (16-bit)
wf.setsampwidth(p.get_sample_size(format))
# Sæt framerate (44100)
wf.setframerate(rate)
# Skriv binære frames til filen
wf.writeframes(b''.join(frames))
# Luk filen
wf.close()

# Ryd op efter pyaudio, frigør ressourcer
p.terminate()

```

Denne testkode var udgangspunkt i designet af audio_handling, idet den gav et godt overblik over de nødvendige trin når der skulle optages og afspilles lyd. Den fulde første iteration af audio_handling kan ses i **Bilag 5 – audio_handling** under ‘Modultest 2’. Et script blev skrevet til at teste audio_handlings metoder (Se nedenunder). Dette script skulle først oprette en instans af audio_handling, åbne en input stream, optage 5 sekunder, pause 5 sekunder og optage 5 sekunder mere, og så til sidst gemme optagelsen. Derefter skulle scriptet teste audio_handling’s afspilning ved at oprette en output stream – kunne det lade sig gøre at afspille en .wav fil gennem audio_handlings metoder? Svaret var ja, men ikke uden fejl, som kan høres [i denne video](https://youtube.com/shorts/CiOW3jz5kL0?feature=share) (<https://youtube.com/shorts/CiOW3jz5kL0?feature=share>).

Dog opstod den samme fejl i afspilningen ikke ved brug af aplay i terminalen, som vises [her](#) (<https://youtube.com/shorts/wsa7WR2vAEc?feature=share>)

```

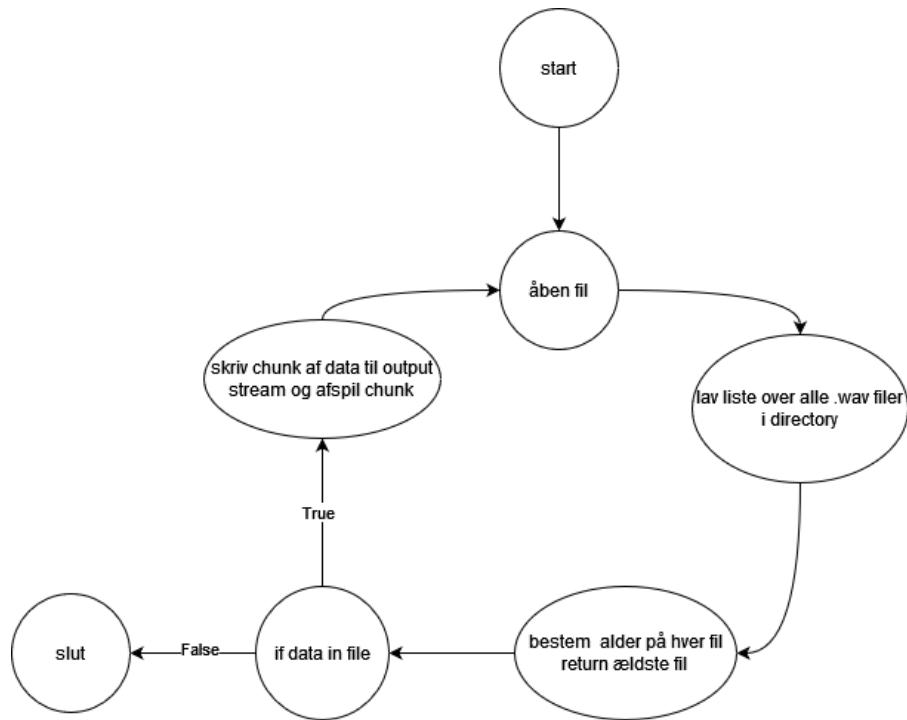
1 import audio_handling as audio
2 import time
3
4
5 # Opret instans af input objekt
6 input = audio.Input()
7 # Klargør "stream" til optagelse
8 input.create_input_stream()
9
10 rate = 44100 # Antal samples i sekundet
11 chunk = 1024 # Antal bits pr. sample
12
13 record = False
14
15
16 try:
17     # Optag i 5 sekunder
18     # (rate / chunk * 5 = chunks pr. sekund * 5 = 5 sekunders optagelse)
19     for i in range(0, int(rate / chunk * 5)):
20         print("start")
21         input.record()
22     # Pause optagelse i 5 sekunder
23     for i in range(5):
24         print("stop")
25         time.sleep(1)
26     # Optag videre i 5 sekunder
27     for i in range(0, int(rate/chunk*5)):
28         print("start")
29         input.record()
30     # Luk input stream, gem audio til .wav fil og terminér objektet
31     input.end_input_stream()
32     input.save_audio()
33     input.terminate()
34
35 except KeyboardInterrupt:
36     print("slut")
37
38 # Opret instans af output objekt og klargør stream til afspilning
39 output = audio.Output()
40 output.create_output_stream()
41
42 file = output.get_latest_wav()
43
44 # Afspil senest optagede fil, derefter luk stream og terminér objekt
45 output.play(file)
46 output.end_output_stream()
47 output.terminate()
48

```

Følgende er fra undertegnedes noter om processen:

"Netop her blev vi i gruppen opmærksomme på, at det AI kamera vi benyttede ikke er kompatibelt med Raspberry Pi OS Lite, som ovenstående program er udviklet og testet til. Undertegnede har valgt at stoppe al udvikling til RPi OS Lite, til fordel for Raspberry Pi OS med desktop (Debian Bookworm 12, 64bit). Eftersom der er mange udfordringer med opsætning af raspberry pi udviklingsmiljøet blev udvikling og nærmere undersøgelse af problemer med afspilning af lydfiler via ovenstående kode udsat. Undertegnedes bedste bud på årsagen til problemet er, at play()

metoden benytter sig af `get_latest_wav()`, som igen benytter sig af `list_files()`. Herunder ses et statediagram, der forklarer fejlen.



```
def play(self):
    with wave.open(self.get_latest_wav(), mode='rb') as wf:
```

```
def get_latest_wav(self):
    # Find den senest oprettede fil i 'wav_files'
    wav_files = self.list_files()
```

Disse processer tager tid, især på mindre kraftfulde systemer som en Raspberry Pi, og skaber derfor et "mellemrum" mellem hver af de afspillede chunks. Derfor vil næste logiske skridt være, at dykke ned i optimering af koden, således at disse "mellemrum" elimineres. Undertegnede vurderer dog, at andre opgaver er mere presserende, og et forsøg på en hurtig løsning er taget i brug, er testet, og afspiller fint lydfiler på en windows pc.

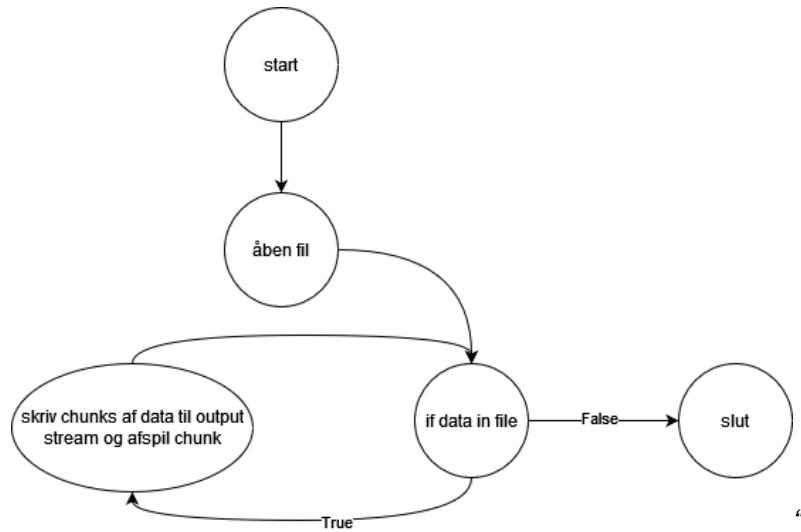
Her oprettes en instans af get_latest_wav() separat i main.py, som så passes som argument til play()

```
file = output.get_latest_wav()

# Afspil senest optagede fil, derefter luk stream og terminér objekt
output.play(file)
```

```
def play(self, file):
    with wave.open(file, mode: 'rb') as wf:
```

Dette statediagram viser det nye flow:



Dette var et udmærket læringspunkt, men havde ultimativt ingen relevans for det endelige produkt, idet systemet ikke skal kunne identificere den sidst oprettede fil. Dog er det inkluderet for at illustrere arbejdsmetoder og tankegange under test af modulet. Det var en fejl af undertegnede at bruge tid og ressourcer på at udarbejde løsningen til dette problem, og må tages som den ultimative læring af dette testforløb: "Hold dig til det, der er relevant for slut-produktet".

5.5.4 Refleksioner

OOP i audio_handling er ikke nødvendigvis skrevet til en guldmedalje, og det har heller ikke været intentionen. Undertegnede har valgt at bruge dette projekt som en mulighed for at prøve kræfter med OOP i en praktisk sammenhæng, og resultatet er overvejende positivt, idet OOP i denne sammenhæng tillader audio_handling at være en kasse af byggeklodser, som man kan sætte sammen efter ønske og behov.

I et færdigt produkt ville en integreret løsning til både mikrofon og højtalere være at foretrække, således at systemet kan eksistere som et enkelt, kompakt modul. Det kunne være værd at undersøge et MEMS mikrofon modul, som f.eks. et INMP441 modul, samt en AR027150MR-2-R højtalere fra PUI inc. med dertilhørende forstærker, som en effektiv indlejret løsning.

I starten af projektet syntes audio_handling at være en god tilgang til at løse flere problemer og opfylde flere krav på én gang, men efterhånden som projektets hardware komponenter ændrede sig, og vi i gruppen udvidede vores viden blev det stadigt klarere at audio_handlings sande formål var at tjene som et fartøj, der førte undertegnede én omgang rundt i den hermeneutiske cirkel. Udviklingen af audio_handling har tilbudt viden omkring behandling af input og output og audio data som helhed, men I et funktionelt eller endda færdigt produkt ville audio_handling ikke have nogen plads.

5.6 WebRTC samtale software og signaling server (SØF)

For at kunne opfylde projektets krav om en to-vejs kommunikation mellem system og en bruger, som f.eks. en pårørende af systemets ejer, valgte undertegnede at benytte sig af WebRTC, Web Real Time Communication.

Arsagerne til dette var, at WebRTC tilbyder super lav latens og høj medie-kvalitet, idet WebRTC benytter sig af en direkte peer-to-peer forbindelse. Andre løsninger vil kræve, at begge peers opretter forbindelse til en relay server, hvilket medfører højere latens, og kræver en del ressourcer for indehaveren af relay serveren. Den ypperste karakteristik af WebRTC er dog, at WebRTC kan implementeres direkte i HTML5 gennem javascript, og derfor ikke kræver en dedikeret klient for hver peer. Undertegnede besluttede dog at benytte AIORTC, som er et python bibliotek til WebRTC, bygget på asyncio. AIORTC blev valgt for at undertegnede ikke skulle lære et helt nyt programmeringssprog midt i projektet. Dette betød dog, som undertegnede først for sent fandt ud af, at hver peer ville kræve en dedikeret WebRTC klient.

5.6.1 WebRTC og signaling

Hvad er WebRTC?

WebRTc er et sæt javascript api's, der tillader brugere at skabe en direkte forbindelse, eller en peer to peer forbindelse. Herved kan data sendes direkte mellem de to (eller flere) peers, og man undgår den latens, der opstår når data først skal fra en peer til en ekstern server, og så videre til den anden peer. Denne direkte forbindelse er ideel for real tids kommunikation for at undgå forsinkelser.

WebRTC bruger UDP til at kommunikere. Takket være UDP's hastighed er protokollen velegnet til streaming, hvor man kan tillade en vis mængde ødelagte eller tabte pakker. UDP er derimod ikke god i situationer, hvor det er essentielt at de sendte og modtagne pakkers integritet er i højsædet. En sådan situation kan f.eks. Være når to peers skal udveksle SDP, Session Description Protocol, pakker, som de to benytter til at findes bl.a. fælles ICE-kandidater eller skabe enighed om anvendt codec til behandling af mediedata. Dette sker gennem en Signaling server, som ikke er en indbygget funktionalitet i WebRTC - det kan WebSockets derimod håndtere.

Hvad er Signalling?

Signaling er udvekslingen af den information, som er nødvendig for begge peers for at de kan oprette en direkte forbindelse mellem hinanden. Derudover udveksles ICE kandidater også gennem signaling.

Signaling er nødvendig, idet hver peer sidder bag en firewall, og deres adresser obfuskeres af NAT. Signaling benyttes til, ud over at udveksle SDP, også til at udveksle ICE kandidater, og disse sidstnævnte er særligt vigtige for at overkomme firewalls og NAT.

Hvad er en SDP?

En SDP er en beskrivelse af hver peers' forbindelse. En SDP indeholder f.eks. Information om en peers adresse, hvilken type medie, der skal overføres, hvilket codec mediet benytter, skal der behandles lyd eller video eller begge, etc.

Hvad er ICE kandidater?

ICE-kandidater er forbindelsespunkter, som to peers bruger til at skabe en direkte forbindelse. De kan bruge til at komme gennem NAT og firewalls. Forskellige ICE-kandidater har forskellige egenskaber, og bruges i forskellige situationer. WebRTC benytter sig af 'trickle ICE', hvor ICE-kandidater sendes mellem peers gennem signaling serveren efterhånden som de opdages og valideres. AIORTC, derimod, samler alle de mulige ICE-kandidater *før* SDP-pakker afsendes, og inkluderer kandidaterne heri.

5.6.2 Modul design

5.6.2.1 Signaling server

Signaling serveren er bygget op omkring Flask og Flask-SocketIO til håndtering af forbindelser og registrering og emitting af events. Signaling serveren benytter event-baseret logik, hvor forbundne peers og serveren udveksler events, som serveren og de forbundne peers så reagerer på.

Til serveren benyttes følgende moduler:

- flask (Flask, request)
- Flask_socketio (socketio, emit)
- Uuid

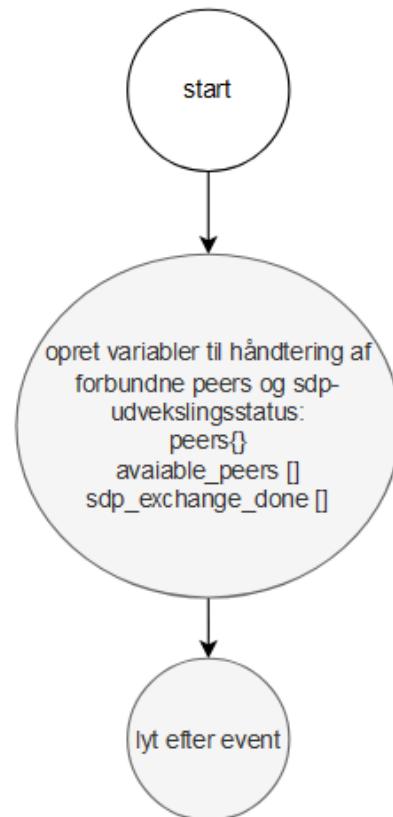
Den komplette kode kan findes i bilagene

Når serveren initialiseres, opretter den en række variabler til at holde styr på hvem der er tilsluttet, og hvor mange, der er færdig med at udveksle SDP pakker.

I slutningen af denne fase begynder serveren at lytte efter events. Det første event, serveren vil modtage, vil som regel være "connect". Når serveren registrerer et "connect" event, vil den tildele den nyligt forbundne peer et unikt UUID4 ID, kombineret med peer'ens session ID, og overskrive disse som hhv. Key og value til et peers{} dictionary. Derefter emitter serveren et "peer_id" event.

"peer_id" eventet har to formål:

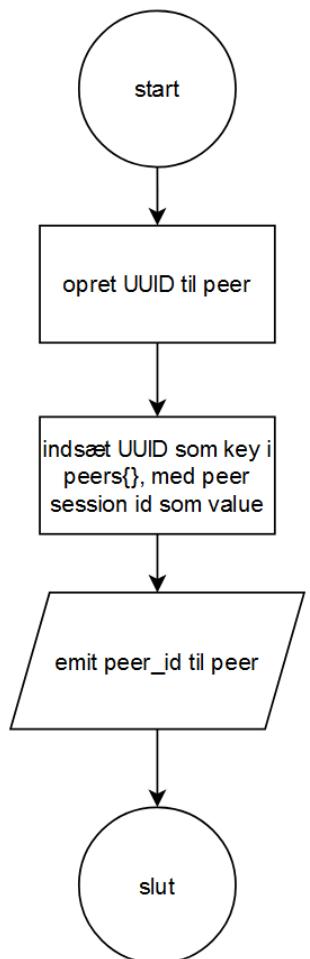
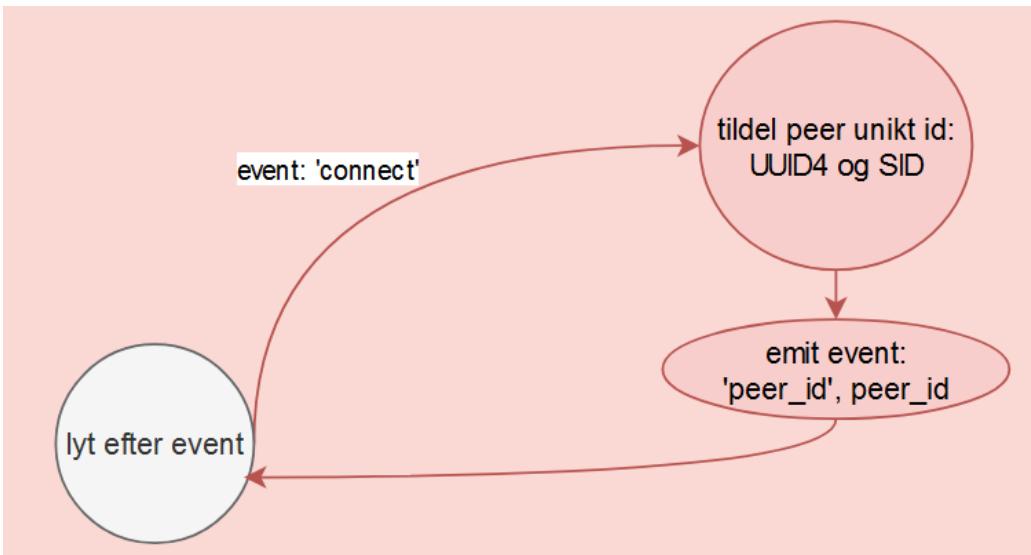
1. at oplyse en peer om dens eget ID
2. at oplyse en peer om andre peers' ID

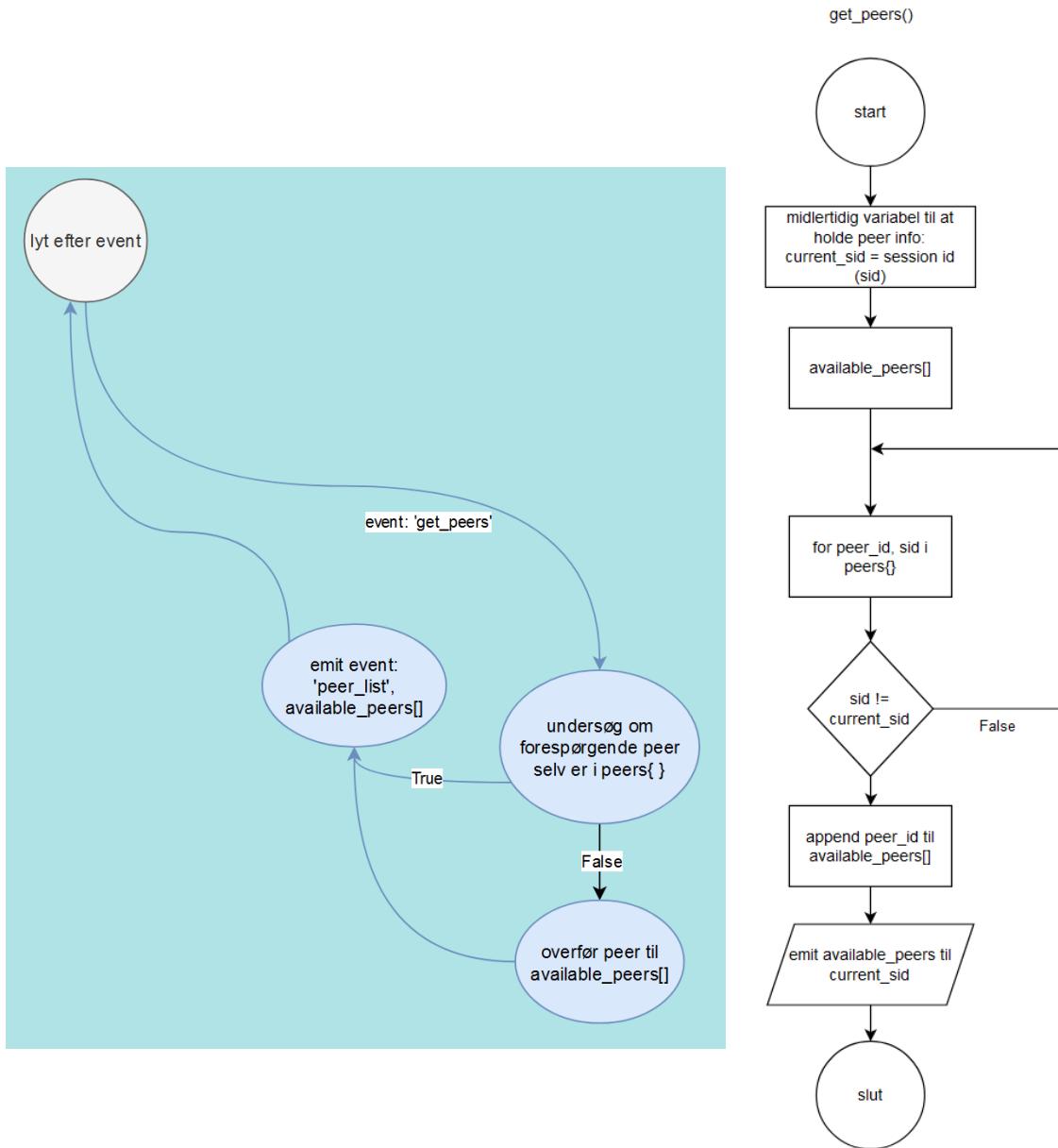


on_connect()

Når en peer forbindes til serveren vil hver peer også modtage et "connect" event. Herefter vil peer klienten udsende et "get_peers" event, for at få en liste over andre muligvis forbundne peers.

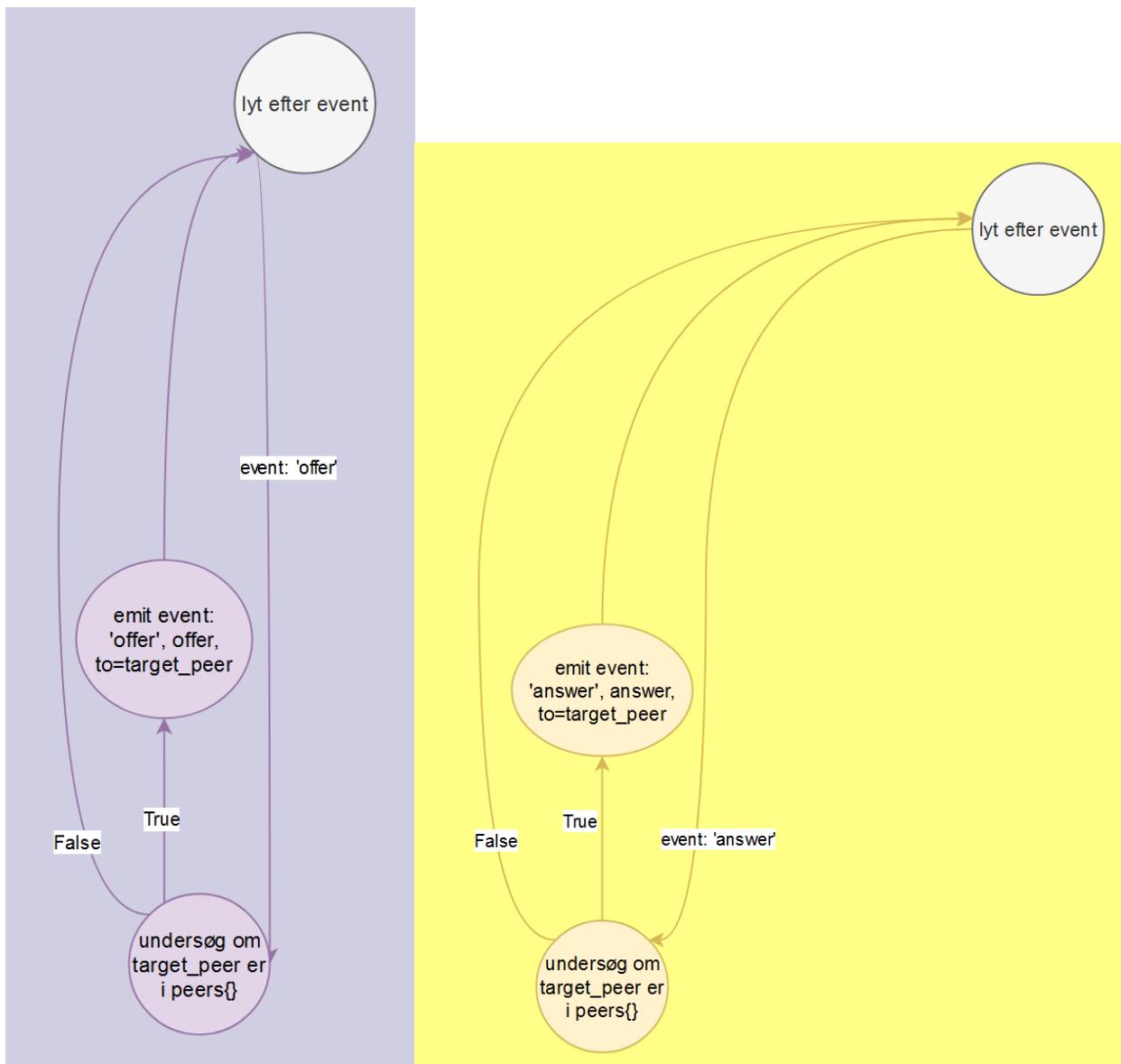
Når serveren modtager et "get_peer" event vil den opdatere listen available_peers[], og sende denne til forespørgeren, således at den forespørgende peer ikke vil se sig selv i available_peers[]



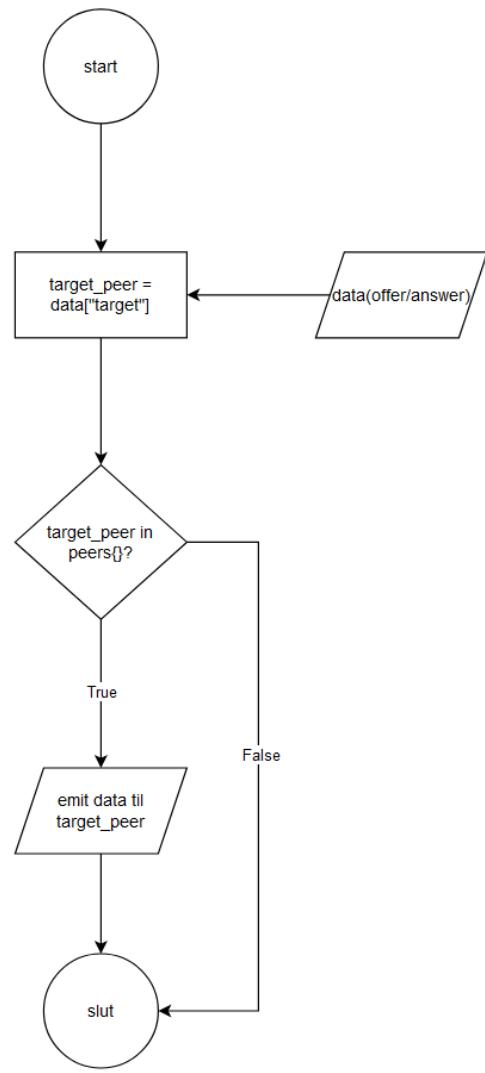


Den peer, der forbinder til serveren først vil således få en tom liste over tilgængelige peers, og derfor ikke reagere. Den næste peer, der forbinder til serveren vil dog se en liste med én tilgængelig peer, og derefter sende en offer-pakke med et “offer” event.

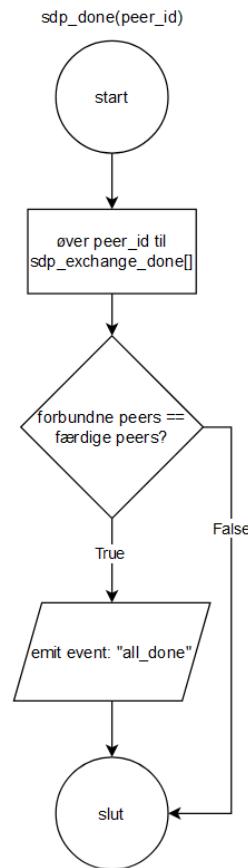
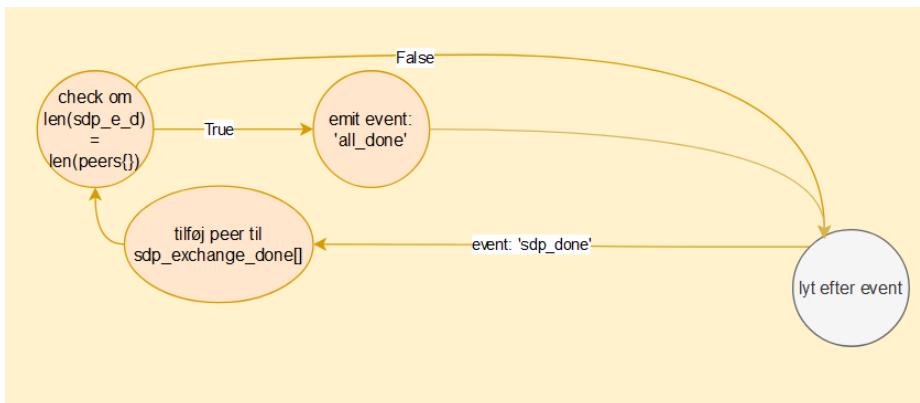
Når serveren registrerer et “offer” event vil den lede i offer-pakken efter en afsender og en modtager, og sørge for at pakken bliver distribueret korrekt. Det samme sker i tilfælde af et “answer” event.



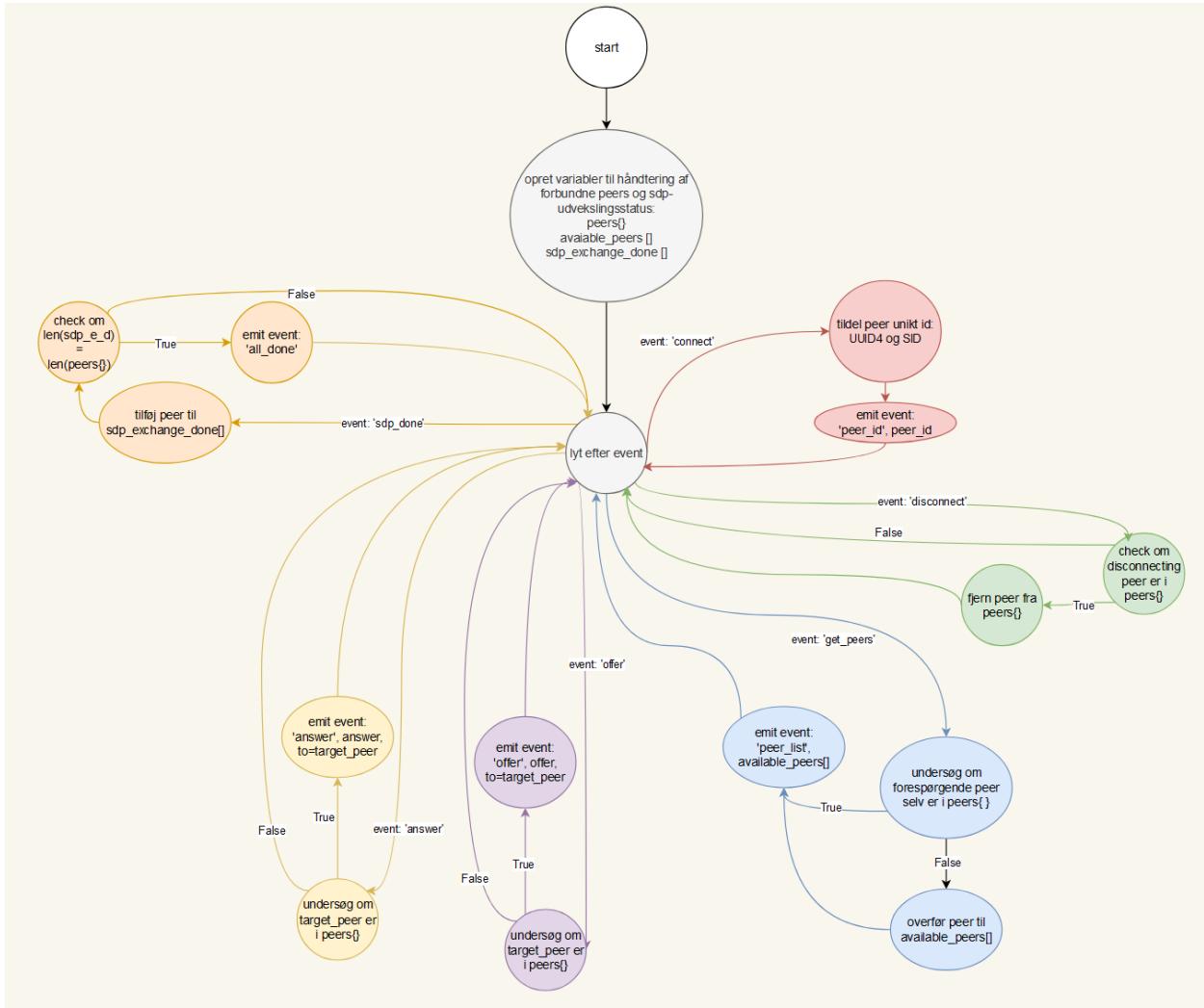
handle_offer(data)
handle_answer(data)



Når hver peer har færdiggjort deres del af SDP udvekslingsprocessen udsender de et "sdp_done" event. Ved dette event tjekker serveren om antallet af peers, der er færdige med SDP-udveksling er det samme antal, som der er forbundet til serveren. Hvis dette er tilfældet, udsender serveren et "all_done" event, som fortæller hver peer, at de sikkert kan lukke deres forbindelse til signaling-serveren.



Således ser signaling-serverens states ud som helhed:



6.5.2.2 WebRTC klient

WebRTC klienten er tænkt som en løsning, der både kan håndtere at være ”offerer” og ”answerer”, idet det ikke altid kan forudses hvilken peer, der først forbinder til signaling-serveren. Den første peer, der sender en SDP er offerer, hvorfor modtageren af denne bliver answerer.

Til peer klienten benyttes følgende moduler:

- Asyncio
- Socketio
- Aiortc, MediaStreamTrack
- Aiohttp
- Fractions
- Av
- audio_handling

Den komplette kode kan findes i bilagene

Peer-klienten benytter OOP for at gøre håndtering af peer_id's og RTC-peer forbindelser nemmere, især under udviklingsprocessen, hvor flere instanser af WebRTC peers kørte på samme maskine. Denne software er skrevet med henblik på en peer-to-peer forbindelse mellem nøjagtigt 2 peers.

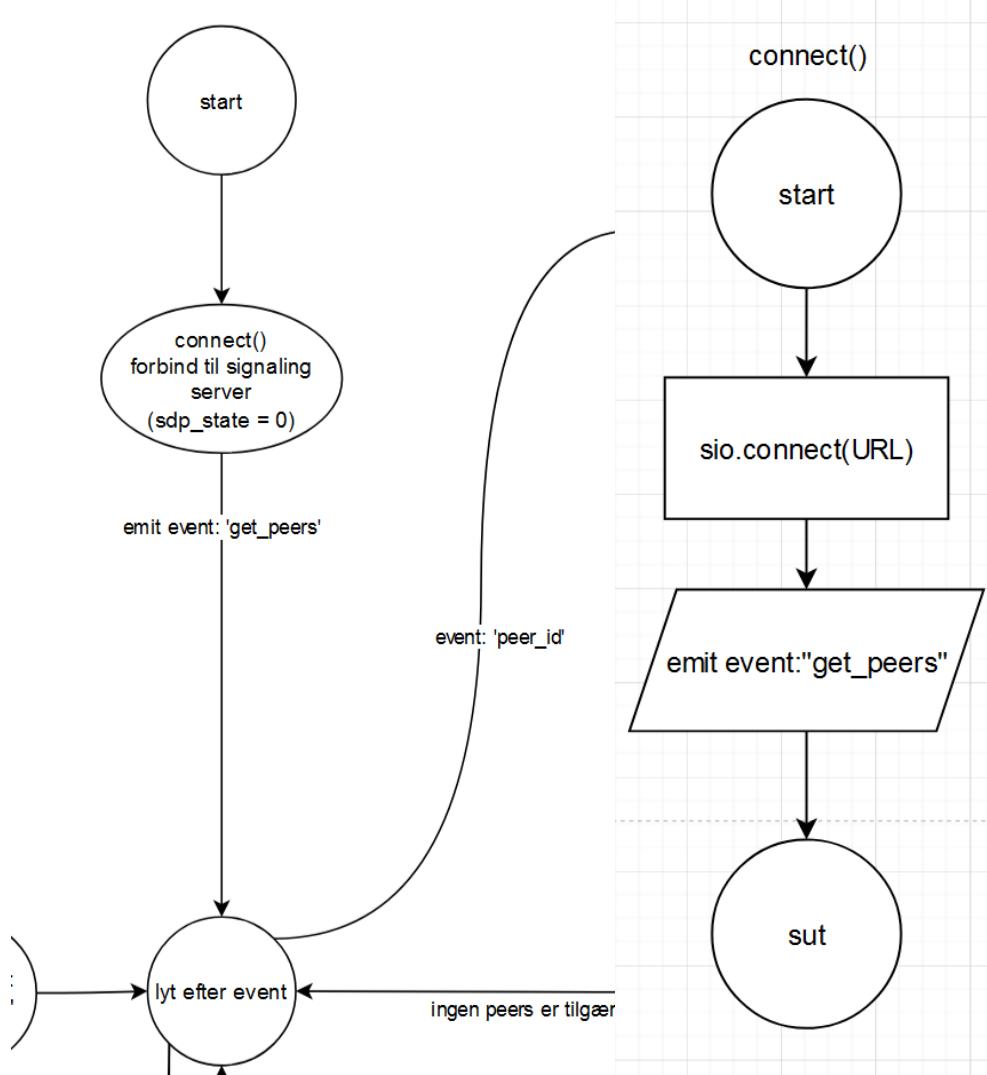
Et UML-diagram er forsøgt oprettet for at illustrere PeerClient klassen i WebRTC klient-softwaren:

<i><<PeerClient>></i> PeerClient
<pre>+ self.io = socketio.Client + self.peer_id = "" + self.target_peer_id = "" + self.peer_connection = aiortc.RTCPeerConnection() + self.io.on: ('connect', self.on_connect) ('disconnect', self.on_disconnect) ('peer_id', self.set_peer_id) ('peer_list', self.handle_peer_list) ('offer', self.handle_offer) ('answer', self.handle_answer) ('candidate', self.on_ice_candidate) + self.sdp_state = int + self.test_track = object + self.audio_output = object + self.audio_output.create_output_stream() + connect(): connect til server og emit event "get_peers" + on_connect(): str + on_disconnect(): str + set_peer_id(data): peer_id + handle_peer_list(data): target_peer_id + send_offer(): sio.emit('offer', offer{}) + send_answer(): self.io.emit('answer', answer{}) + create_offer(): localDescription.sdp(offer_sdp) + create_answer(): localDescription.sdp(answer_sdp) + handle_offer(offer_sdp): setRemoteDescription(offer_sdp) + handle_answer(answer_sdp): setRemoteDescription(answer_sdp) + add_audio: sæt self.test_track og tilføj mediespor til self.peer_connection + wait(): self.io.wait()</pre>

Herunder beskrives designet af SDP-udvekslingsprocessen mellem to peers.

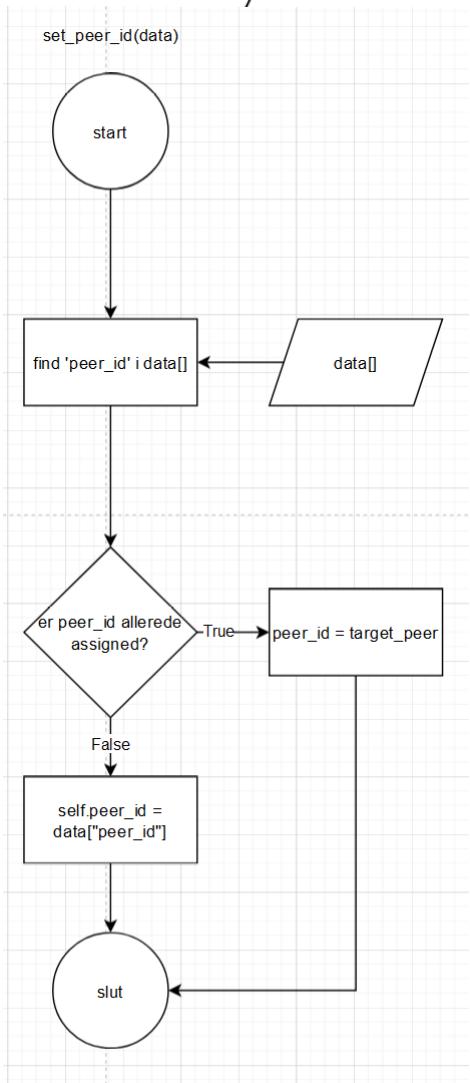
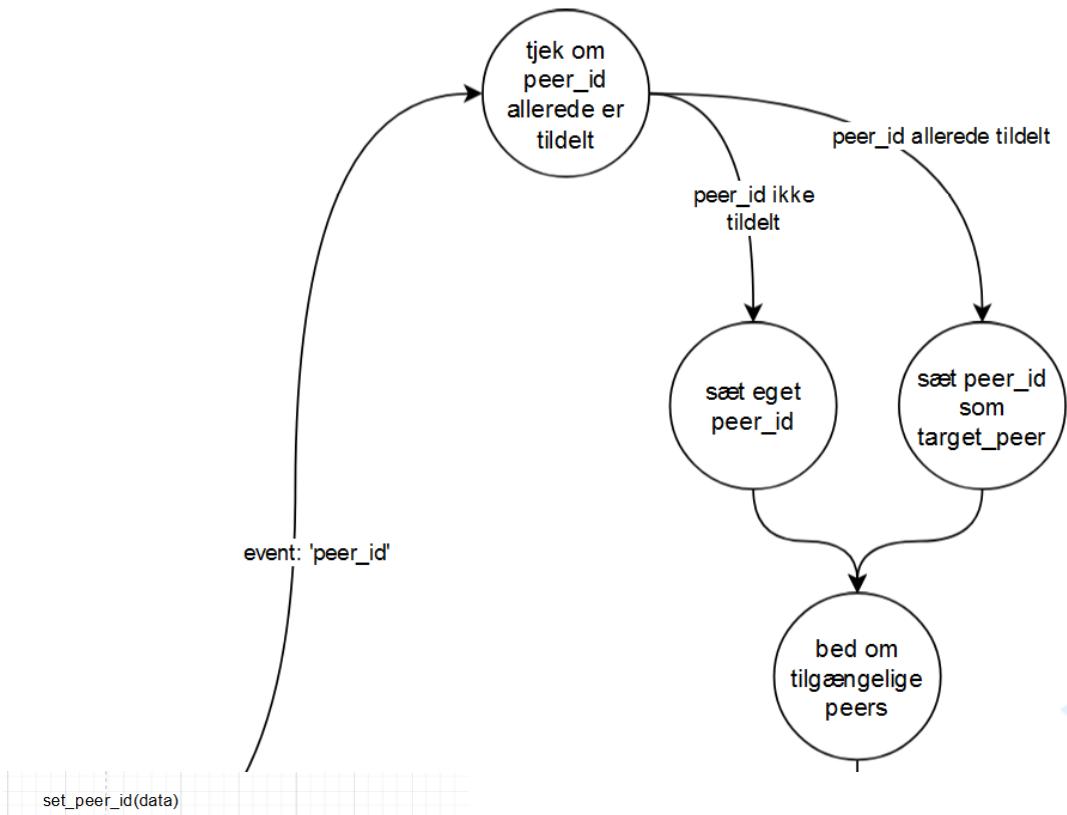
Når peer-klienten starter op, skal den forsøge at forbinde til signaling serveren, og derefter spørge serveren, om der er andre tilgængelige peers.

peer
SDP og ICE udveksling



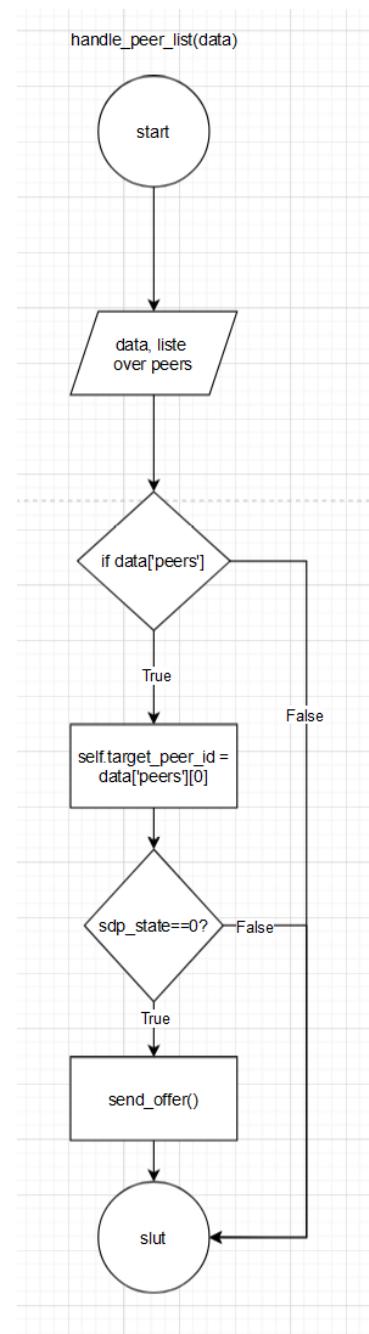
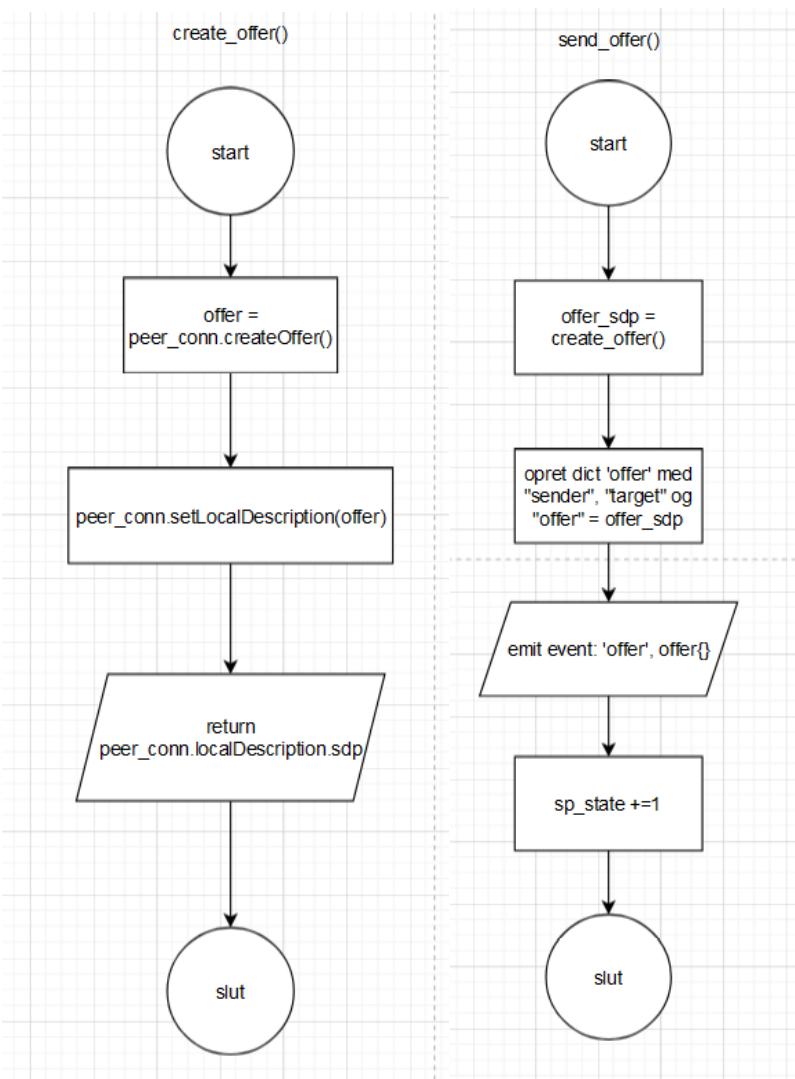
Umiddelbart efter klienten er forbundet til serveren vil den få et peer id-event med et peer_id. Herefter vil klienten undersøge om den har et ID eller ej, og handle derefter.

Har den et peer_id antager den, at det tilsendte peer_id er den anden peers' ID, og sætter dette som target_peer_id. Ellers sætter den peer_id som sit egen.

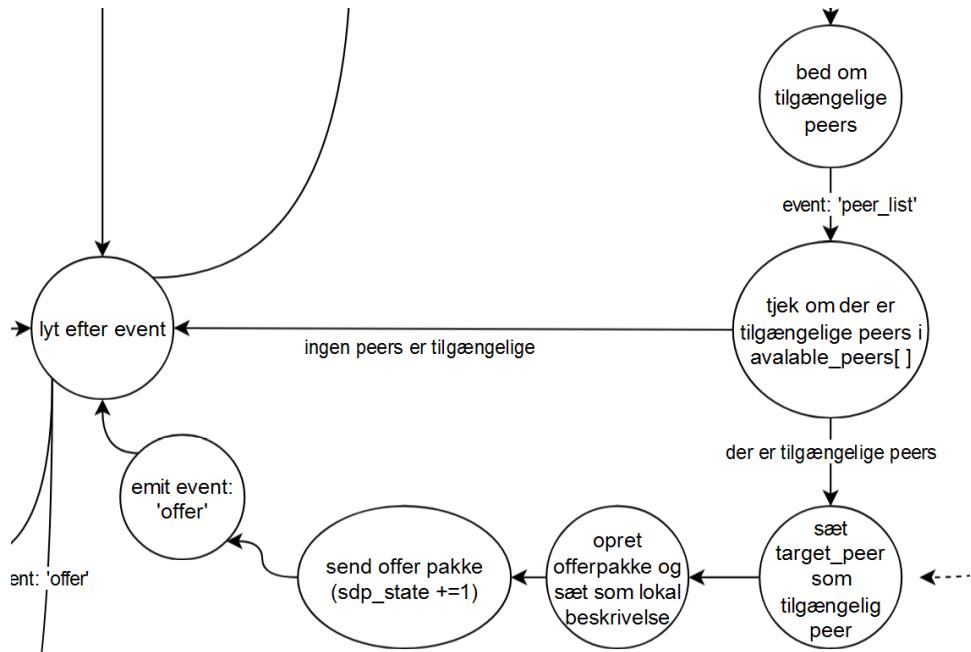


Hver gang et peer_id er sat vil den givne peer undersøge, om der er tilgængelige peers forbundet til signaling-serveren.

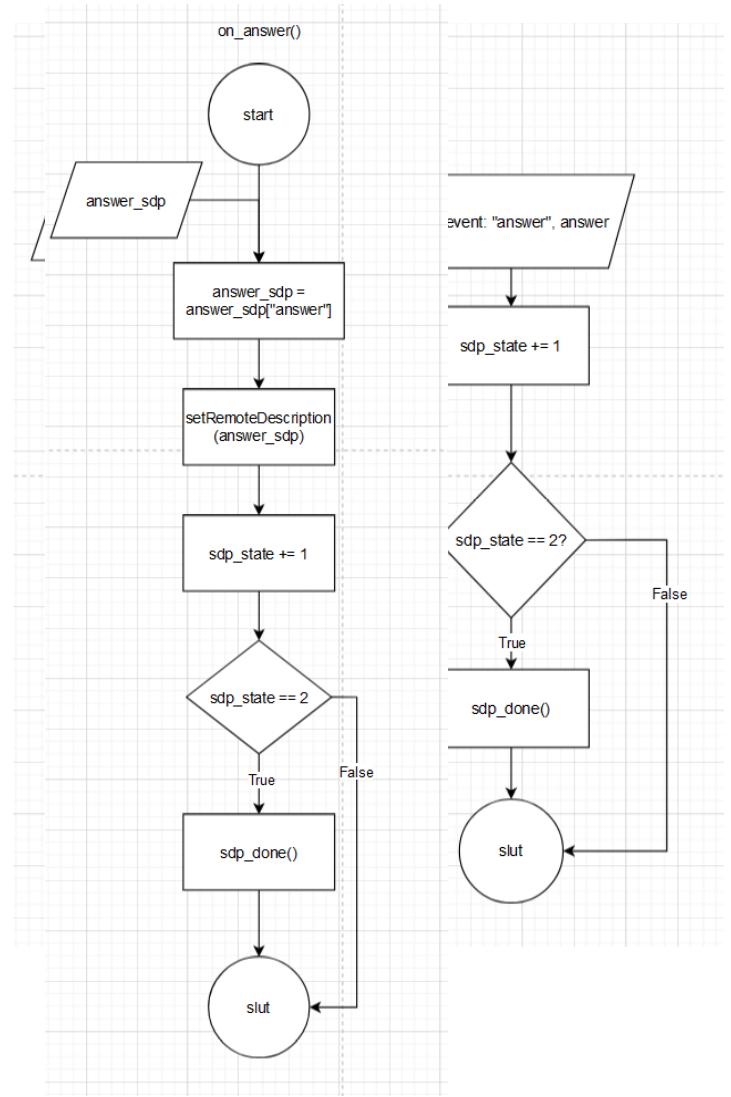
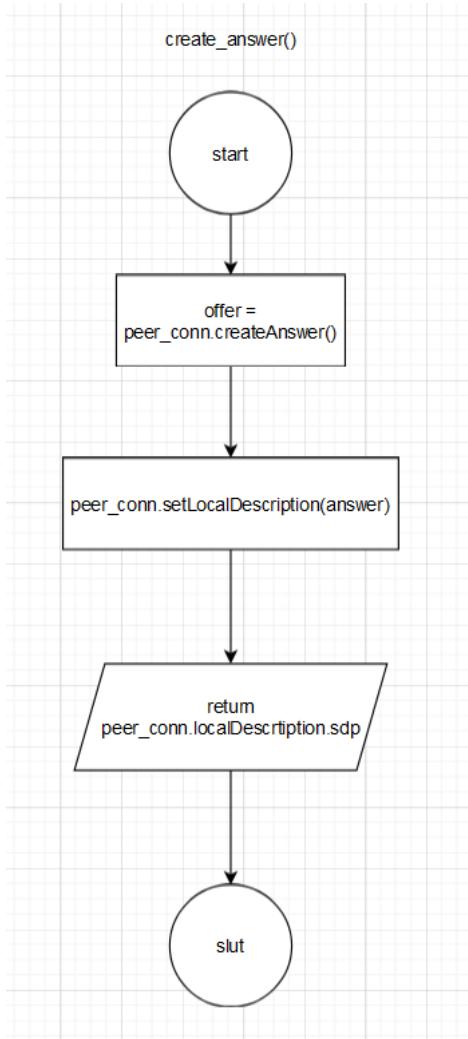
Hvis dette er tilfældet, sætter den tilgængelige peers som target_peer_id, og opretter en offer-pakke, som den sætter som sin lokale beskrivelse.



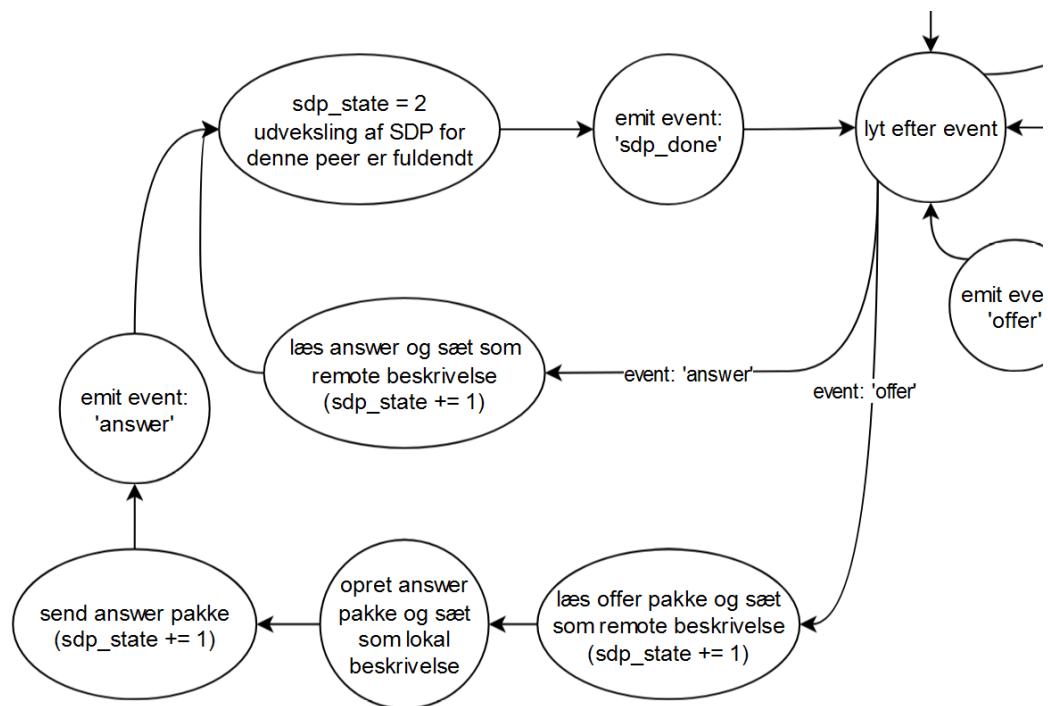
Offer-pakken sendes med et “offer” event til signaling serveren. Serveren står så for at omdele pakken korrekt. Derudover forøger den sit sdp_state med 1. sdp_state bruges til at holde styr på hvornår klienten er færdig med at udveksle SDP pakker.



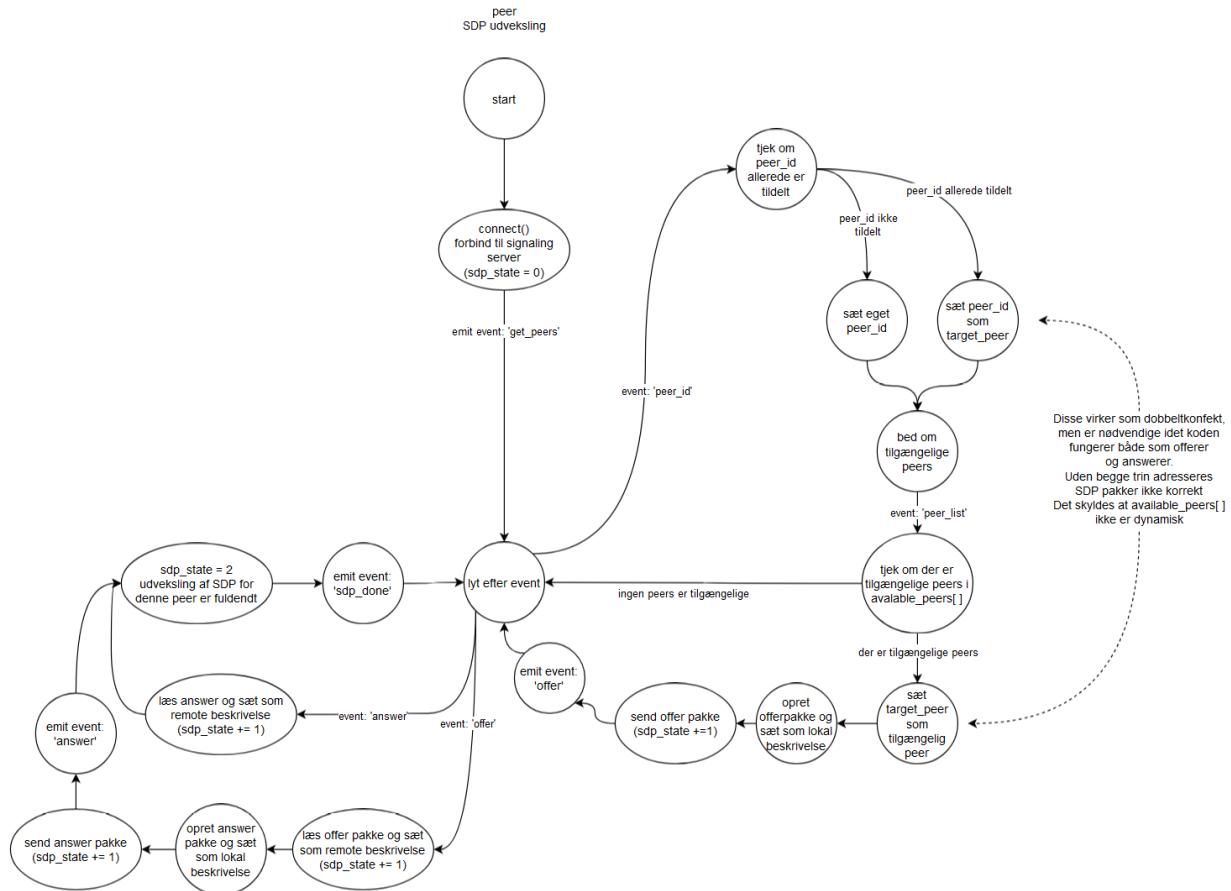
Answeren reagerer den ved at læse pakken og sætte den som "remote" beskrivelse, altså som den fjerne peers beskrivelse. Herefter forøger den sit sdp_state med 1. Den opretter så en "answer" pakke, og sætter denne som sin egen lokale beskrivelse. Derefter forøger den sit sdp_state med 1 endnu en gang. Nu er sdp_state = 2, og herved ved denne peer, at dens SDP-udveksling er færdig.



“Offerer”-peeren modtaget dette “answer”, læser den og sætter den som *sit* remote beskrivelse. Derefter forøger den sit sdp_state med 1, og sdp_state er nu 2 for begge peers. Begge peers har altså en local og remote beskrivelse, og dermed er SDP udvekslingen ovre. Når hver peer har sdp_state = 2 fortæller de serveren at de er færdige ved at emitte et “sdp_done” event.

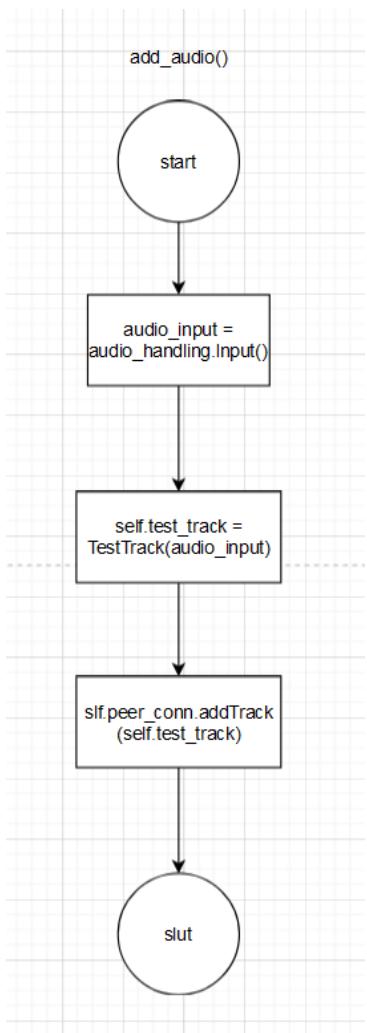


Det fulde billede af udvekslingen af SDP-pakker ser således ud:



Herfra sørger AIORTC modulet for at håndtere et DTLS handshake og oprette og vedligeholde en RTC-forbindelse mellem de to peers. RTC står for Real Time Communication.

For at oprette en RTC-forbindelse skal AIORTC dog have et mediespor. Til dette lavede undertegnede en TestTrack klasse, som kunne tage input data fra audio_handling og pakke disse ned i en AudioFrame fra 'av' modulet. Metoden add_audio() i PeerClient klassen sorgede så for at tilføje dette TestTrack til RTC-forbindelsens mediespor.



Undertegnede lavede et UML-diagram over TestTrack klassen

<<MediaStreamTrack>>

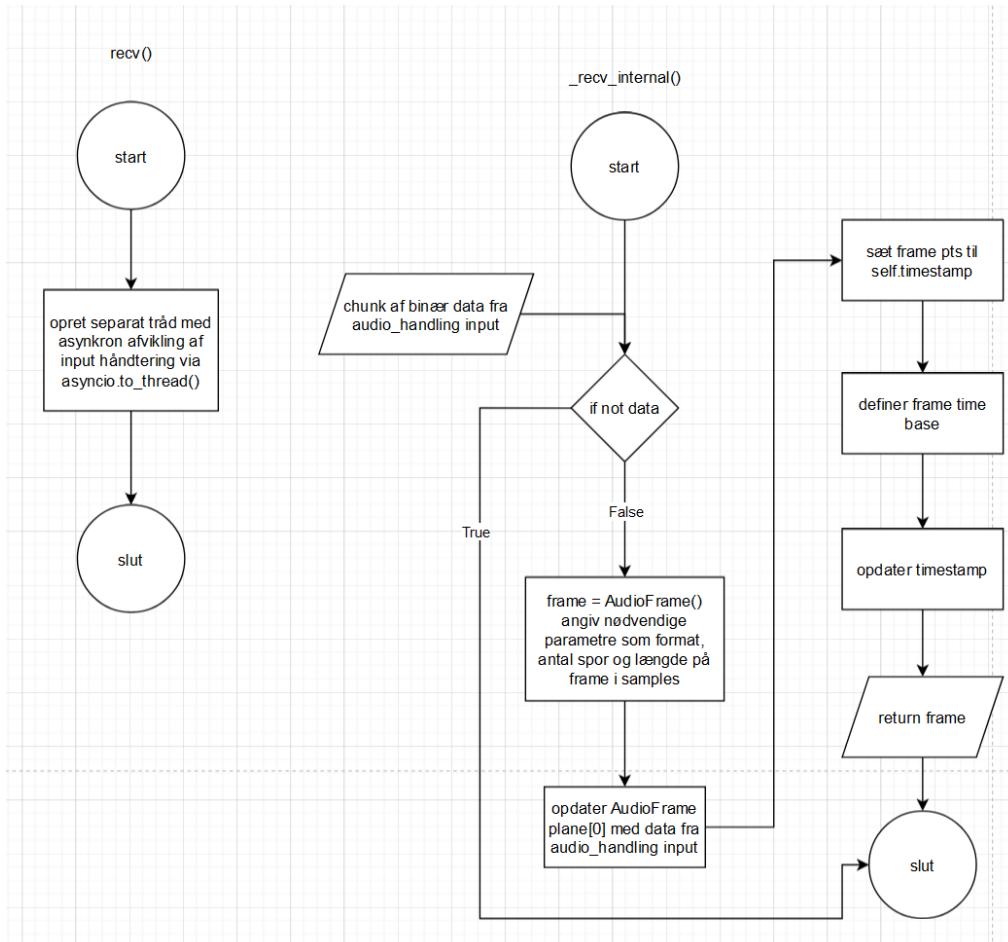
TestTrack

Denne klasse arver fra AIORTC's MediaStreamTrack klasse, så TestTracks metoder kan bruges når audio data skal læses over i et mediespor i RTC-forbindelsen.

I et forsøg på at imødekomme potentielle bottleneck-problemer på Raspberry Pi 4B benyttede undertegnede sig af asyncio's to_thread() metode for at starte recv() i en separat tråd.

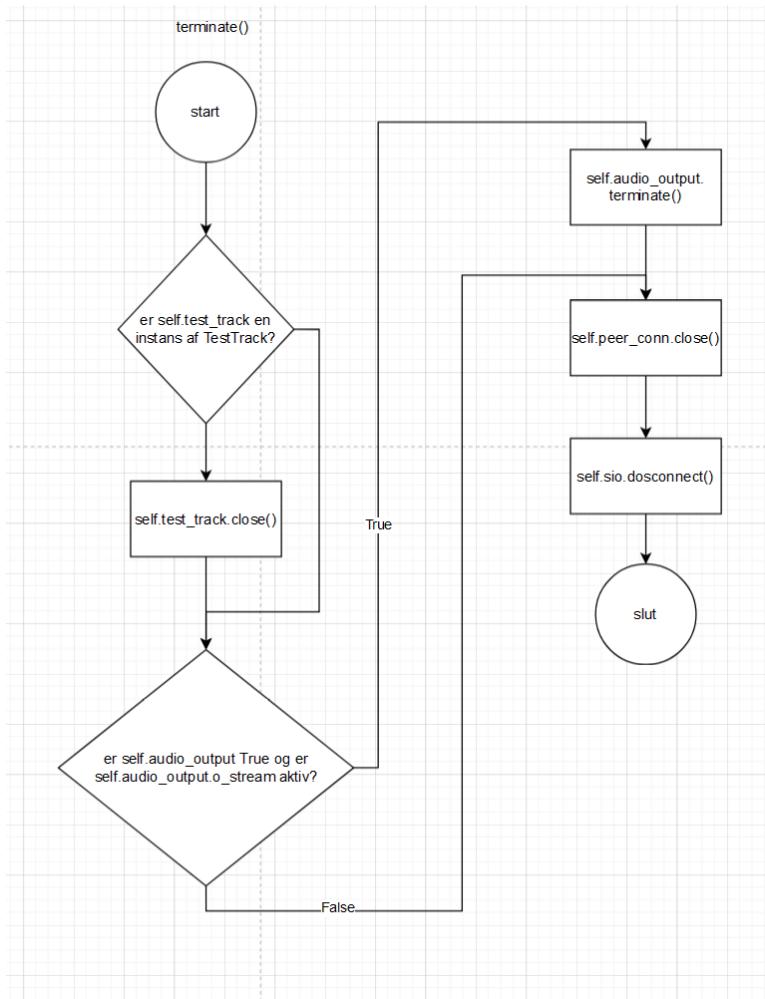
```
+ self.kind: "audio"
+ self.input_track: Input()
+ self.samples: int
+ self.timestamp: int

+ recv()
+ _recv_internal(): av.AudioFrame()
+ self.close:
```



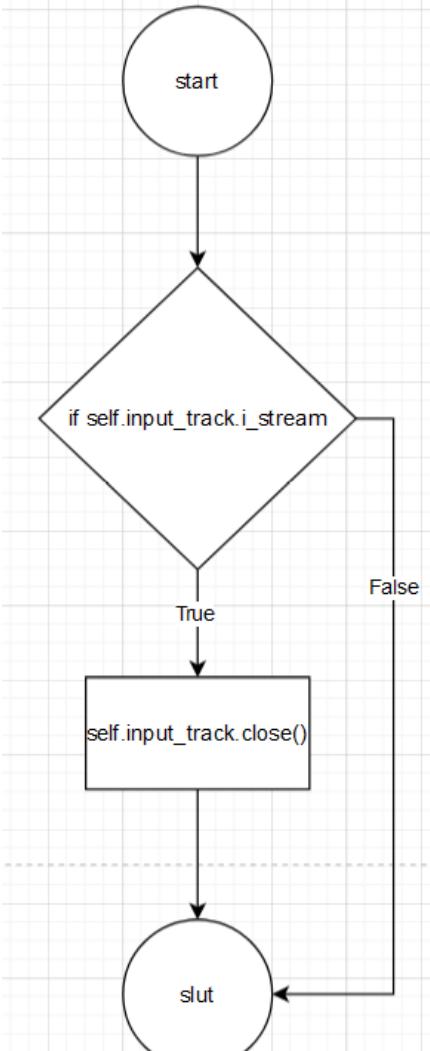
Til slut skal TestTrack og PeerClient kunne lukkes korrekt for at frigøre deres ressourcer

PeerClient.terminate()



TestTrack.close()

close()



5.6.3 Formel modultest

5.6.3.1 WebRTC klient

Koden til test af WebRTC klienten kan findes under **Bilag 11 – WebRTC klient, Kode til test af WebRTC klient.**

Herunder viser WebRTC klientens debug log, at den kan få forbindelse til signaling serveren, og modtager og afsender de forventede events. Den hvide tekst er print statements, som kommer lidt vilkårligt - dette skyldes klientens asynkrone natur.

Ikke desto mindre fremgår det tydeligt herunder, at klienten etablerer forbindelse til signaling servere, og får tildelt et unikt id, her "b641e26d-[..]", herefter "peer-b64".

Available peers listen er tom for nu, idet denne klient er den første forbundne. Den anden peer vil dog kunne se peer-b64 i samme liste.

```
DEBUG:asyncio:Using proactor: IocpProactor
Signal handler is unsupported
WARNING:engineio.client:Signal handler is unsupported
Attempting WebSocket connection to ws://192.168.0.8:5000/socket.io/?transport=websocket&EIO=4
INFO:engineio.client:Attempting WebSocket connection to ws://192.168.0.8:5000/socket.io/?transport=websocket&EIO=4
WebSocket connection accepted with {'sid': 'tt8sftHS-ZrMb2C7AAAA', 'upgrades': [], 'pingTimeout': 20000, 'pingInterval': 25000}
INFO:engineio.client:WebSocket connection accepted with {'sid': 'tt8sftHS-ZrMb2C7AAAA', 'upgrades': [], 'pingTimeout': 20000, 'pingInterval': 25000}
Engine.IO connection established
INFO:socketio.client:Engine.IO connection established
Sending packet MESSAGE data 0{}
INFO:engineio.client:Sending packet MESSAGE data 0{}
Received packet MESSAGE data 2[{"peer_id":{"peer_id":"b641e26d-a98c-4c3a-9f73-830e4b6681ba"}]
INFO:engineio.client:Received packet MESSAGE data 2[{"peer_id":{"peer_id":"b641e26d-a98c-4c3a-9f73-830e4b6681ba"}]
Received packet MESSAGE data 0{"sid":"C_5UUKPuEHk20v4nAAAB"}
INFO:engineio.client:Received packet MESSAGE data 0{"sid":"C_5UUKPuEHk20v4nAAAB"}
Received event "peer_id" []
INFO:socketio.client:Received event "peer_id" []
Namespace / is connected
INFO:socketio.client:Namespace / is connected
Emitting event "get_peers" []
INFO:socketio.client:Emitting event "get_peers" []
Sending packet MESSAGE data 2["get_peers"]
INFO:engineio.client:Sending packet MESSAGE data 2["get_peers"]
Received packet MESSAGE data 2[{"peer_list":{"peers":[]}]
INFO:engineio.client:Received packet MESSAGE data 2[{"peer_list":{"peers":[]}}
Received event "peer_list" []
INFO:socketio.client:Received event "peer_list" []
Created instance of Peer Client
Connecting...
Assigned peer id: b641e26d-a98c-4c3a-9f73-830e4b6681ba
Connected to signaling server!
Available peers: []
```

Her er et udsnit af debug-loggen, der viser den anden klients perspektiv:

```
Created instance of Peer Client
Connecting...
Assigned peer id: 14333795-a894-4183-ad4f-1bcb613ff724
Connected to signaling server!
Available peers: ['b641e26d-a98c-4c3a-9f73-830e4b6681ba']
```

Den får tildelt et unikt ID, her "14333795-[..]", fremover "peer-143".

Peer-143 kan se peer-b64 i listen "available peers".

Samtidigt med at peer-143 forbindes til signaling serveren videresendes peer-143's ID til peer-b64, og hver peer sætter hinandens respektive ID som target_id.

Så snart en peer kender sin makkers ID kan denne nu oprette en SDP-pakke. Samtidigt med at denne SDP pakke oprettes, påbegyndes en undersøgelse af tilgængelige ICE-kandidater. Disse ICE-kandidater pakkes ned i SDP pakken, og sendes til makker-peer'en så snart pakken er klar.

Dette er anderledes end fra normal WebRTC, som benytter sig af "trickling ICE". "Trickling" refererer til, at hver WebRTC klient løbende leder efter ICE kandidater, og sender dem separat fra SDP-pakken - gerne efter SDP-pakkerne er udvekslet. Her kommer ICE kandidaterne altså "dryssende" ind til hver peer.

Aiorc sender dog først SDP pakker når nødvendige ICE-kandidater er undersøgt. Man kan konfigurere dette, således at så snart nye ICE-kandidater bliver fundet vil disse blive overført. Det har jeg dog valgt fra i denne omgang, eftersom dette blot er et simpelt proof-of-concept.

Herunder ses et eksempel på en SDP-pakke, som man kan dykke lidt ned i. Jeg har omkranset 3 punkter med hhv. en gul, orange og rød streg.

Stykket omkranset af gul fortæller, at denne peer er i stand til at sende og modtage.

Det orange fortæller hvilke codec, sample-rate og antal spor den forventer af et audio-spor.

Stykket omkranset af rødt viser en række kandidater - her en masse host kandidater, som er lokale ICE-kandidater. Netop disse IP-adresser og porte svarer til min egen PC's IP-adresse på mit hjemmenetværk (192.168.0.8), samt mine installerede virtuelle maskiners netværks interfaces. Den sidste ICE-kandidat er af typen srflx, altså en server reflexive kandidat, eller en STUN-kandidat. Denne fortæller hvilken offentlige IP-adresse og port min PC's private adresse opløses til gennem NAT.

Kilde:

https://en.wikipedia.org/wiki/Session_Description_Protocol

<https://datatracker.ietf.org/doc/html/rfc4566>

<https://webrtcchacks.com/sdp-anatomy/>

```
v=0
o-- 3941793374 3941793374 IN IP4 0.0.0.0
s--
t=0 0
a=group:BUNDLE 0
a=msid-semantic:WMS *
m=audio 57958 UDP/TLS/RTP/SAVPF 96 0 8
c=IN IP4 192.168.64.1
a=sendrecv
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=mid:0
a=msid:99478416-0574-4457-a9e7-8eee9528e400 test
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=candidate:e5bcfe9c71647ac306da9adfe5d02ca8 1 udp 2130706431 192.168.64.1 57958 typ host
a=candidate:92f6804a2cec20ceca1d84e2682189b7 1 udp 2130706431 192.168.234.1 57959 typ host
a=candidate:ef0a740123e0eb7a526bfc23581ddecc 1 udp 2130706431 192.168.11.1 57960 typ host
a=candidate:5fc68ff9a354c4a0add93bad3fb1589a 1 udp 2130706431 192.168.12.1 57961 typ host
a=candidate:76e208abe79aa2884271ef2ddfe7252e 1 udp 2130706431 192.168.0.8 57962 typ host
a=candidate:9d6e235763ed7b68e2891be6e15a1636 1 udp 1694498815 93.165.245.254 46376 typ srflx raddr 192.168.0.8 rport 57962
a=end-of-candidates
a=ice-ufrag:9exN
a=ice-pwd:MbmbCrmJvzvGg4MGnoh8re
a=fingerprint:sha-256 8B:4D:67:93:62:F5:F8:F2:4C:B3:6C:C0:18:F7:71:DD:45:96:C2:00:B7:F5:FB:C9:6B:F0:C4:2F:5F:84:9B:30
a=setup:actpass
```

Til højre ses de samme ip-adresser i kommandoprompten ved en ipconfig kommando

```

Ethernet adapter VMware Network Adapter VMnet1:
  Connection-specific DNS Suffix . :
  Link-local IPv6 Address . . . . . : fe80::f0e7:892f:64bb:d1bb%24
  IPv4 Address . . . . . : 192.168.64.1
  Subnet Mask . . . . . : 255.255.255.0
  Default Gateway . . . . . :

Ethernet adapter VMware Network Adapter VMnet8:
  Connection-specific DNS Suffix . :
  Link-local IPv6 Address . . . . . : fe80::3741:cfc0:2c57:ff32%9
  IPv4 Address . . . . . : 192.168.234.1
  Subnet Mask . . . . . : 255.255.255.0
  Default Gateway . . . . . :

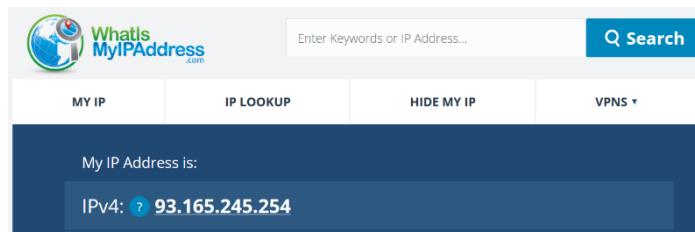
Ethernet adapter VMware Network Adapter VMnet2:
  Connection-specific DNS Suffix . :
  Link-local IPv6 Address . . . . . : fe80::ac2e:aaf6:3541:d366%8
  IPv4 Address . . . . . : 192.168.11.1
  Subnet Mask . . . . . : 255.255.255.0
  Default Gateway . . . . . :

Ethernet adapter VMware Network Adapter VMnet3:
  Connection-specific DNS Suffix . :
  Link-local IPv6 Address . . . . . : fe80::df6b:850e:b61e:bc4d%44
  IPv4 Address . . . . . : 192.168.12.1
  Subnet Mask . . . . . : 255.255.255.0
  Default Gateway . . . . . :

Wireless LAN adapter Wi-Fi:
  Connection-specific DNS Suffix . : home
  Link-local IPv6 Address . . . . . : fe80::c6b:4bef:c53b:5f6d%5
  IPv4 Address . . . . . : 192.168.0.8
  Subnet Mask . . . . . : 255.255.255.0

```

Og ganske rigtigt er undertegnede public IP ifølge whatismyipaddress.com den samme som den, der er angivet i SDP-pakken:



Herunder viser Wireshark hvordan PC peer forbinder til en STUN server i et forsøg på at få oplost denne offentlige IP-adresse. Denne test er foretaget en senere dag - 192.168.0.8 er undertegnede PC, og 192.168.0.65 er RPI peer:

179	18.790904	192.168.0.8	74.125.250.129	STUN	62 Binding Request
181	18.826433	74.125.250.129	192.168.0.8	STUN	74 Binding Success Response XOR-MAPPED-ADDRESS: 93.165.252.94:10319
184	23.774429	192.168.0.8	192.168.0.65	STUN	130 Binding Request user: upH8:7oKD
190	23.804022	192.168.0.65	192.168.0.8	STUN	106 Binding Success Response XOR-MAPPED-ADDRESS: 192.168.0.8:50256
225	23.822003	192.168.0.65	192.168.0.8	STUN	134 Binding Request user: 7oKD:upH8
226	23.822509	192.168.0.8	192.168.0.65	STUN	106 Binding Success Response XOR-MAPPED-ADDRESS: 192.168.0.65:46780

Når begge peers har udvekslet SDP pakker med hinanden begynder de hver at tjekke ICE kandidat-par igennem. Disse er par af forbindelsespunkter, som de to peers kan bruge til at skabe en direkte forbindelse med hinanden, og hver peer undersøger hver mulig kombination for at finde den bedste forbindelse til hinanden.

Her ser det kedeligt ud. Kun én success af MANGE forsøg.

```
INFO:aioice.ice:Connection(0) Check CandidatePair(('192.168.64.1', 57958) -> ('192.168.64.1', 57965)) State.IN_PROGRESS -> State.SUCCEEDED
INFO:aioice.ice:Connection(0) Check CandidatePair(('192.168.12.1', 57961) -> ('192.168.0.8', 57964)) State.WAITING -> State.FAILED
INFO:aioice.ice:Connection(0) Check CandidatePair(('192.168.0.8', 57962) -> ('192.168.0.8', 57964)) State.WAITING -> State.FAILED
INFO:aioice.ice:Connection(0) Check CandidatePair(('192.168.234.1', 57959) -> ('192.168.64.1', 57965)) State.FROZEN -> State.FAILED
INFO:aioice.ice:Connection(0) Check CandidatePair(('192.168.11.1', 57960) -> ('192.168.64.1', 57965)) State.FROZEN -> State.FAILED
INFO:aioice.ice:Connection(0) Check CandidatePair(('192.168.12.1', 57961) -> ('192.168.64.1', 57965)) State.FROZEN -> State.FAILED

INFO:aioice.ice:Connection(0) Check CandidatePair(('192.168.11.1', 57960) -> ('93.165.245.254', 46377)) State.FROZEN -> State.FAILED
INFO:aioice.ice:Connection(0) Check CandidatePair(('192.168.12.1', 57961) -> ('93.165.245.254', 46377)) State.FROZEN -> State.FAILED
INFO:aioice.ice:Connection(0) Check CandidatePair(('192.168.0.8', 57962) -> ('93.165.245.254', 46377)) State.FROZEN -> State.FAILED
INFO:aioice.ice:Connection(0) ICE completed
```

Ikke desto mindre ser processen ud til at være succesfuld, og de to peers laver et DTLS(Datagram Transport Layer Security) handshake, således at forbindelsen er sikker.

Når dette handshake er fuldendt kan RTP(Real-time Transport Protocol) forbindelsen starte, i dette eksempel med et "dummy" audio-spor.

```
DEBUG:aiortc.rtctransport:RTCIceTransport(controlling) - checking -> completed
DEBUG:aiortc.rtcpeerconnection:RTCPeerConnection() iceConnectionState checking -> completed
DEBUG:aiortc.rtcdtlstransport:RTCDtlsTransport(server) - State.NEW -> State.CONNECTING
DEBUG:aiortc.rtcdtlstransport:RTCDtlsTransport(server) x DTLS handshake negotiated SRTP_AEAD_AES_256_GCM
DEBUG:aiortc.rtcdtlstransport:RTCDtlsTransport(server) - DTLS handshake complete
DEBUG:aiortc.rtcdtlstransport:RTCDtlsTransport(server) - State.CONNECTING -> State.CONNECTED
DEBUG:aiortc.rtcpeerconnection:RTCPeerConnection() connectionState connecting -> connected
DEBUG:aiortc.rtcrtpsender:RTCRtpSender(audio) - RTP started
```

For nu opstår en fejl, idet presentation-timestampet I medie-sporet I RTC forbindelsen faktisk er None. Derfor kan klienterne ikke behandle sporet, og RTP forbindelsen lukkes.

```
WARNING:aiortc.rtcrtpsender:RTCRtpSender(audio) Traceback (most recent call last):
  File "C:\Users\sf97\PycharmProjects\Eksamensprojekt 2. sem\Audio stream 1\.venv\Lib\site-packages\aiortc\rtcrtpsender.py", line 346, in _run_rtp
    enc_frame = await self._next_encoded_frame(codec)
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\sf97\PycharmProjects\Eksamensprojekt 2. sem\Audio stream 1\.venv\Lib\site-packages\aiortc\rtcrtpsender.py", line 298, in _next_encoded_frame
    payloads, timestamp = self._encoder.pack(data)
                ^^^^^^^^^^
  File "C:\Users\sf97\PycharmProjects\Eksamensprojekt 2. sem\Audio stream 1\.venv\Lib\site-packages\aiortc\codecs\opus.py", line 100, in pack
    timestamp = convert_timebase(packet.pts, packet.time_base, TIME_BASE)
                ^^^^^^^^
AttributeError: 'NoneType' object has no attribute 'pts'

DEBUG:aiortc.rtcrtpsender:RTCRtpSender(audio) - RTP finished
```

Det gør dog ikke noget for nu. Forbindelsen oprettes igen, der transmitemes bare ikke mediedata over mediesporet.

```

DEBUG:aiortc.rtcrtpsender:RTCRtpSender(audio) - RTP finished
DEBUG:aiortc.rtcrtpsender:RTCRtpSender(audio) - RTCP started
DEBUG:aiortc.rtcrtpsender:RTCRtpSender(audio) > RTcpSrPacket(ssrc=2556481507, sender_info=RtcpSenderInfo(ntp_timestamp=0, rtp_timestamp=0, packet_count=0, octet_count=0), reports=[{}])
DEBUG:aiortc.rtcrtpsender:RTCRtpSender(audio) > RtcpSdesPacket(chunks=[RtcpSourceInfo(ssrc=2556481507, items=[(1, b'c481d758-3701-4934-9f0d-eed350f7627d')])])
DEBUG:aiortc.rtcrtpreceiver:RTCRtpReceiver(audio) < RtcpSrPacket(ssrc=3866482343, sender_info=RtcpSenderInfo(ntp_timestamp=0, rtp_timestamp=0, packet_count=0, octet_count=0), reports=[{}])
DEBUG:aiortc.rtcrtpreceiver:RTCRtpReceiver(audio) < RtcpSrPacket(ssrc=3866482343, sender_info=RtcpSenderInfo(ntp_timestamp=0, rtp_timestamp=0, packet_count=0, octet_count=0), reports=[{}])
DEBUG:aiortc.rtcrtpreceiver:RTCRtpReceiver(audio) < RtcpSrPacket(ssrc=2556481507, sender_info=RtcpSenderInfo(ntp_timestamp=0, rtp_timestamp=0, packet_count=0, octet_count=0), reports=[{}])
DEBUG:aiortc.rtcrtpreceiver:RTCRtpReceiver(audio) < RtcpSrPacket(ssrc=2556481507, items=[(1, b'c481d758-3701-4934-9f0d-eed350f7627d')]))
DEBUG:aiortc.rtcrtpreceiver:RTCRtpReceiver(audio) < RtcpSdesPacket(chunks=[RtcpSourceInfo(ssrc=2556481507, items=[(1, b'c481d758-3701-4934-9f0d-eed350f7627d')])])
DEBUG:aiortc.rtcrtpreceiver:RTCRtpReceiver(audio) < RtcpSdesPacket(ssrc=2556481507, items=[(1, b'c481d758-3701-4934-9f0d-eed350f7627d')]))
DEBUG:aiortc.rtcrtpreceiver:RTCRtpReceiver(audio) < RtcpSdesPacket(chunks=[RtcpSourceInfo(ssrc=2556481507, items=[(1, b'c481d758-3701-4934-9f0d-eed350f7627d')])])
DEBUG:aiortc.rtcrtpreceiver:RTCRtpReceiver(audio) < RtcpSdesPacket(chunks=[RtcpSourceInfo(ssrc=2556481507, items=[(1, b'c481d758-3701-4934-9f0d-eed350f7627d')])])
DEBUG:aioice.ice:Connection(0) protocol(0) < ('192.168.64.1', 57965) Message(message_method=Method.BINDING, message_class=Class.REQUEST, transaction_id=b'\xe9\xfb\xed\xc2\xc7\x80\xb2\x85\xe7\x8a')
DEBUG:aioice.ice:Connection(0) protocol(0) > ('192.168.64.1', 57965) Message(message_method=Method.BINDING, message_class=Class.RESPONSE, transaction_id=b'\xe9\xfb\xed\xt\xc2\xc7\x80\xb2\x85\xe7\x8a')
DEBUG:aiortc.rtcrtpsender:RTCRtpSender(audio) > RtcpSrPacket(ssrc=2556481507, sender_info=RtcpSenderInfo(ntp_timestamp=0, rtp_timestamp=0, packet_count=0, octet_count=0), reports=[{}])
DEBUG:aiortc.rtcrtpreceiver:RTCRtpSender(audio) < RtcpSdesPacket(chunks=[RtcpSourceInfo(ssrc=2556481507, items=[(1, b'c481d758-3701-4934-9f0d-eed350f7627d')])])
DEBUG:aiortc.rtcrtpreceiver:RTCRtpReceiver(audio) < RtcpSrPacket(ssrc=3866482343, sender_info=RtcpSenderInfo(ntp_timestamp=0, rtp_timestamp=0, packet_count=0, octet_count=0), reports=[{}])
DEBUG:aioice.ice:Connection(0) protocol(0) > ('192.168.64.1', 57965) Message(message_method=Method.BINDING, message_class=Class.REQUEST, transaction_id=b'\x1c\xca\x9k\xae\xf2\x13\x07t')
DEBUG:aioice.ice:Connection(0) protocol(0) < ('192.168.64.1', 57965) Message(message_method=Method.BINDING, message_class=Class.RESPONSE, transaction_id=b'\x1c\xca\x9k\xae\xf2\x13\x07t')

```

Ovenstående er illustreret med to instanser af samme python program, der kører på samme computer. For at skabe et mere virkelighedsnært billede af forbindelsen har undertegnede kørt samme test med en virtuel masking, der kører Ubuntu som OS.

Herunder vises et udsnit af debug loggen fra en forbindelse mellem en peer på untertegnedes PC med adressen 192.168.0.8, og en peer på en virtuel maskine med adressen 169.168.0.66. Signaling serveren kører på adressen 192.168.0.8:5000. Disse udsnit vidner om en to-vejs forbindelse mellem de to peers.

PC peer:

```

DEBUG:aioice.ice:Connection(0) protocol(4) > ('192.168.0.66', 36287) Message(message_method=Method.BINDING, message_class=Class.REQUEST, transaction_id=b'\xe9\xfb\xed\xc2\xc7\x80\xb2\x85\xe7\x8a')
DEBUG:aioice.ice:Connection(0) protocol(4) < ('192.168.0.66', 36287) Message(message_method=Method.BINDING, message_class=Class.RESPONSE, transaction_id=b'\xe9\xfb\xed\xt\xc2\xc7\x80\xb2\x85\xe7\x8a')

```

VM peer:

```

DEBUG:aioice.ice:Connection(0) protocol(0) < ('192.168.0.8', 59343) Message(message_method=Method.BINDING, message_class=Class.REQUEST, transaction_id=b'\xe9\xfb\xed\xc2\xc7\x80\xb2\x85\xe7\x8a')
DEBUG:aioice.ice:Connection(0) protocol(0) > ('192.168.0.8', 59343) Message(message_method=Method.BINDING, message_class=Class.RESPONSE, transaction_id=b'\xe9\xfb\xed\xt\xc2\xc7\x80\xb2\x85\xe7\x8a')

```

Undertegnede forsøger at benytte audio_handling til at læse audio data live og transmittere dette gennem P2P-forbindelsen.

```

async def add_audio(self):
    audio_input = audio.Input()
    self.test_track = TestTrack(audio_input)
    self.peer_connection.addTrack(self.test_track)

```

Audio_handling.py importeres som 'audio', og en instans af Input klassen initieres som audio_input. Derefter sættes test-sporet som en instans af en klasse, TestTrack, hvor audio_input passes til, således at TestTrack kan gøre brug af audio_handling. Til sidst tilføjes TestTrack til et mediespor i WebRTC forbindelsen med 'addTrack()'.

```

class TestTrack(MediaStreamTrack):
    def __init__(self, audio_input):
        super().__init__()
        self.kind = "audio"
        self.input_track = audio_input
        self.sample_rate = 48000
        self.samples = 512

    @asyncio.coroutine
    def recv(self):
        self.input_track.create_input_stream()
        try:
            data = self.input_track.record_live()
            if not data:
                print("No data available, skipping frame")
                return
            frame = AudioFrame(format="s16", layout="mono", samples=self.samples)
            print(f"Read {len(data)} bytes of audio data")

            frame.sample_rate = self.sample_rate
            # Audio data organiseres i planes - for mono audio er der kun én plane
            # [0] indikerer, at den første (og eneste) plane i 'frame' tilgås.
            # 'update(data)' lægger dataene ind i plane
            frame.planes[0].update(data)
            return frame
        except IOError as e:
            print("recv error: ", e)
            return

```

Idet TestTrack tilføjes til et WebRTC mediespor køres 'recv()' funktionen, som bruger instansen af audio_handlings 'record_live()' metode i Input klassen til at læse et input.

Fra audio_handling:

```

def record_live(self):
    try:
        data = self.i_stream.read(self.chunk, exception_on_overflow=False)
        return data
    except IOError as e:
        print("Error: ", e)

```

Herefter forsøger undertegnede at læse om de sendte data bliver modtaget, ved at lytte til et aiorpc event på peer_connection instansen:

```

self.peer_connection.on(event: "track", self.on_track)

```

Herefter printer remote peer en simpel debug-besked:

```

@asyncio.coroutine
def on_track(self, track):
    print(f"Received track: ", track)

```

Det fremgår at denne debug log, at denne peer kan modtage et spor fra en remote peer, som linjen understreget med gult beskriver. Derudover kan man se, at denne peer også læser 1024 bytes af audio af to omgange, hvilket er forventet eftersom recv() i TestTrack beder om en chunk data som jeg har sat til at være 512 samples. I et 16-bit format indeholder et sample 2 bytes, og 512^2 er 1024 bytes.

```

SDP exchange completed
Received track: <aiortc.rtcrtpreceiver.RemoteStreamTrack object at 0x00000197BA0B9190>
Disconnected from signaling server!
DEBUG:aiortc.rtcicetransport:RTCIceTransport(controlling) - checking -> completed
DEBUG:aiortc.rtcpeerconnection:RTCPeerConnection() iceConnectionState checking -> completed
DEBUG:aiortc.rtcdtlstransport:RTCDtlsTransport(server) - State.NEW -> State.CONNECTING
DEBUG:aiortc.rtcdtlstransport:RTCDtlsTransport(server) x DTLS handshake negotiated SRTP_AEAD_AES_256_GCM
DEBUG:aiortc.rtcdtlstransport:RTCDtlsTransport(server) - DTLS handshake complete
DEBUG:aiortc.rtcdtlstransport:RTCDtlsTransport(server) - State.CONNECTING -> State.CONNECTED
DEBUG:aiortc.rtcpeerconnection:RTCPeerConnection() connectionState connecting -> connected
DEBUG:aiortc.rtcrtpsender:RTCRtpSender(audio) - RTP started
DEBUG:aiortc.rtcrtpsender:RTCRtpSender(audio) - RTP started
DEBUG:aiortc.rtcrtpreceiver:RTCRtpReceiver(audio) - RTCP started
Read 1024 bytes of audio data
WARNING:aiortc.rtcrtpsender:RTCRtpSender(audio) Traceback (most recent call last):
  File "C:\Users\sf97\PycharmProjects\Eksamensprojekt 2. sem\Audio stream 1\.venv\lib\site-packages\aiortc\rtcrtpsender.py", line 350, in _run_rtp
    timestamp = uint32_add(timestamp_origin, enc_frame.timestamp)
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\sf97\PycharmProjects\Eksamensprojekt 2. sem\Audio stream 1\.venv\lib\site-packages\aiortc\utils.py", line 39, in uint32_add
    return (a + b) & 0xFFFFFFFF
               ^^^^
TypeError: unsupported operand type(s) for +: 'int' and 'NoneType'

DEBUG:aiortc.rtcrtpsender:RTCRtpSender(audio) - RTP finished
Read 1024 bytes of audio data

```

I ovenstående debug log ses der et nyt problem med hvordan AudioFrame behandles før den sendes. Den har problemer med at behandle timestamps. RTPsender forsøger at beregne et timestamp for den udgående AudioFrame, men dette kan ikke lade sig gøre idet den ikke kan addere integers og NoneTypes. Det lader til, at dette timestamp ikke oprettes korrekt automatisk, og derfor skal programmet sørge for dette.

Efter en lang fejfindingsproces med ChatGPT som sparringspartner har undertegnede fundet frem til følgende:

For at kunne afspille lyd gennem WebRTC er det vigtigt at have et timestamp på hver frame der sendes og modtages, uanset om der er lyd eller video.

Dette er for at sikre synkronisering og at hver frame afspilles i den korrekte rækkefølge.

Disse tre linjer sørger for dette:

```

frame.pts = self.timestamp
frame.time_base = Fraction(1, self.sample_rate) # Timebase er 1/sample_rate (1/48000)
self.timestamp += self.samples # Forøg timestamp med antallet af samples i en frame

```

Frame.pts er et 'presentation timestamp', altså hvornår denne frame skal afspilles. I denne kode holdes der styr på pts gennem en timestamp variabel, som forøges med det antal samples, der er i en frame, for hver frame der afsendes. F.eks. 1. frame indeholder en chunk på 512 samples - timestamp starter på 0, og forøges med 512 når framen er sendt afsted, og frame.pts ligeså. Når næste frame optages, forøges timestamp med endnu 512, og frame.pts igen ligeså.

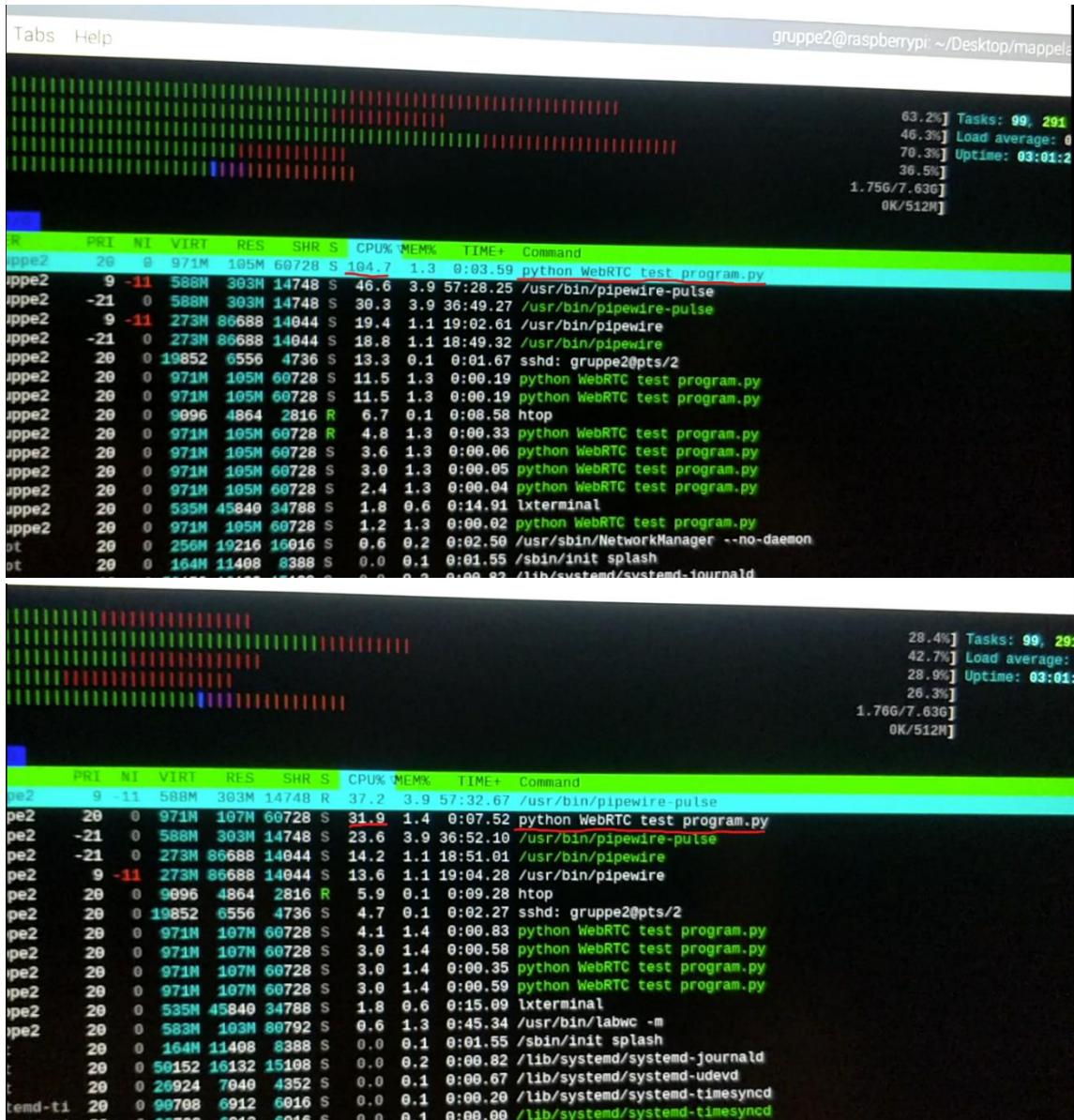
frame.time_base er en repræsentation af, hvor lang tid en frame tager i sekunder. Det er nødvendigt at lave en fraction her, idet aiortc forventer et rationelt tal til at udregne timestamps. Fordi vi har en samplerate på 48000hz skal aiortc vide, at en frame er en 48000.del af et sekund, derfor Fraction(1, 48000), som svarer til 1/48000.

Efter dette timestamp er korrekt sat kan data nu tilsyneladende overføres.

Dog kommer pakkerne med data nu fra den ene peer, Raspberry Pi'en, i en størrelse på 3 bytes, hvor pakkerne fra den anden peer, min PC, kommer med en størrelse på omkring 250 bytes. Det er noget af en forskel.

```
DEBUG:aiortc.rtcrtpsender:RTCRtpSender(audio) > RtpPacket(seq=20927, ts=1889885522, marker=1, payload=96, 248 bytes)
DEBUG:aiortc.rtcrtpreceiver:RTCRtpReceiver(audio) < RtpPacket(seq=22320, ts=713159109, marker=1, payload=96, 3 bytes)
```

Dette synes at ske, når aiortc "baglandet" får de forventede antal bytes læst. Undertegnedes intuition fortæller, at det kan have med RPI'ens processor kraft at gøre. Derfor benyttes htop til at holde et øje på CPU processen mens programmet køres, startet gennem putty.



Her ses det, at CPU'en peaker kort med virkelig højt forbrug, men lægger sig på lidt over 30% - umiddelbart virker dette ikke helt galt.

Hvis man printer information om den oprettede audioframe i recv() ser man et problem - at sample raten i den oprettede audioframe er 0 hz. Koden, der benyttes til dette kan findes under **Bilag 11 - WebRTC klient, recv() 0 hz kode**

```

Captured audio frame size: 1920 bytes
Audio frame created: <av.AudioFrame pts=None, 960 samples at 0Hz, mono, s16 at 0x21747386c20 with 960 samples
Sample rate: 48000

```

Altså kan det godt være, at der bliver målt 1920 bytes / 960 samples ved 16bit, som passer udemærket til den codec, Opus, der jf. SDP'en benyttes

```
a=rtpcp-mux  
a=ssrc:1089451301 cname:63b775d7-1a98-46bb-bc67-5e804f9bdc41  
a=rtpmap:96 opus/48000/2  
a=rtpmap:0 PCMU/8000  
a=rtpmap:8 PCMA/8000
```

Men de 1920 bytes / 960 samples er så godt som tomme:

(udsnit af en frame)

En senere måling - nu er der pludseligt data i hver frame, men sample raten er stadig 0 hz ifølge audio framen.

```
b'\x8c\x00\x92\x00\x98\x00\x9c\x00\x9a\x00\x94\x00\x9a\x00\x97\x00\x9a\x00\x97\x00\x9a\x00\x94\x00\x9f\x00\x96\x00\x8d\x00\x81\x00u'
Captured audio frame size: 1920 bytes
Audio frame created: <av.AudioFrame pts=None, 960 samples at 0Hz, mono, s16 at 0x257a3e986a0 with 960 samples
Sample rate: 48000
```

Dette skyldtes at AudioFrame objektet blev oprettet i `recv()` metoden. Efter at have justeret TestTrack koden bliver AudioAframe objektet oprettet når TestTrack initieres, og mine debug print statements fortæller mig følgende:

```
Captured audio frame size: 1920 bytes  
Audio frame created: <av.AudioFrame pts=116160, 960 samples at 48000Hz, mono, s16 at 0x22c010471c0 with 960 samples  
Sample rate: 48000 hz
```

Nu har AudioFrame en pts og en samplerate, men forskellen på pakker fra RPI-peer og PC-peer er stadiq ekstremt stor.

Jeg undersøger pakkerne gennem Wireshark for at sikre, at det ikke er et problem med repræsentationen af pakkerne gennem debug loggen:

1911	24.940658	192.168.0.65	192.168.0.8	UDP	81 46780 → 50256 Len=39
1918	24.940658	192.168.0.65	192.168.0.8	UDP	81 46780 → 50256 Len=39
1921	24.940658	192.168.0.65	192.168.0.8	UDP	81 46780 → 50256 Len=39
1933	24.942392	192.168.0.65	192.168.0.8	UDP	81 46780 → 50256 Len=39
1939	24.943371	192.168.0.8	192.168.0.65	UDP	326 50256 → 46780 Len=284
1941	24.943876	192.168.0.65	192.168.0.8	UDP	81 46780 → 50256 Len=39
1946	24.947718	192.168.0.65	192.168.0.8	UDP	81 46780 → 50256 Len=39
1950	24.948670	192.168.0.65	192.168.0.8	UDP	81 46780 → 50256 Len=39
1962	24.953316	192.168.0.65	192.168.0.8	UDP	81 46780 → 50256 Len=39
1966	24.956908	192.168.0.65	192.168.0.8	UDP	81 46780 → 50256 Len=39
1976	24.974039	192.168.0.8	192.168.0.65	UDP	327 50256 → 46780 Len=285
1977	24.983656	192.168.0.8	192.168.0.65	UDP	326 50256 → 46780 Len=284
1979	25.013064	192.168.0.8	192.168.0.65	UDP	327 50256 → 46780 Len=285
1980	25.023464	192.168.0.8	192.168.0.65	UDP	326 50256 → 46780 Len=284
1981	25.053634	192.168.0.8	192.168.0.65	UDP	327 50256 → 46780 Len=285
1982	25.063986	192.168.0.8	192.168.0.65	UDP	326 50256 → 46780 Len=284
1983	25.092961	192.168.0.8	192.168.0.65	UDP	327 50256 → 46780 Len=285
1984	25.103656	192.168.0.8	192.168.0.65	UDP	326 50256 → 46780 Len=284

192.168.0.8 = PC-peer

192.168.0.65 = RPI-peer

Heraf fremgår det, at de UDP-pakker, som WebRTC, eller i dette tilfælde aiortc, sender frem og tilbage, virkelig er forskellige størrelser. Fra PC-peer til RPI sendes pakker på omkring 284 bytes, og fra RPI-peer til PC-peer sendes pakker på 39 bytes. Forskellen på pakkerne er 245 bytes, hvilket er konsistent med den forskel jeg ser i min debug log. I det mindste kan jeg endegyldigt bekræfte, at pakkerne rent faktisk sendes og modtages over det lokale netværk.

Noget kan tyde på, at RPI'en ikke er i stand til at processere dataene i den nødvendige hastighed, eftersom to PC-peers på samme PC godt kan sende pakker af den rigtige størrelse mellem hinanden. Undertegnede og SHK forsøgte at teste dette ved at overføre undertegnedes PyCharm projekt til SHK's PC, og så køre en peer fra SHK's PC.

Men eftersom interne referencer til moduler mv. er indbagt i et projekt (f.eks. Modul xyz findes på lokation a la /SØF/Pycharm/Projects/[projekt]/.venv/lib) kan dette ikke lade sig gøre uden at oprette et helt nyt projekt og installere samtlige nødvendige moduler der.

5.6.4 Refleksioner

Da denne udviklingsproces har stået på i snart to uger, *langt* over det initiale estimat, og der nu kun er få dage til aflevering af projektet indstilles al udvikling af samtale-softwaren. På nuværende tidspunkt kan samtale-softwaren følgende:

- Signaling server kan uddele unikke ID til forbundne peers til senere SDP pakke adressering
- Signaling server kan modtage og videresende SDP pakker mellem forbundne peers
- Signaling server kan lukke sig selv ned efter fuldendt SDP-udveksling for at spare ressourcer
- Peers kan oprette forbindelse til signaling server
- Peers kan udveksle SDP pakker
- Peers kan udveksle ICE kandidater
- Peers kan gemme respektive SDP pakker for den respektive afsender
- Peers kan forhandle DTLS og gennemføre handshake
- Peers kan oprette og vedligeholde en direkte forbindelse mellem hinanden
- Peers kan oprette mediespor og lægge dette på forbindelsen
- Peers kan genkende type af mediespor modtaget fra remote peers
- Peers kan læse input data fra audio_handling
- Peers kan oprette AudioFrames til udveksling af audio data
- Peers kan læse audio data til AudioFrames
- Peers kan sende AudioFrame data mellem hinanden via RTC mediesporet gennem RTP/UDP

Den manglende funktionalitet, som undertegnede på nuværende tidspunkt kender til er følgende:

- Konsekvent lægge korrekt input data I de imellem hinanden udvekslede UDP pakker jf. Opus-codec'ets forventninger
 - Dette *kan* opnås på en PC, men ikke på en Raspberry Pi 4B
- Læse data fra UDP-pakker og videresende til en audio_handling output stream
- Muligvis håndtere trickle ICE

Dette er en *nice to have* rettere end en *need to have*. De to peers indsamler ICE-kandidater allerede *inden* SDP pakkerne udveksles, og sender disse med SDP pakkerne, så trickle ICE ville efter undertegnede forståelse blot optimere peer-to-peer forbindelsen efterhånden som bedre ICE kandidater bliver fundet. Udviklingen af WebRTC samtale-softwaren har vist sig at være en særligt krævende proces. Til trods for, at denne på nuværende tidspunkt ikke er færdig og implementeret er undertegnede sikker på at dette blot er et spørgsmål om tid. For at forkorte processen kunne en ekstern signaling server have været anvendt. For at sikre funktionaliteten som ønsket skal modulet skrives om fra start, denne gang i javascript, som intentionen er med WebRTC.

6. Bevis (HB, SØF, HJL)

6.1 Samlet system (HB)

<https://youtu.be/z0fyJ8KGEjo>

6.2 Audio_handling (SØF)

Youtube video, der viser audio_handlings funktionalitet gennem et simpelt test-script

Dette script kan ses under **Bilag 10 – kode til audio_handling bevis**

https://www.youtube.com/shorts/_TzQrx5xAbo

6.3 WebRTC klient og Signaling server (SØF)

6.3.1 Signaling server

Den endelige kode kan findes under **Bilag 11 – Signaling server, endelig signaling server kode**

```
"C:\Users\sf97\PycharmProjects\Eksamensprojekt 2. sem\Audio stream 1\.venv\Scripts\python.exe" "C:\Users\sf97\PycharmProjects\Eksamensprojekt 2. sem\Audio stream 1\signaling.py"
Peer '87395c05-1b86-4a7b-b177-7893ae84bc3f' : xIehFXY9abM_QD9_AAAB connected!
Peers: {'87395c05-1b86-4a7b-b177-7893ae84bc3f': 'xIehFXY9abM_QD9_AAAB'}
Peer_id sent!
emitting event "peer_id" to all []
2024-12-05 09:55:51,299 - INFO - emitting event "peer_id" to all []
Sent list of available peers to xIehFXY9abM_QD9_AAAB.
- Available peers: []
received event "get_peers" from xIehFXY9abM_QD9_AAAB []
2024-12-05 09:55:51,398 - INFO - received event "get_peers" from xIehFXY9abM_QD9_AAAB []
emitting event "peer_list" to xIehFXY9abM_QD9_AAAB []
2024-12-05 09:55:51,398 - INFO - emitting event "peer_list" to xIehFXY9abM_QD9_AAAB []
Peer 'b697c693-fee5-4bc7-9e7e-bf1c67c354a5' : 4HyxEg_GQTr0hRm_AAAD connected!
Peers: {'87395c05-1b86-4a7b-b177-7893ae84bc3f': 'xIehFXY9abM_QD9_AAAB', 'b697c693-fee5-4bc7-9e7e-bf1c67c354a5': '4HyxEg_GQTr0hRm_AAAD'}
Peer_id sent!
emitting event "peer_id" to all []
2024-12-05 09:55:52,808 - INFO - emitting event "peer_id" to all []
Sent list of available peers to 4HyxEg_GQTr0hRm_AAAD.
- Available peers: ['87395c05-1b86-4a7b-b177-7893ae84bc3f']
received event "get_peers" from 4HyxEg_GQTr0hRm_AAAD []
2024-12-05 09:55:52,892 - INFO - received event "get_peers" from 4HyxEg_GQTr0hRm_AAAD []
emitting event "peer_list" to 4HyxEg_GQTr0hRm_AAAD []
2024-12-05 09:55:52,892 - INFO - emitting event "peer_list" to 4HyxEg_GQTr0hRm_AAAD []
received event "offer" from 4HyxEg_GQTr0hRm_AAAD []
2024-12-05 09:55:57,917 - INFO - received event "offer" from 4HyxEg_GQTr0hRm_AAAD []
emitting event "offer" to xIehFXY9abM_QD9_AAAB []
2024-12-05 09:55:57,918 - INFO - emitting event "offer" to xIehFXY9abM_QD9_AAAB []
Forwarded offer from b697c693-fee5-4bc7-9e7e-bf1c67c354a5 to '87395c05-1b86-4a7b-b177-7893ae84bc3f'
Forwarded answer from '87395c05-1b86-4a7b-b177-7893ae84bc3f' to 'b697c693-fee5-4bc7-9e7e-bf1c67c354a5'
2
1
2
2
2
2
received event "answer" from xIehFXY9abM_QD9_AAAB []
2024-12-05 09:56:02,954 - INFO - received event "answer" from xIehFXY9abM_QD9_AAAB []
received event "sdp_done" from xIehFXY9abM_QD9_AAAB []
2024-12-05 09:56:02,955 - INFO - received event "sdp_done" from xIehFXY9abM_QD9_AAAB []
emitting event "answering" to 4HyxEg_GQTr0hRm_AAAD []
2024-12-05 09:56:02,955 - INFO - emitting event "answer" to 4HyxEg_GQTr0hRm_AAAD []
received event "sdp_done" from 4HyxEg_GQTr0hRm_AAAD []
2024-12-05 09:56:02,961 - INFO - received event "sdp_done" from 4HyxEg_GQTr0hRm_AAAD []
emitting event "all_done" to all []
2024-12-05 09:56:02,962 - INFO - emitting event "all_done" to all []
```

6.3.2 WebRTC klient

Den endelige kode til WebRTC klienten kan findes under **Bilag 12 – WebRTC Client, Endelig WebRTC klient kode**

Wireshark .pcapng fil, der viser trafik mellem to peers:

https://drive.google.com/file/d/15WS40H06sosSAbYy6K3Xf3opVRH96zZr/view?usp=drive_link

Video, der viser en P2P forbindelse mellem to peers på samme host:

<https://youtu.be/3GiaFMxZ3DY>

Video, der viser en P2P forbindelse mellem en PC og en RPI. Udveksling af pakker, og pakkers størrelse vises gennem Wireshark:
<https://youtu.be/ydjY7AjvqyM>

Wireshark filen fra videoen:

https://drive.google.com/file/d/1IDt5Gdv9NhBrJLYLy4mKojhu2Fg5-hW-/view?usp=drive_link

Læg mærke til mikrofon-ikonet i nedre højre hjørne i begge ovenstående videoer, som indikerer, at programmet får adgang til computerens mikrofon for at opsamle audio data.

WebRTC debug log fra PC peer som .pdf:

https://drive.google.com/file/d/1tG5Hq-FcLcgDoDAPYxItdG01h8bvUsUJ/view?usp=drive_link

6.4 Bevis for succesfuld test af PIR Sensor

[Youtube Video af succesfuld test, PIR Sensor.](#)

7. Projektledelse (HJL)

Indledende projektledelse og planlægning - uge 45. (HJL)

Under projektarbejde i uge 45 havde vi i fællesskab fokus på design, overblik og planlægning af hele projektet, dets omfang, komponenter, og den generelle tilgang til projektet som helhed. Vi kørte i gruppen en kort white-hat session for at samle inspiration og sammenligne vores foreløbige produkt-idé med de professionelle alternativer som var på markedet på daværende tidspunkt.

Vi rundede uge 45 af med at lave en komplet produktliste. Denne vurderede vi som værende kritisk, eftersom denne skulle fungere som baggrunden for vores WBS - Work Breakdown Structure - som vi havde planlagt til at skulle laves og færdiggøres i starten af uge 46. Herunder blev det ligeledes planlagt, at uge 46 hovedsageligt skulle gå med design- og planlægning af WBS, diverse charts, Gannt skema og designe software.

7.1 PBS. Fuldkommen produktliste (SØF, HJL, SHK, HB):

1. Kamera

1.1 Live-Feed

1.1.1 Software

1.1.1.1 Håndter live-feed(Tænde og slukke)

1.1.2 Integration med HTML

1.2 Optagelse

1.2.1 Software

1.2.1.1 Håndtering af optagelse

1.2.1.2 Lagring af div. Optagelse

1.2.1.3 Afsend optagelse til database

1.3 Ansigtsgenkendelse

1.3.1 Software

1.3.1.1 Machinelearning

1.3.1.2 Database integration

2. Højtalere

2.1 Software

2.1.1 Web integrering

2.1.2 Afspil lyd (Tale og Alarm)

2.1.3 Tovejskommunikation

3. Mikrofon

3.1 Software

3.1.1 Lydininput

3.1.2 Lydoptagelse

3.1.3 Lagring af optagelse

4. PIR sensor

4.1 Software

4.1.1 Bevægelsesgenkendelse

4.1.1.1 Aktivering af kamera

5 Database

5.1 Cloud

5.2 Software

5.2.1 Lokal lagring

5.2.2 Upload/link

6. RPI 4B+ processor

6.1 Software

6.1.1 OS

7. Evt. dørsegment eller LED

7.1 Software

7.1.2 Lys

8. Webinterface

8.1 Frontend

- 8.1.1 HTML og CSS, hjemmeside layout
- 8.1.2 CRUD, særligt henblik på READ og DELETE
- 8.1.3 Videoafspiller
- 8.1.4 Konfigurationsmenu
- 8.1.5 Døråbning og alarm knap

8.2 Backend

- 8.2.1 CRUD
- 8.2.2 Routing, flask
- 8.2.3 Video og lyd afspiller
- 8.2.4 Døråbning og alarmkald
- 8.2.5 Software til håndtering af opkald mellem webinterface og kameradims
- 8.2.6 Gem konfigurationer

9. Master software

7.2 WBS. Faseopdelt (HJL, HB, SØF, SHK)

Nr.	Aktivitet	Ansvar	Depende ns	Varighe d
1	Rapport og dokumentation (Løbende)	Alle	6	100T
	Planlægningsfase			

2	Initieringsmøde	Alle	-	4T
3	Gantt diagram	Alle	2	4T
4	Opgavefordeling	Alle	2	2T
5	Kravspecifikation	Alle	2	4T
6	Initiering af Rapportskrivning og dokumentation	Alle	2	2T
	Design og Development fase			
7	<u>Design modul diagram</u>	HJL, SØF	5	4T
8	Design formel modultest kode	Fælles, individuel	7	6T
9	Design elektronik diagram	SØF	7	4T
10	Design backend på web interface	SHK	7	6T
11	Design frontend på web interface	HJL	10	4T
12	Design og opsætning af database	SHK	7	4T
13	Design databasehåndtering software	SHK	12	2T
14	<u>Design mikrofon og højtalere software</u>	SØF	7	4T
15	<u>Design kamera software</u>	HB	7	4T
16	Design software til samling af lyd og video	SØF, HB	14, 15	4T

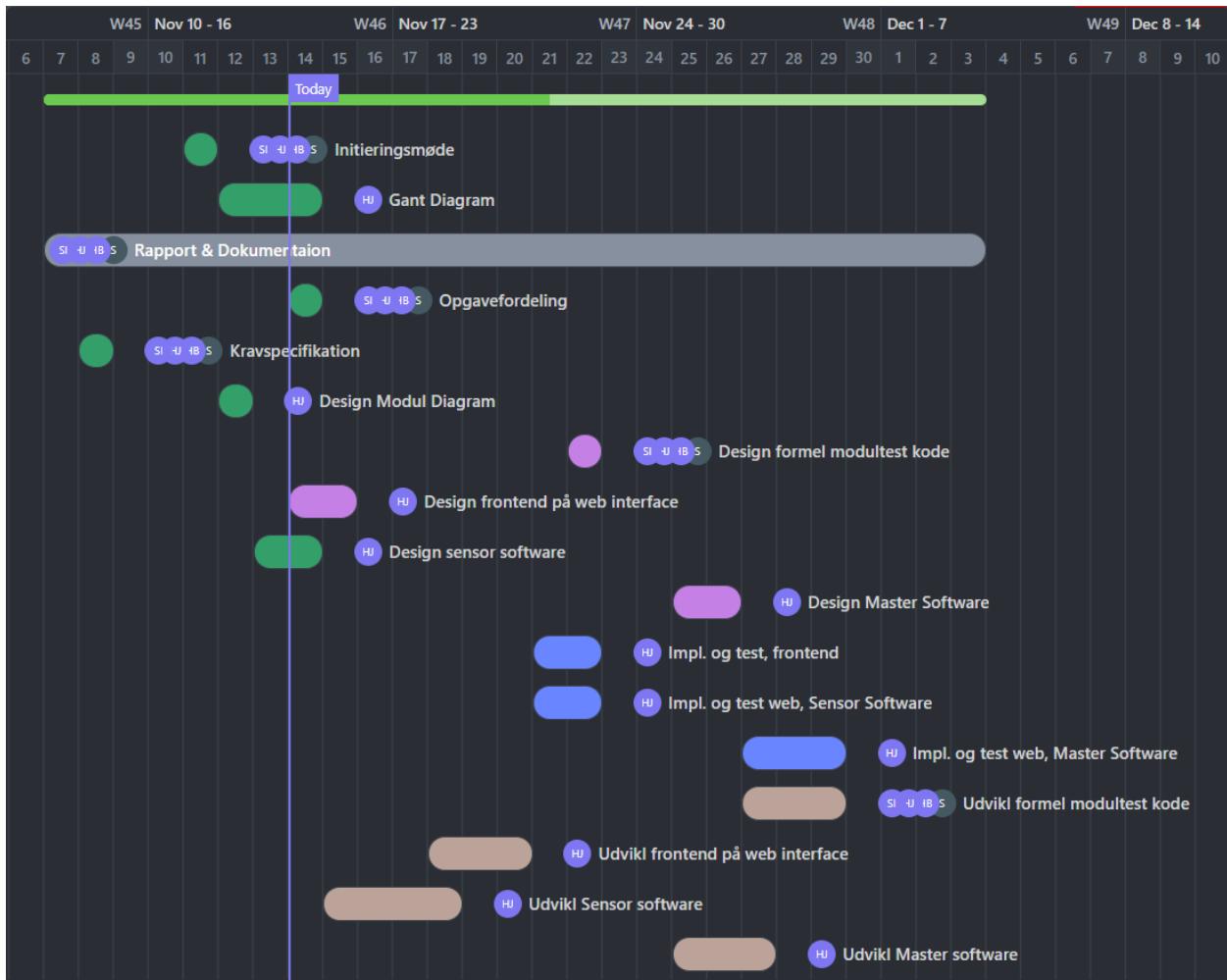
17	Design sensor software	HJL	7	2T
18	Design samtale software	HB, SØF	7	4T
19	Design Master Software	HJL	11, 12, 16, 17, 18	8T
20	Udvikl formel modultest kode	Fælles, Individuel	8	12T
21	Udvikl frontend på web interface	HJL	11	6T
22	Udvikl backend på web interface	SHK	10	12T
23	Udvikl databasehåndtering software	SHK	13	10T
24	<u>Udvikl mikrofon og højtalер software</u>	SØF	14	10T
25	<u>Udvikl kamera software</u>	HB	15	10T
26	Udvikl software til samling af lyd og video	SØF, HB	16	4T
27	Udvikl sensor software	HJL	17	10T
28	Udvikl samtale software	HB	18	8T
29	Udvikl Master Software	HJL	19	10T
	Implementerings- og testfase			
30	Impl. og test web, backend	SHK	22	6t
31	Impl. og test web, frontend	HJL	21	6t

32	Impl. og test, Database	SHK	23	4t
33	<u>Impl. og test, Lyd-samtale software (Mic & højtaler)</u>	SØF, HB	24, 26, 28	8T
34	<u>Impl. og test, Kamera software</u>	HB	25	8T
34.1	<u>Impl. og test lyd og video-samlings sftw</u>	SØF	24, 25, 26	8T
35	Impl. og test web, Sensor software	HJL	27	4T
36	<u>Impl. og test web, Master Software</u>	HJL	29, 30, 31, 32, 33, 34, 34.1, 35	12T
	Afslutningsfasen			
37	<u>Gennemgang af Rapport- og dokumentation</u>	Fælles	1	24T
38	Evaluering	Fælles	alt	1t

7.3 Gannt Skema (HJL)

7.3.1 Første udkast. 14.11.2024

Det første udkast af vores Gannt skema blev udfærdiget d. 14.11.2024. Vores indledende tidsestimering gjorde, at skema'et første udkast så således ud:



7.3.2 Gannt Skema. Endelig version.

Som del af undervisningen til Projekt Styring blev vi undervist i, at estimering af arbejdstid og dermed også arbejdsfordeling i størstedelen af tilfælde, udelukkende estimeres ud fra erfaring. Eftersom vi ingen erfaring har inden for projektstyring- og ledelse, gjorde vi vores bedste forsøg ift. at estimere tidsforbruget på vores enkelte arbejdsopgaver.

Undertegnede tilføjede ligeledes afhængigheder til arbejdsopgaverne. På denne vis fik vi et overblik over, hvilke opgaver der var afhængige – og blokerende – for hinanden.

Vi eksporterede vores endelige Gannt skema i .pdf format. Denne er vedhæftet opgaven og ses ligeledes et billede af skemaet i bilags-sektionen²².

7.3.3 Gannt. Indsnævring af omfang og justering af tidsplan. (HJL)

Mandag d. 2/12/2025 måtte vi indse, at vi ikke havde formået at overholde bestemte opgavers deadlines i vores WBS-Gannt. Herunder iblandt udvikling og test af Master software, som er vores styringsprogram til selve produktet.

Udvikling, test og implementering af master software blev udvidet fra den originale deadline d. 27/11 til at vare t.o.m. 4/12.

Ligeledes oplevede SØF problemer med test og implementeringen af audio-handling softwaren hvilket forårsagede, at deadlinen blev sprængt. SØF var herunder nødsaget til at arbejde videre på denne opgave indtil d. 3/12. Se afsnittet vedr. "audio-handling" for yderligere.

Vores endelige rapportskrivnings-deadline blev grundet ovenstående forsinkelser derfor flyttet fra d. 3/12 til afleveringsdagen d. 6/12.

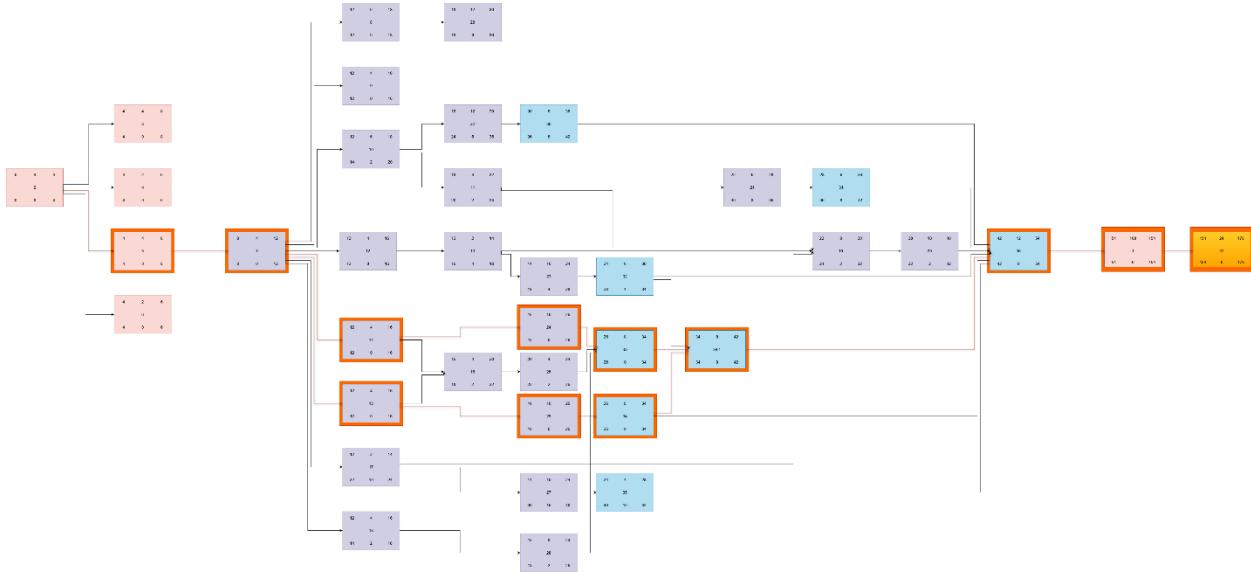
7.4 Relations diagram med kritisk sti (SØF)

Ud fra vores færdige WBS kunne undertegnede udarbejde et aktivitets relations diagram. Herefter lavede undertegnede et *forward pass* for at udregne de tidligste tidspunkter i processen hvor på en aktivitet kunne påbegyndes og færdiggøres. Dette efterfulgtes af et *backward pass*, for at finde de seneste tidspunkter hvor på aktiviteterne skulle være begyndt og afsluttet. Til sidst udregnedes mængden af slæk for hver opgave, og den kritiske sti blev identificeret og markeret med rødt.

Dette aktivitetsdiagram tager udgangspunkt i en WBS, der med høj sandsynlighed har ramt forbi ift. tidsestimeringer, og er derfor kun et vejledende værktøj. Med mere erfaring vil vores tidsestimeringer i WBS'en ramme tættere på virkeligheden, og derved vil vores relations

²² Bilag 4. Gannt Skema.

diagrammer også blive mere præcise og nyttige. Ikke desto mindre er det et godt værktøj til at holde overblikket over hvilke opgaver, der skal løses hvornår.



8. Perspektiv / refleksioner (SHK)

Refleksion af proces

Alle medlemmer af gruppen har i projektets samlede forløb benyttet deres tid så produktivt som muligt, og har dedikeret så meget tid til dette projekt som realistisk muligt. Vi benyttede ofte tiden fysisk tilstedevarende på fælles uddannelsessted. Hertil forekom der ofte dage med hjemmearbejde, hvis disse arbejdsopgaver tillod dette. Som gruppe kunne der forbedres estimering af deadlines og arbejdsopgaver, da der i flere arbejdsopgaver var estimeret for kort tid. Dette gjorde at nogle arbejdsopgaver tog længere tid end forventet. For gruppen var det også forventeligt at estimering af tid, er en svær opgave at håndholde, da projektet spænder bredt over mange aspekter. Gruppen er udover dette tilfredse over projektets samlede proces.

Refleksion af produktet som helhed

Vi har som gruppe lært, at konteksten for dette projekt har haft stor betydning for udfaldet. Ambitionsniveauet har været højt, og vi undervurderede, hvor meget arbejde der kræves for at udvikle et produkt med så mange funktioner. Hvis vi skulle lave et lignende produkt igen, ville det kræve en mere professionel ramme omkring projektet. Det ville indebære et større fokus på planlægning, bedre struktur og måske også et samarbejde med personer, der har mere erfaring inden for de relevante teknologier. Med de erfaringer, vi har fået her, er det tydeligt, at en mere professionel tilgang og kontekst ville være nødvendigt for at sikre, at produktet kan lykkes.

9. Konklusion (HB, SØF, SHK, HJL)

Ved dette projekt satte vi os for at udvikle et tryghedsskabende system, der tilgodeser den private borger ved at kunne streame, optage og lagre både video og lyd, samt kunne genkende ønskede ansigter. Systemet skulle ligeledes kunne tilgås gennem et webbaseret interface.

Projektet har i sin helhed være en delvis succes. De fleste komponenter virker som ønsket. Vores produkt er dog ikke endt med at opfylde det fulde omfang som originalt beskrevet.

Integrationsprocessen af flere af komponenterne har undervejs skabt problemer. Vores WebRTC peer-to-peer forbindelse kan oprettes gennem vores signaling server, og vedligeholdes uden denne, og data kan udveksles. Dog kan vores WebRTC klient ikke sikre, at den korrekte mængde audio data overføres til pakker *udelukkende* når klienten kører på en Raspberry Pi 4B.

Produktets PIR sensor virker som ønsket, idet den håndteres i styringsdelen af softwaren, hvor sensoren formår at aktivere produktets kamera.

Den cloud-baserede database fungerer som forventet og ønsket, da denne kan tilgås og administreres direkte fra PgAdmin. Hertil kan der modtages metadata fra det samlede systemprogram og hertil kan der også fjernes data fra web-interface.

Herudover fungerer uploadvideo.py og app.py, som forventeligt. Uploadvideo kan tage den senest lagrede video og uploade metadataen med korrekte informationer. Flask app kan benyttes som GUI til at navigere rundt i de forskellige routes, og benytte de forskellige routes funktioner som ønsket.

Kameraet udfører den ønskede funktionalitet i relation til programmet, det modtager et signal fra PIR sensoren om at starte og udfører sine handlinger baseret på de input den får fra billederne, der læses gennem linsen. Hvis et ukendt ansigt detekteres, gemmes der en video med lyd og tidsstempel lokalt på RPI 4B og videoen kan tilgås gennem vores webinterface.

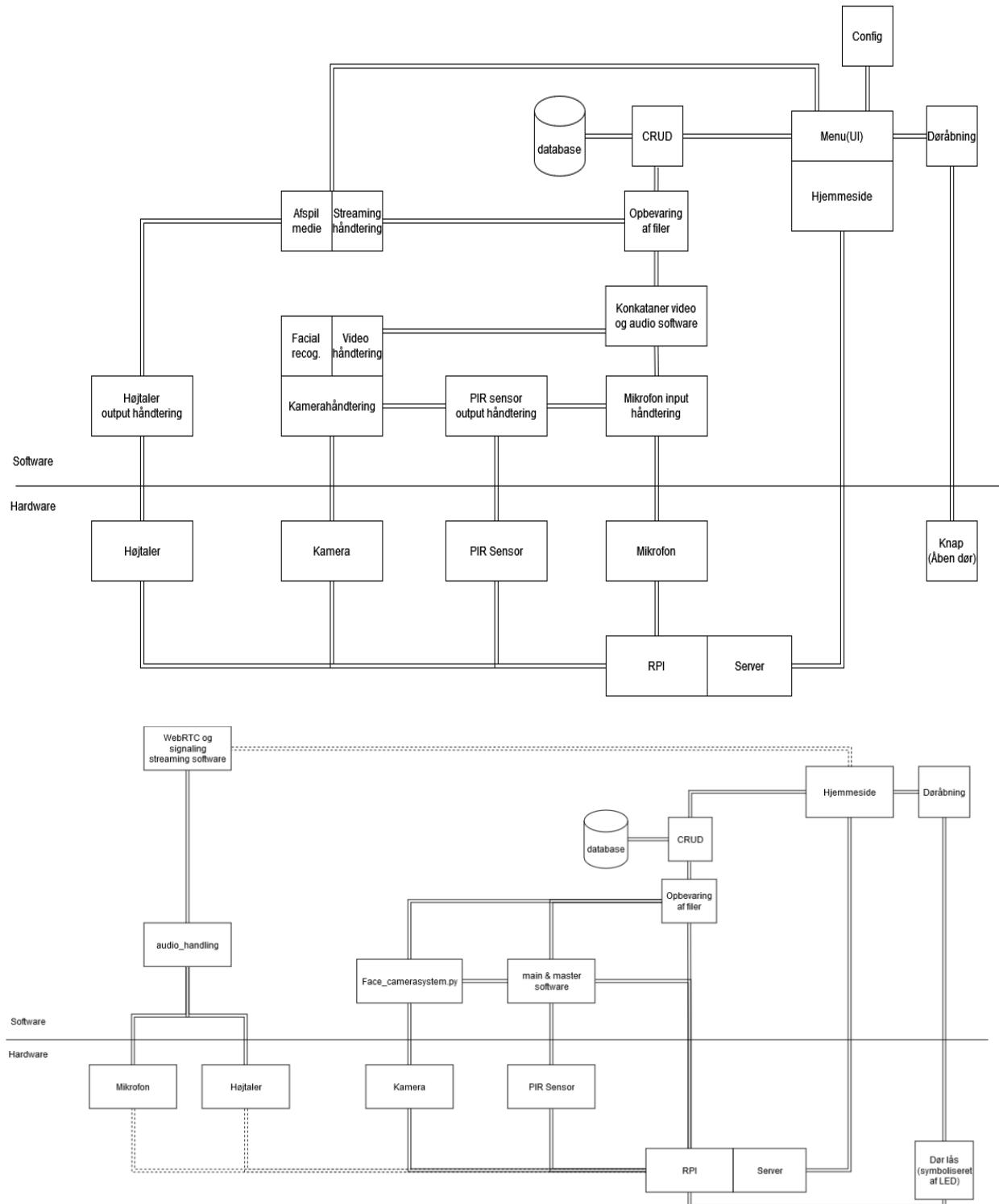
Vi har ikke formået at implementere, at systemet kan lave alarmkald eller låse/låse op for en elektronisk lås. For at simulere låsen opsatte vi i stedet en test i form af en Blå LED, som repræsenterer en låse-funktion. Denne er implementeret i vores web-interface således, at det blå LED-lys kan tændes og slukkes via vores hjemmesides interface.

Projektledelse- og projektstyringsværktøjer er undervejs blevet benyttet til at holde styr på tidsplanen og overblikket over projektet. Herunder har vi undervejs været nødsaget til at nedjustere omfanget grundet mangel på ressourcer i form af tid.

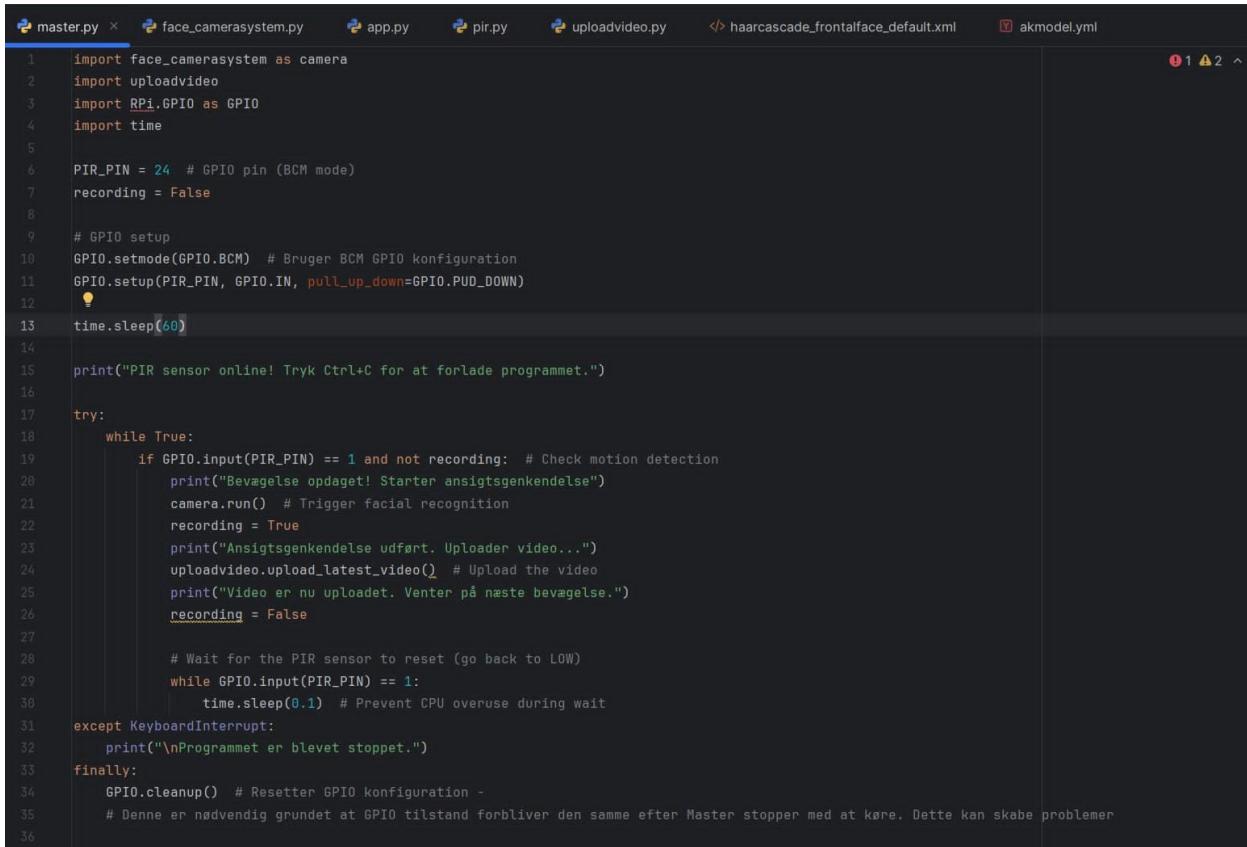
Opsummerende kan vi således konkludere, at vi er opnået en delvis succes. Vi er nået i mål med store dele af vores originale mål, men har samtidig set os nødsaget til at nedjustere på visse parametre af omfanget i projektet.

10. Bilag og tillæg

Bilag 1 – Moduldiagram (SØF, HJL)



Bilag 2. Sensor Python Modul (HJL)



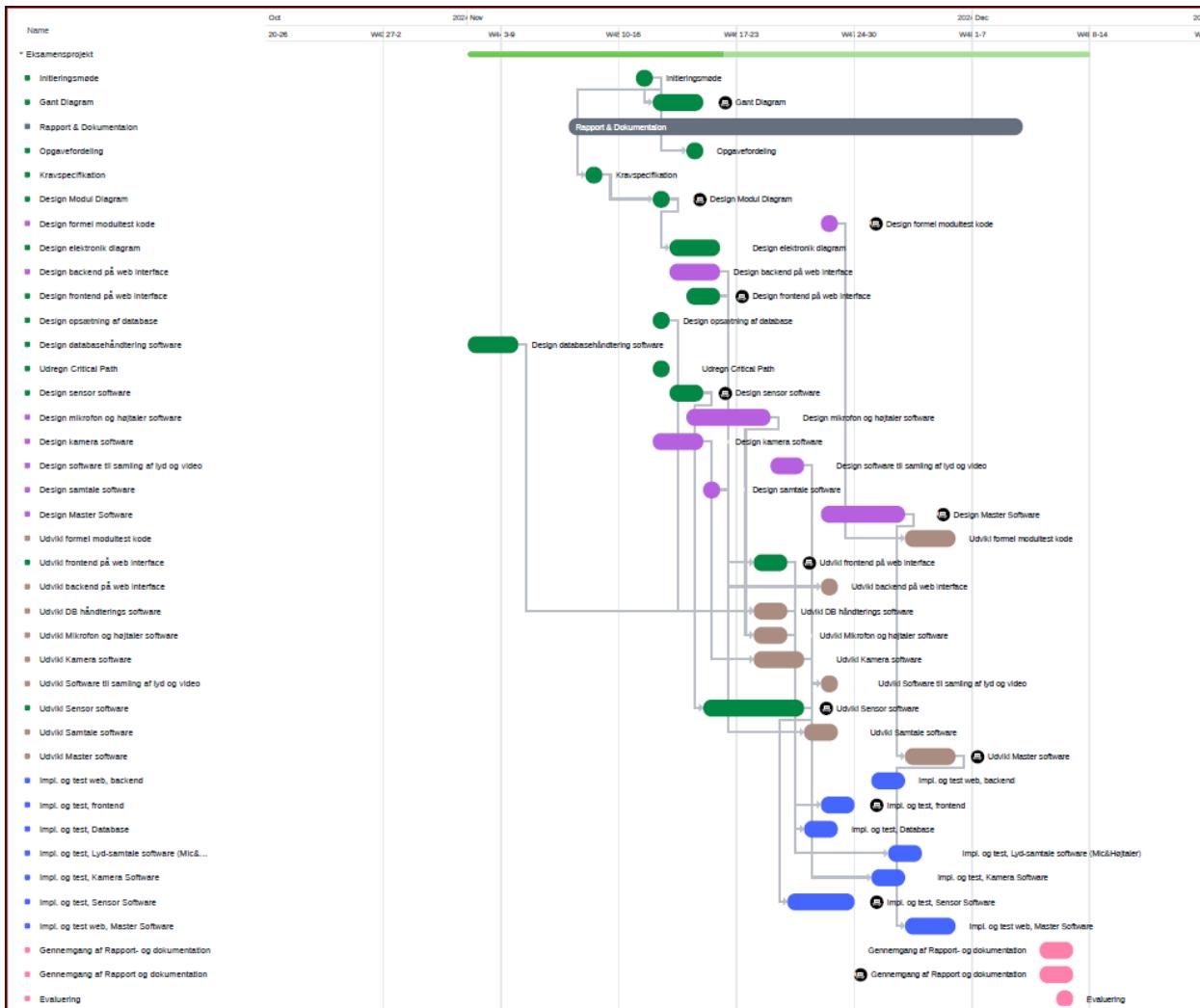
The screenshot shows a code editor with several tabs at the top: master.py, face_camerasytem.py, app.py, pir.py, uploadvideo.py, haarcascade_frontalface_default.xml, and akmodel.yml. The master.py tab is active, displaying the following Python code:

```
1 import face_camerasytem as camera
2 import uploadvideo
3 import RPi.GPIO as GPIO
4 import time
5
6 PIR_PIN = 24 # GPIO pin (BCM mode)
7 recording = False
8
9 # GPIO setup
10 GPIO.setmode(GPIO.BCM) # Bruger BCM GPIO konfiguration
11 GPIO.setup(PIR_PIN, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
12
13 time.sleep(60)
14
15 print("PIR sensor online! Tryk Ctrl+C for at forlade programmet.")
16
17 try:
18     while True:
19         if GPIO.input(PIR_PIN) == 1 and not recording: # Check motion detection
20             print("Bevægelse opdaget! Starter ansigtsgenkendelse")
21             camera.run() # Trigger facial recognition
22             recording = True
23             print("Ansigtsgenkendelse udført. Uploader video...")
24             uploadvideo.upload_latest_video() # Upload the video
25             print("Video er nu uploadet. Venter på næste bevægelse.")
26             recording = False
27
28         # Wait for the PIR sensor to reset (go back to LOW)
29         while GPIO.input(PIR_PIN) == 1:
30             time.sleep(0.1) # Prevent CPU overuse during wait
31 except KeyboardInterrupt:
32     print("\nProgrammet er blevet stoppet.")
33 finally:
34     GPIO.cleanup() # Resetter GPIO konfiguration -
35     # Denne er nødvendig grundet at GPIO tilstand forbliver den samme efter Master stopper med at køre. Dette kan skabe problemer
36
```

Bilag 3. Succesfuld Sensor Test Video. (HJL)

<https://youtube.com/shorts/w8HwlkGpXqI?feature=share>

Bilag 4. Gannt skema. (HJL)



Bilag 5. Youtube video af Web-aktivering af LED.(HJL)

[Hyperlink](#)

Bilag 6 – Model træning og mappestruktur

6.1 Mappestruktur træningsmodel

```
MODEL_TRAINING          # Rodmappen til hele projektet

└── venv                # Virtuelt miljø til Python-pakker

└── picture_taker.py    # Python-script til at tage billeder

└── negative_generator.py # Genererer 100 forskellige billeder af 1

└── model_trainer.py    # Python-script til at træne modellen

└── recognize_tester.py  # Python-script til at teste modellen

└── aug_negative_images # Indholder billeder til negative_generator

└── positive_folder     # Indholder positive billeder

└── akmodel.xml         # XML-resultat af model_trainer.py

└── h_f_d.yml           # YML-fil til at detektere ansigter

└── read_me.txt          # Dokumentation eller vejledning
```

6.2 LBPH træner

picture_taker.py

```
import cv2
import time

def collect_images(faces_folder, user_id="user", num_samples=36):
    # Start webcam og ansigtsdetektor
    camera = cv2.VideoCapture(0)
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

    print(f"Indsamler {num_samples} billeder for bruger: {user_id}. Tryk 'q' for at stoppe tidligt.")

    count = 0
    while count < num_samples:
        # Tag et billede fra kameraet
        ret, frame = camera.read()
        if not ret:
            break

        # Find ansigter
```

```

gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)

for (x, y, w, h) in faces:
    # Beskær ansigtet
    cropped_face = gray[y:y+h, x:x+w]
    count += 1
    filename = f'{faces_folder}/{user_id}_{count}.jpg'
    cv2.imwrite(filename, cropped_face)

    # Tegn en firkant omkring ansigtet og vis billedet
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
    cv2.putText(frame, f'Billede {count}/{num_samples}', (10, 30),
               cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

cv2.imshow("Indsamler billeder", frame)

# Vent lidt mellem billederne
time.sleep(0.2)

# Luk programmet, hvis 'q' trykkes
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

# Luk kameraet og vinduet
camera.release()
cv2.destroyAllWindows()
print(f'{count} billeder gemt i: {faces_folder}')

# Start med at indsamle billeder
# Sørg for, at mappen allerede eksisterer, før koden køres
collect_images("faces_folder")

```

negative_generator.py

```

import os
import cv2

def generate_image_variations(input_folder, output_folder, num_variations=10):
    # Find alle billedfiler i input-mappen
    image_files = [f for f in os.listdir(input_folder) if f.endswith('.jpg')]

    for image_file in image_files:
        image_path = os.path.join(input_folder, image_file)
        image = cv2.imread(image_path)

```

```

if image is None:
    print(f"Kunne ikke læse billedet: {image_path}")
    continue

print(f'Laver variationer for: {image_file}')

# Generer variationer
for i in range(1, num_variations + 1):
    # Lav en simpel ændring: spejlvend billedet
    variation = cv2.flip(image, 1) # Horisontal flip

    # Gem det nye billede
    new_filename = f'{os.path.splitext(image_file)[0]}_var_{i}.jpg'
    output_path = os.path.join(output_folder, new_filename)
    cv2.imwrite(output_path, variation)

print(f'{num_variations} variationer lavet for: {image_file}')

# Brug funktionen
input_folder = "negative_images" # Mappe med originale billeder
output_folder = "augmented_images" # Mappe til gemte variationer (skal allerede eksistere)
generate_image_variations(input_folder, output_folder, num_variations=10)

```

model_trainer.py

```

import cv2
import os
import numpy as np

def model_trainer(positive_folder, model_path):
    # Opret ansigtsgenkendelsesmodel
    model = cv2.face.LBPHFaceRecognizer_create()

    # Lister til billeder og labels
    images = []
    labels = []

    # Gå gennem billederne i mappen
    for file_name in os.listdir(positive_folder):
        file_path = os.path.join(positive_folder, file_name)

        # Læs billedet som gråtone
        image = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)

```

```

# Tilføj billedet og label til listerne
images.append(image)
labels.append(1) # 1 er label for positive billeder

# Træn modellen med billeder og labels
print("Træner modellen...")
model.train(images, np.array(labels))

# Gem modellen til en fil
model.write(model_path)
print(f"Modellen er gemt som: {model_path}")

# Brug funktionen
positive_folder_path = "faces_folder" # Din mappe med billeder
model_output_file = "akmodel.yml"

model_trainer(positive_folder_path, model_output_file)

```

recognize_tester.py

```

import cv2

def recognize_face(model_path):
    # Indlæs ansigtsgenkendelsesmodellen og ansigtsdetektoren
    recognizer = cv2.face.LBPHFaceRecognizer_create()
    recognizer.read(model_path)
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
    'haarcascade_frontalface_default.xml')

    # Start webcam
    camera = cv2.VideoCapture(0)

    print("Tryk 'q' for at afslutte.")
    while True:
        ret, frame = camera.read()
        if not ret:
            print("Kan ikke få adgang til kameraet.")
            break

    # Konverter til gråtone og find ansiger
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)

    for (x, y, w, h) in faces:
        cropped_face = gray[y:y+h, x:x+w]

```

```

label, distance = recognizer.predict(cropped_face)

# Print resultater
if distance < 50: # Tærskel for genkendelse
    print(f"Genkendt ID: {label} med sikkerhed: {round(distance, 2)}%")
else:
    print("Ukendt ansigt")

# Luk programmet, hvis 'q' trykkes
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

# Luk kameraet
camera.release()

# Brug funktionen
recognize_face("akmodel.yml")

```

6.3 Kamera kode

```

import cv2
from datetime import datetime, timedelta
import os

class FaceDetect:
    def __init__(self, cascade_path):
        self.face_cascade = cv2.CascadeClassifier(cascade_path)
        self.duration = timedelta(seconds=20)

    def detect_faces(self, camera):
        end_time = datetime.now() + self.duration

        while datetime.now() < end_time:
            ret, frame = camera.read()
            if not ret:
                break

            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            faces = self.face_cascade.detectMultiScale(gray, scaleFactor=1.3,
minNeighbors=6, flags=0, minSize=(60, 60))

            if len(faces) > 0:
                return frame, faces

        return None, None

```

```

class FaceRecognition:
    def __init__(self, model_path):
        self.recognizer = cv2.face.LBPHFaceRecognizer_create()
        self.recognizer.read(model_path)

    def recognize_faces(self, frame, faces):
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        results = []
        for (x, y, w, h) in faces:
            face = gray[y:y + h, x:x + w]
            label, distance = self.recognizer.predict(face)
            results.append({
                "coordinates": (x, y, w, h),
                "label": label if distance < 50 else "Unknown",
                "distance": distance
            })
        return results

class FFmpegRecorder:
    def __init__(self, video_device="/dev/video0", audio_device="hw:3,0"):
        import subprocess
        self.subprocess = subprocess
        self.video_device = video_device
        self.audio_device = audio_device

    def record_video_with_audio(self, output_file_path, duration):
        command = [
            "ffmpeg",
            "-y",
            "-f", "v4l2",
            "-i", "/dev/video0",
            "-f", "alsa",
            "-ac", "1",
            "-i", "plughw:3,0",
            "-t", str(duration),
            "-c:v", "libx264",
            "-preset", "veryfast",
            "-c:a", "aac",
            "-b:a", "128k",
            output_file_path
        ]
        self.subprocess.run(command, check=True)

    def run():

```

```

cascade_path = "haarcascade_frontalface_default.xml"
model_path = "akmodel.yml"
recording_duration = 30
output_dir = "static/videos"

os.makedirs(output_dir, exist_ok=True)

face_detector = FaceDetect(cascade_path)
face_recognizer = FaceRecognition(model_path)
recorder = FFmpegRecorder()
camera = cv2.VideoCapture(0)

unknown_frame_count = 0
unknown_threshold = 10

while True:
    frame, faces = face_detector.detect_faces(camera)

    if frame is None and faces is None:
        break

    if faces is not None and len(faces) > 0:
        results = face_recognizer.recognize_faces(frame, faces)

        for result in results:
            label = str(result["label"])
            distance = result["distance"]
            print(f"Label: {label}, Distance: {distance:.2f}%")

            if label == "Unknown":
                unknown_frame_count += 1
                print(f"Ukendt ansigt detekteret i {unknown_frame_count}/{unknown_threshold} frames.")
            else:
                unknown_frame_count = 0

        if unknown_frame_count >= unknown_threshold:
            print("Starter optagelse på grund af vedvarende ukendt ansigt.")
            output_file = os.path.join(output_dir,
                                         f"{datetime.now().strftime('%Y-%m-%d_%H-%M-%S')}.mp4")
            camera.release()
            recorder.record_video_with_audio(output_file, recording_duration)
            camera = cv2.VideoCapture(0)
            unknown_frame_count = 0

```

```
camera.release()
cv2.destroyAllWindows()
print("Program afsluttet.")

if __name__ == "__main__":
    run()
```

Bilag 7. Database

7.1 – Tabeloverblik

Object Explorer Statistics Dependencies Dependents Processes public.video_files/... public.video_files/Videofiles/avnadmin@dbweb1

Query History

```
1 ✓ SELECT * FROM public.video_files
2 ORDER BY id ASC
```

Scratch Pad

Data Output

	id [PK] integer	filename text	created_at timestamp without time zone
1	9	2024-12-04_09-25-53.mp4	2024-12-04 09:26:44
2	10	2024-12-04_09-33-56.mp4	2024-12-04 09:34:47
3	11	2024-12-04_09-33-56.mp4	2024-12-04 09:35:45
4	12	2024-12-04_09-33-56.mp4	2024-12-04 09:42:36
5	13	2024-12-04_12-11-10.mp4	2024-12-04 12:12:01
6	14	2024-12-04_12-11-10.mp4	2024-12-04 12:12:41

Total rows: 6 of 6 Query complete 00:00:00.104 Lp 1 Col 1

Bilag 8. Web

8.1 – Home

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Eksamensprojekt</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles/stylemedfoto.css') }}>
```

```
</head>
<body>
    <div class="container">

        <header>
            <h1>Eksamensprojekt</h1>
            <p>Tryghedssystem</p>
            <p>Gruppe 2</p>
        </header>

        <nav>
            <a href="#">#home</a>
            
            </a>
            <a href="livefeed">Livefeed</a>
            <a href="bibliotek">Bibliotek</a>
            <a href="opendoor">Døråbning</a>
            <a href="deletevideo">Delete Video</a>
        </nav>

        <main>
            <p>PLACEHOLDER --- PLACEHOLDER --- PLACEHOLDER --- </p>
            
        </main>

        <footer>
            <p>2024 IT-Teknologi, 2. Semester, Gruppe 2.</p>
        </footer>
    </div>
</body>
</html>
```



8.2 – Library

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Bibliotek</title>
    <link rel="stylesheet" href="static/styles/stylemedfoto.css">
</head>
<body>
    <div class="container">
        <header>
            <h1>Bibliotek</h1>
        </header>

        <nav>
            <a href="">Home</a>
            <a href="livefeed">Livefeed</a>
            <a href="opendoor">Døråbning</a>
            <a href="deletevideo">Delete Video</a>
        </nav>

        <main>
            <a href="{{ url_for('home') }}" class="livefeed-link">
                
            </a>
        </main>
        <table>
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Filename</th>
                    <th>Hyperlink</th>
                </tr>
            </thead>
            <tbody>
                {% for video in videos %}
                <tr>
                    <td>{{ video[0] }}</td> <!-- ID -->
                    <td>{{ video[1] }}</td> <!-- Filename -->
                    <td><a href="/play/{{ video[1] }}">Play</a></td> <!-- Hyperlink -->
                </tr>
                {% else %}
                <tr>
                    <td colspan="3">No videos found.</td>
                </tr>
                {% endfor %}
                </tbody>
            </table>
        </div>
    </body>
</html>
```

Bibliotek

Home Livefeed Døråbning Delete Video

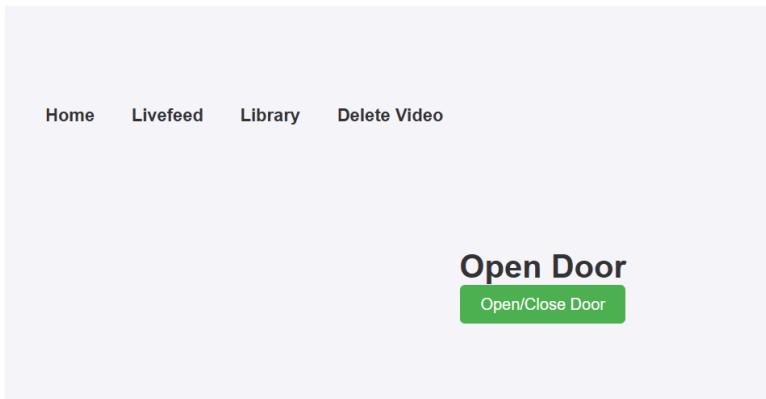


ID	Filename	Hyperlink
9	2024-12-04_09-25-53.mp4	Play
10	2024-12-04_09-33-56.mp4	Play
11	2024-12-04_09-33-56.mp4	Play
12	2024-12-04_09-33-56.mp4	Play
13	2024-12-04_12-11-10.mp4	Play
14	2024-12-04_12-11-10.mp4	Play

2024 IT-Teknologi, 2. Semester, Gruppe 2.

8.3 – Open Door

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Open Door</title>
    <link rel="stylesheet" href="static/styles/stylemedfoto.css">
<style>
    button {
        background-color: #4CAF50;
        border: none;
        color: white;
        padding: 10px 20px;
        font-size: 16px;
        cursor: pointer;
        border-radius: 5px;
    }
    button:hover {
        background-color: #45a049;
    }
</style>
<body>
    <nav>
        <a href="/">Home</a>
        <a href="livefeed">Livefeed</a>
        <a href="bibliotek">Library</a>
        <a href="deletevideo">Delete Video</a>
    </nav>
</head>
<div style="text-align: center; margin-top: 20%;">
    <h1>Open Door</h1>
    <form action="/opendoor" method="POST">
        <button type="submit">Open/Close Door</button>
    </form>
</div>
</body>
</html>
```



8.4 - Udkast af Live-Feed

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>livefeed</title>
    <link rel="stylesheet" href="html/static/styles/stylemedfoto.css">
</head>
<body>
    <div class="container">
        <header>
            <h1>livefeed</h1>
        </header>

        <nav>
            <a href="">Home</a>
            <a href="bibliotek">Bibliotek</a>
            <a href="opendoor">Døråbning</a>
            <a href="deletevideo">Delete Video</a>
        </nav>

        <main>
            <a href="{{ url_for('home') }}" class="livefeed-link">
                
            </a>
            <iframe
                src=
                width="640"
                height="360"
                frameborder="0"
                allow="fullscreen">
            </iframe>
        </main>

        <footer>
            <p>2024 IT-Teknologi, 2. Semester, Gruppe 2.</p>
        </footer>
    </div>
</body>
</html>
```

Livefeed

[Home](#) [Bibliotek](#) [Døråbning](#) [Delete Video](#)



8.5 – Delete Video

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Delete Video</title>
    <link rel="stylesheet" href="static/styles/stylemedfoto.css">
</head>
<body>
    <div class="container">
        <header>
            <h1>Delete Video</h1>
        </header>
        <nav>
            <a href="">Home</a>
            <a href="livefeed">Livefeed</a>
            <a href="bibliotek">Bibliotek</a>
            <a href="opendoor">Døråbning</a>
        </nav>
        <main>
            <h2>Available Videos</h2>
            <table border="1">
                <thead>
                    <tr>
                        <th>ID</th>
                        <th>Filename</th>
                        <th>Created At</th>
                    </tr>
                </thead>
                <tbody>
                    {% for video in videos %}
                    <tr>
                        <td>{{ video[0] }}</td> <!-- ID -->
                        <td>{{ video[1] }}</td> <!-- Filename -->
                        <td>{{ video[2] }}</td> <!-- Created At -->
                    </tr>
                    {% endfor %}
                </tbody>
            </table>
            <h2>Delete a Video</h2>
            <form method="POST">
                <label for="video_id">Enter Video ID to Delete:</label>
                <input type="number" name="video_id" id="video_id" required>
                <button type="submit">Delete</button>
            </form>
        </main>
        <footer>
            <p>2024 IT-Teknologi, 2. Semester, Gruppe 2.</p>
        </footer>
    </div>
</body>
</html>
```

Delete Video

Home Livefeed Bibliotek Døråbning

Available Videos

ID	Filename	Created At
9	2024-12-04_09-25-53.mp4	2024-12-04 09:26:44
10	2024-12-04_09-33-56.mp4	2024-12-04 09:34:47
11	2024-12-04_09-33-56.mp4	2024-12-04 09:35:45
12	2024-12-04_09-33-56.mp4	2024-12-04 09:42:36
13	2024-12-04_12-11-10.mp4	2024-12-04 12:12:01
14	2024-12-04_12-11-10.mp4	2024-12-04 12:12:41

Delete a Video

Enter Video ID to Delete:

2024 IT-Teknologi, 2. Semester, Gruppe 2.

Bilag 9 – Div. Kode

9.1 – app.py

```
app.py ×
 1 from flask import Flask, render_template, request, send_from_directory
 2 import os
 3 import psycopg2
 4 import RPi.GPIO as GPIO
 5 GPIO.cleanup()
 6
 7 app = Flask(__name__, template_folder = 'templates', static_folder = 'static')
 8 app.config['TEMPLATES_AUTO_RELOAD'] = True
 9
10 DB_CONFIG = {
11     "dbname": "Videofiles",
12     "user": "avnadmin",
13     "password": "AVNS_mHuad1pXmTqBmChDFb",
14     "host": "dbweb1-intro-1.f.aivencloud.com",
15     "port": "28400"
16 }
17
18 LED_PIN = 14 # GPIO pin for LED
19 GPIO.setmode(GPIO.BCM) # Use Broadcom pin numbering
20 GPIO.setup(LED_PIN, GPIO.OUT) # Set the pin as output
21
22 @app.route('/')
23 def home():
24     return render_template('home.html')
25
26 @app.route('/livefeed')
27 def livefeed():
28     return render_template('livefeed.html')
```

```

30 @app.route('/opendoor', methods=['GET', 'POST'])
31 def opendoor():
32     if request.method == 'POST':
33         # Check if the LED is currently ON or OFF and toggle it
34         if GPIO.input(LED_PIN): # LED is ON, so turn it OFF
35             GPIO.output(LED_PIN, GPIO.LOW)
36         else: # LED is OFF, so turn it ON
37             GPIO.output(LED_PIN, GPIO.HIGH)
38
39     # Render the page (page reloads after button press, no feedback)
40     return render_template('opendoor.html')
41
42 @app.route('/bibliotek')
43 def bibliotek():
44     conn = psycopg2.connect(**DB_CONFIG)
45     cursor = conn.cursor()
46     cursor.execute("SELECT id, filename, created_at FROM video_files")
47     videos = cursor.fetchall()
48     cursor.close()
49     conn.close()
50     return render_template("bibliotek.html", videos=videos)
51
52 @app.route('/play/<path:filename>')
53 def play_video(filename):
54     video_folder = os.path.join(app.root_path, 'static', 'videos')
55     print(f"Looking for file: {filename} in {video_folder}")
56     if os.path.exists(os.path.join(video_folder, filename)):
57         return send_from_directory(video_folder, filename)
58     else:
59         return "File not found", 404

```

```

@app.route('/deletevideo', methods=['GET', 'POST'])
def delete_video():
    conn = psycopg2.connect(**DB_CONFIG)
    cursor = conn.cursor()

    # Fetch all videos from the database
    cursor.execute("SELECT id, filename, created_at FROM video_files")
    videos = cursor.fetchall()

    if request.method == 'POST':
        video_id = request.form.get('video_id')

        # Fetch filename to delete the corresponding file
        cursor.execute("SELECT filename FROM video_files WHERE id = %s", (video_id,))
        result = cursor.fetchone()

        if result:
            filename = result[0]
            video_path = os.path.join(app.root_path, 'static', 'videos', filename)

            # Delete the file from the filesystem
            if os.path.exists(video_path):
                os.remove(video_path)
                print(f"Deleted file: {video_path}")
            else:
                print(f"File not found: {video_path}")

            # Delete from the database
            cursor.execute("DELETE FROM video_files WHERE id = %s", (video_id,))
            conn.commit()

    cursor.close()
    conn.close()
    return render_template("deletevideo.html", videos=videos) # Success message

```

```
95     cursor.close()
96     conn.close()
97     return render_template("deletevideo.html", videos=videos)
98
99
100 if __name__ == '__main__':
101     app.run(debug=True, host='0.0.0.0')
...  
...
```

9.2 – uploadvideo.py

```
uploadvideo.py x
1 import os
2 import psycopg2
3 from datetime import datetime
4
5 # Database connection details
6 DB_CONFIG = {
7     "dbname": "Videofiles",
8     "user": "avnadmin",
9     "password": "AVNS_mHuaad1pXmTqBmChDFb",
10    "host": "dbweb1-intro-1.f.aivencloud.com",
11    "port": "28400"
12 }
13
14 def upload_latest_video(directory):
15     """
16     Find the latest .mp4 file in the given directory and upload its details to the database.
17     """
18     try:
19         # Find the latest .mp4 file
20         files = [f for f in os.listdir(directory) if f.endswith(".mp4")]
21         if not files:
22             raise FileNotFoundError(f"No .mp4 files found in {directory}")
23         latest_file = max(files, key=lambda f: os.path.getctime(os.path.join(directory, f)))
24         filepath = os.path.join(directory, latest_file)
25
26         # Connect to the database and upload
27         conn = psycopg2.connect(**DB_CONFIG)
28         cursor = conn.cursor()
29         created_at = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
30         filename = os.path.basename(filepath)
31
32         query = """
33             INSERT INTO video_files (filename, created_at)
34             VALUES (%s, %s)
35             """
36         cursor.execute(query, (filename, created_at))
37         conn.commit()
38         cursor.close()
39         conn.close()
40         print(f"Uploaded {filename} to database.")
41     except FileNotFoundError as e:
42         print(e)
43
44 # Example usage
45 # upload_latest_video("/home/gruppe2/Desktop/env/static/videos") #directory bliver sat i parantesen
46
```

9.3 – master.py

```
master.py ×
1 import face_camerasytem as camera
2 import uploadvideo
3 import RPi.GPIO as GPIO
4 import time
5
6 PIR_PIN = 26 # GPIO pin (BCM mode)
7 recording = False
8 time.sleep(60)
9 # GPIO setup
10 GPIO.setmode(GPIO.BCM) # Bruger BCM GPIO konfiguration
11 GPIO.setup(PIR_PIN, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
12
13 print (GPIO.input(PIR_PIN))
14
15 print("PIR sensor online! Tryk Ctrl+C for at forlade programmet.")
16
17 try:
18     print("Prøver")
19     print(GPIO.input(PIR_PIN))
20     while True:
21         time.sleep(0.1)
22         print("Venter på PIR input")
23         print(GPIO.input(PIR_PIN))
24         if GPIO.input(PIR_PIN) == 1 and recording == False: # Check motion detection
25             print("Bevægelse opdaget! Starter ansigtsgenkendelse")
26             camera.run() # Trigger facial recognition
27             recording = True
28             print("Ansigtsgenkendelse udført. Uploader video...")
29             uploadvideo.upload_latest_video("/home/gruppe2/Desktop/env/static/videos") # Upload the video
30             print("Video er nu uploadet. Venter på næste bevægelse.")
31             recording = False
32
33         # Wait for the PIR sensor to reset (go back to LOW)
34         while GPIO.input(PIR_PIN) == 1:
35             time.sleep(0.1) # Prevent CPU overuse during wait
36     except KeyboardInterrupt:
37         print("\nProgrammet er blevet stoppet.")
38     finally:
39         GPIO.cleanup() # Resetter GPIO konfiguration -
40         # Denne er nødvendig grundet at GPIO tilstand forbliver den samme efter Master stopper med at køre. Dette kan skabe problemer
```

9.4 – main.py

```
1 # main.py
2
3 # Importer master-modulet
4 import master
5
6 if __name__ == "__main__":
7     master.run()
```

9.5 – startup.sh

```
#!/bin/bash

# Navigate to the env directory
cd ~/Desktop/env/

# Launch main.py in one terminal with the virtual environment activated
lxterminal --working-directory=/home/gruppe2/Desktop/env/ -e "bash -c 'source bin/activate; python3 main.py; exec bash'" &

# Launch app.py in another terminal with the virtual environment activated
lxterminal --working-directory=/home/gruppe2/Desktop/env/ -e "bash -c 'source bin/activate; python3 app.py; exec bash'" &

# Keep the script running to prevent the service from exiting
while true; do
    sleep 10
done
```

9.6 – env_startup.service

```
[Unit]
Description=Start Env Scripts at Boot
After=graphical.target
Requires=graphical.target

[Service]
Type=simple
ExecStart=/bin/bash -c "export DISPLAY=:0; export XDG_RUNTIME_DIR=/run/user/1000; bash /home/gruppe2/Desktop/env/startup.sh"
Environment=DISPLAY=:0
Environment=XDG_RUNTIME_DIR=/run/user/1000
User=gruppe2
Restart=always
RestartSec=10

[Install]
WantedBy=graphical.target
```

9.7 – Testvideo af uploadvideo & app.py

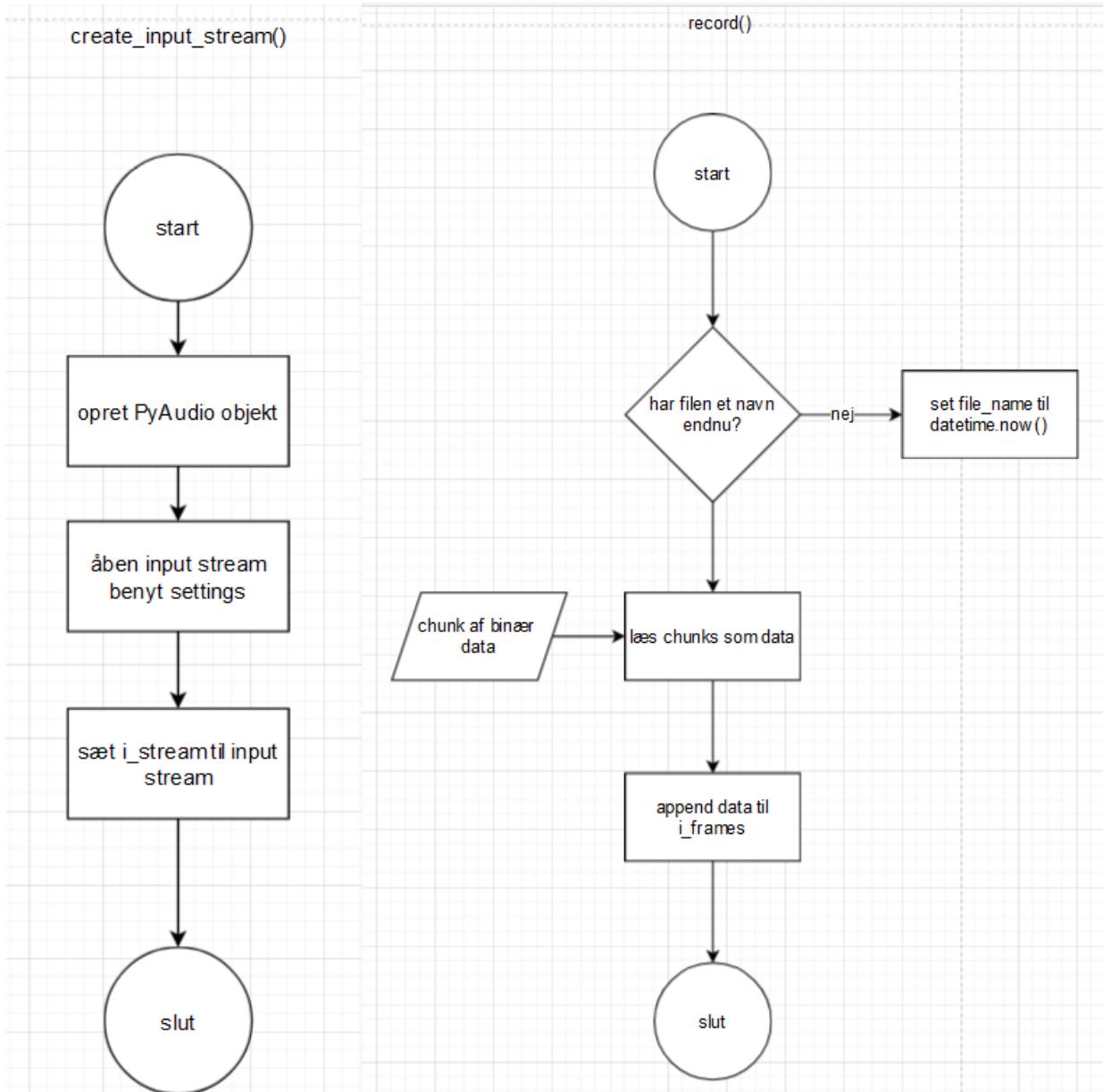
<https://youtu.be/AaMf-2BjZRU?si=SJ1ICXZ-yXAzFXbj>

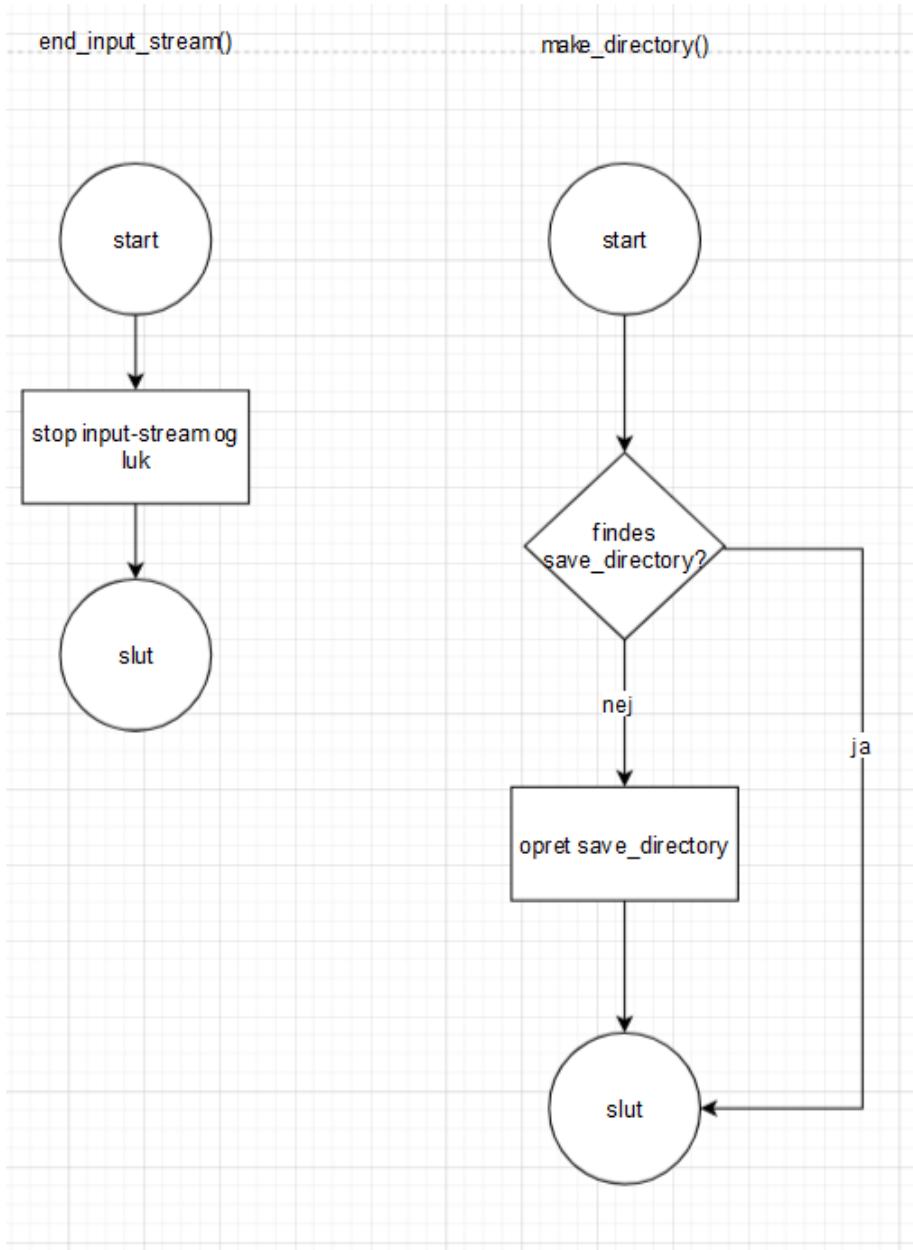
Bilag 10 – audio_handling (SØF)

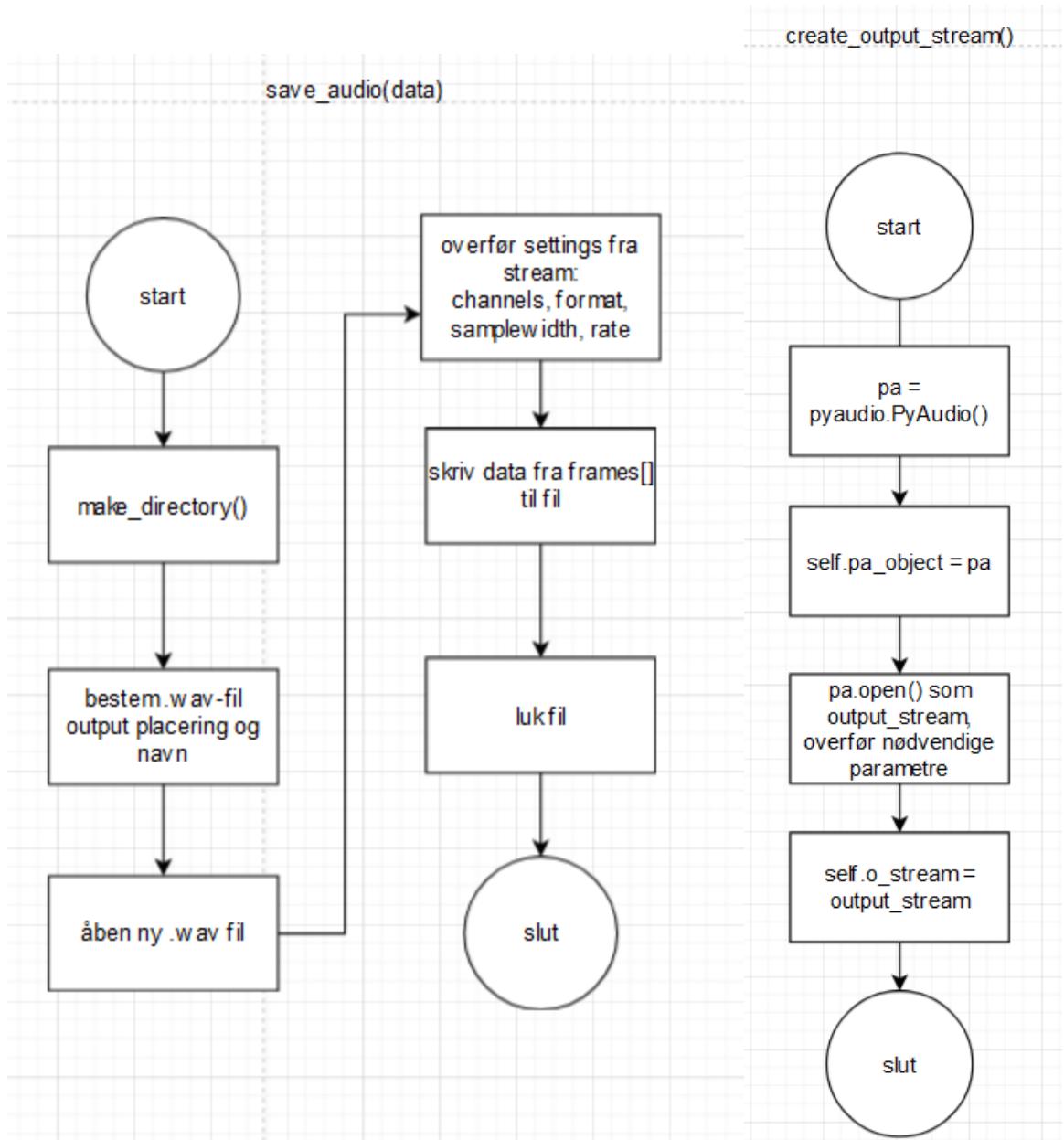
Formatteringen i disse bilag er underlig, og dette beklages – Word er en speciel størrelse.

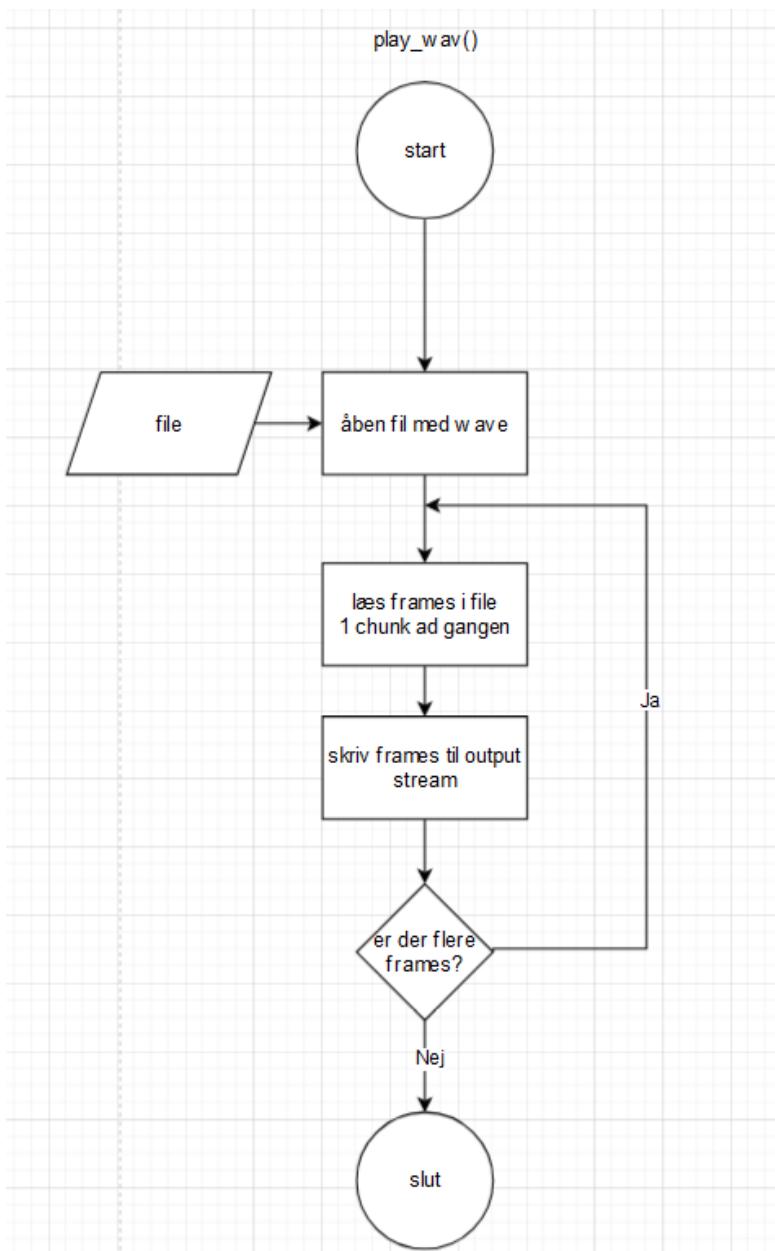
Ikke desto mindre håber undertegnede, at det er nogenlunde navigerbart, og kan give et fordybende indblik i audio_handling.

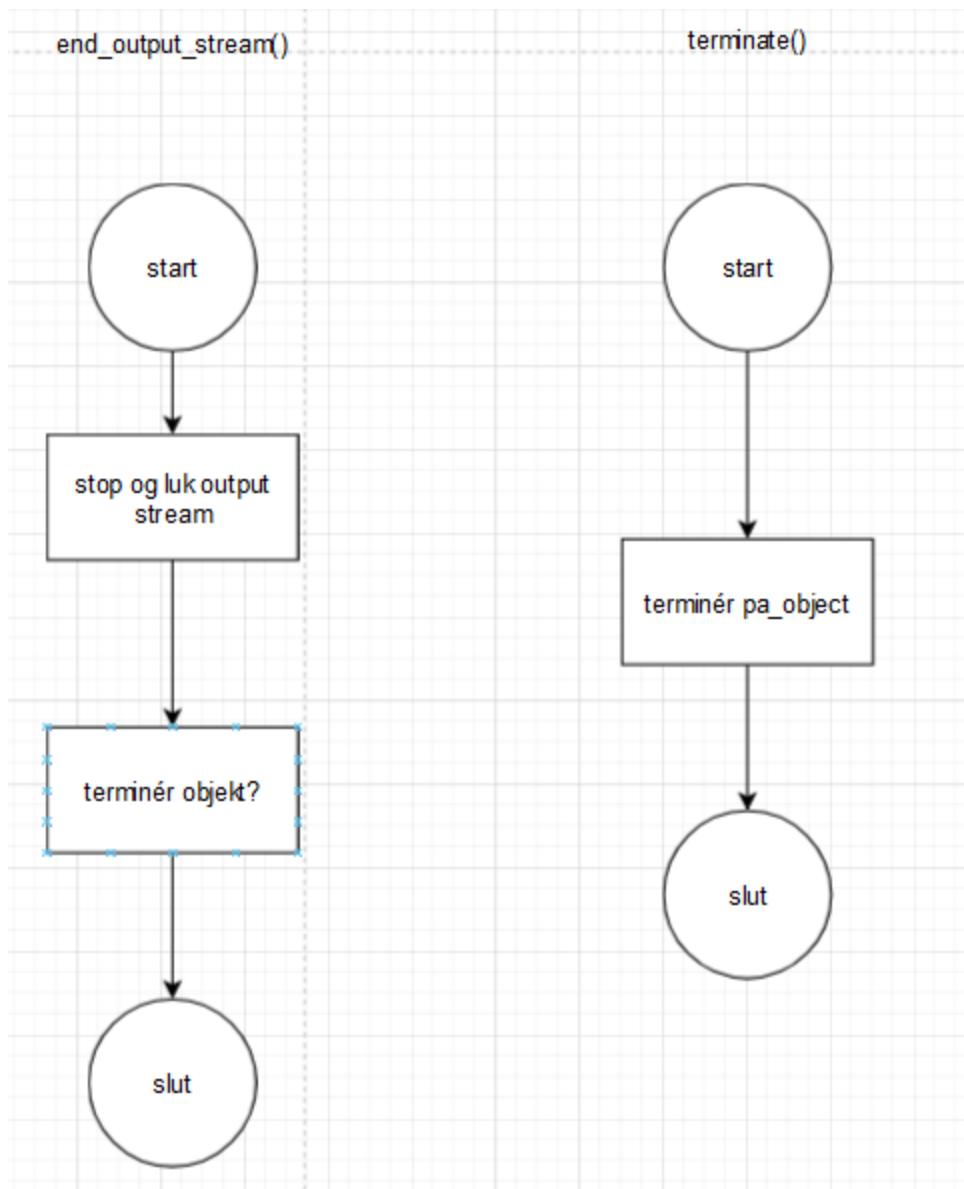
10.1 Flowcharts











10.2 Modultest kode

Modultest 1

```
# Håndter real-tid audio input og output
import pyaudio
# Håndter .wav file - reading og writing
import wave

# Audio settings
chunk = 1024
format = pyaudio.paInt16
channels = 1
rate = 44100
output_file = "recorded_audio.wav"

"""Chunk er en data-buffer, altså antallet af frames, der behandles ad
gangen.
Mindre chunks giver mindre latens, men kræver flere ressourcer.

Format specificerer lydformat i bits, paInt16 definerer formatet som 16-bit
audio samples.

Channels er antallet af kanaler - 1 er mono, 2 er stereo

Rate er sample-rate, altså hvor mange gange pr. sekund der tages en måling.
44100 er standard CD-kvalitet

output_file definerer navnet på den optagede fil."""

# Opret instans af PyAudio til at håndtere audio input og output
p = pyaudio.PyAudio()

# Åben USB mikrofon som input stream
input_stream = p.open(format=format,
                      channels=channels,
                      rate=rate,
                      input=True,
                      input_device_index=4) # Brug default input

output_stream = p.open(format=format,
                      channels=channels,
                      rate=rate,
```

```

        output=True,
        output_device_index=5) # Brug default output

print("Recording...")

# Opbevaring af optagede chunks
frames = []

try:
    for i in range(0, int(rate / chunk * 5)):
        data = input_stream.read(chunk)
        frames.append(data)
        #output_stream.write(data) # Playback in real-time
finally:
    print("Finished recording...")

    # Stop og luk streams
    input_stream.stop_stream()
    input_stream.close()
    # output_stream.stop_stream()
    # output_stream.close()

    # Gem optaget audio

    # Definer hvilken fil, der skal skrives til, og hvilken type data, der
    # skal skrives til den
    wf = wave.open(output_file, 'wb')
    # Sæt antal kanaler på lydfil (mono)
    wf.setnchannels(channels)
    # Sæt sample-width (16-bit)
    wf.setsampwidth(p.get_sample_size(format))
    # Sæt framerate (44100)
    wf.setframerate(rate)
    # Skriv binære frames til filen
    wf.writeframes(b''.join(frames))
    # Luk filen
    wf.close()

    # Ryd op efter pyaudio, frigør ressourcer
    p.terminate()

```

Kode til audio_handling bevis

```
import time
import audio_handling as a

# Opret audio_handling instans til input
audio_input = a.Input()

# Opret audio_handling instans til output
audio_output = a.Output()

try:
    # Initialiser input stream
    audio_input.create_input_stream()
    print("Recording audio for 5 seconds...")

    # Optag lyd i 5 sekunder
    start_time = time.time()
    while time.time() - start_time < 5:
        audio_input.record_to_file()
    print("Recording complete.")

    # Gem optaget lyd til en .wav fil
    audio_input.save_audio()
    print(f"Audio saved to:
{audio_input.save_directory}{audio_input.file_name}")

    # Find senest oprettede .wav fil i filstien
    latest_file = audio_output.get_latest_wav()
    print(f"Playing back: {latest_file}")

    # Initialiser output stream
    audio_output.create_output_stream()

    # Afspil optaget fil
    audio_output.play_wav(latest_file)
    print("Playback complete.")

finally:
    # Luk input- og output streams, ryd ressourcer op
    audio_input.close()
    audio_output.terminate()
```

10.3 Audio_handling kode

```
import pyaudio
import wave
import datetime
import os

"""
Optag lyd:

Key steps:
Definer audio settings:
    format, antal bits pr. sample
    bitrate
    antal kanaler
    filnavn
    buffer size - antal bits til behandling
Initialiser PyAudio, og åben en input-stream
Optag chunks af data, og gem i en liste
Stop optagelse, luk input-stream
Opret fil
Gem data i en fil, ved at samle chunks fra 'frames'
Luk fil

"""

class Audio:
    def __init__(self):
        self.chunk = 960
        self.format = pyaudio.paInt16
        self.channels = 1
        self.rate = 48000
        self.file_name = ""
        self.pa_object = object
        self.i_stream = object
        self.o_stream = object
```

```

# expanduser gør filstien linux-kompatibel, erstatter '~' med user
#
https://docs.python.org/3.12/library/os.path.html#os.path.expanduser
    self.save_directory = os.path.expanduser("~/mit-venv/wave-files/")

class Input(Audio):
    def __init__(self):
        super().__init__()
        self.input_device_index = None # None er default input device
        self.i_frames = []

    # Opret instans af PyAudio og åben input stream
    def create_input_stream(self):
        pa = pyaudio.PyAudio()
        self.pa_object = pa
        input_stream = pa.open(format=self.format,
                               channels=self.channels,
                               rate=self.rate,
                               input=True,

        input_device_index=self.input_device_index)
        self.i_stream = input_stream

    # Optag chunks af data og gem i liste
    def record_to_file(self):
        if not self.file_name:
            self.file_name = str(datetime.datetime.now().strftime("%Y-%m-
%d_%H-%M-%S")) + ".wav")
        try:
            data = self.i_stream.read(self.chunk,
exception_on_overflow=False)
        except IOError as e:
            print("Buffer overflow: ", e)
            data = None
        self.i_frames.append(data)

    def record_live(self):
        try:
            data = self.i_stream.read(self.chunk,
exception_on_overflow=False)

```

```

        return data
    except IOError as e:
        print("Error: ", e)

    # Afslut input stream
    # def end_input_stream(self):
    #     self.i_stream.stop_stream()
    #     self.i_stream.close()

    # Tjek om save directory findes. Hvis ikke oprettes dette.
    def make_directory(self):
        if not os.path.exists(self.save_directory):
            os.makedirs(self.save_directory)

    # Gem optagede frames til .wav fil
    def save_audio(self):
        self.make_directory()
        output_file = os.path.join(self.save_directory,
f"Recorded_Audio_{self.file_name}")
        with wave.open(output_file, 'wb') as wf:
            wf.setnchannels(self.channels)
            # 2 bytes, 16-bit sample width
            wf.setsampwidth(self.pa_object.get_sample_size(self.format))
#self.pa_object.get_sample_size(self.format)
            wf.setframerate(self.rate)
            wf.writeframes(b''.join(self.i_frames))
            wf.close()

    # Fjern objekt og frigør ressourcer
    def close(self):
        if self.i_stream:
            self.i_stream.stop_stream()
            self.i_stream.close()
        self.pa_object.terminate()

class Output(Audio):
    def __init__(self):
        super().__init__()
        self.output_device_index = None # None er default output device
        #self.o_frames = []

    # Opret output stream, angiv nødvendige parametre
    def create_output_stream(self):

```

```

pa = pyaudio.PyAudio()
self.pa_object = pa
output_stream = pa.open(format=self.format,
                        channels=self.channels,
                        rate=self.rate,
                        output=True,
                        output_device_index=self.output_device_index)
self.o_stream = output_stream

def list_files(self):
    # Lav en ny liste, 'wav_files', som indeholder alt
    # i 'self.save_directory', der slutter med '.wav'.
    # 'os.listdir()' viser alt i 'self.save_directory', men kun filer med
    # '.wav' file-extension appendes
    wav_files = []
    for f in os.listdir(self.save_directory):
        if f.endswith(".wav"):
            wav_files.append(f)
    # Håndtering af tomt 'self.save_directory' - hvis ingen filer er til
    stede, returneres None
    if not wav_files:
        return None
    else:
        return wav_files

# Find den sidst oprettede fil i en mappe
def get_latest_wav(self):
    # Find den senest oprettede fil i 'wav_files'
    wav_files = self.list_files()

    # Opret placeholder til seneste fil, samt en fil-alder
    latest_file = None
    latest_time = 0

    # Iterér over samtlige filer i wav_files
    for file in wav_files:

        # Find den fulde filsti, og find dens *modifikations-tid* siden
        # 1970
        # ved hjælp af os.path.getmtime()
        file_path_full = os.path.join(self.save_directory, file)
        file_time = os.path.getmtime(file_path_full)

```

```

        # Overfør ældste fil til placeholder variabel, opdater ny
latest_time
    if file_time > latest_time:
        latest_time = file_time
        latest_file = file

    return os.path.join(self.save_directory, latest_file)

# Afspil sidst oprettede fil i en mappe
def play_wav(self, file):
    with wave.open(file, 'rb') as wf:
        while True:
            data = wf.readframes(self.chunk)
            if not data:
                break
            self.o_stream.write(data)

# def end_output_stream(self):
#     self.o_stream.stop_stream()
#     self.o_stream.close()

def terminate(self):
    if self.o_stream and isinstance(self.o_stream, pyaudio.Stream):
        self.o_stream.stop_stream()
        self.o_stream.close()
    if self.pa_object and isinstance(self.pa_object, pyaudio.PyAudio):
        self.pa_object.terminate()

```

Bilag 11 – Signaling server (SØF)

11.1 Endelig signaling server kode

```
from flask import Flask, request
from flask_socketio import SocketIO, emit
import uuid
import logging

logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')

app = Flask(__name__)

socketio = SocketIO(app, cors_allowed_origins='*', logger=True)

# Dict til at håndtere forbundne peers internt i serveren
peers = {}

# Liste over tilgængelige peers, som info til peers
available_peers = []

sdp_exchange_done = []

# I et reelt produktions-scenarie ville denne signaling server højest sandsynligt benytte sig af rooms,
# som de forskellige peers kunne opdeles i, således at der ikke skulle en signaling server til for hver
# p2p-forbindelse. I dette tilfælde gør vi ikke brug af rooms, idet vi kun har to peers.

@app.route('/')
def index():
    return "Server is running!"

# Ved etableret forbindelse tildelles en universalt unikt identifikator, UUID til den nyligt forbundne peer
# Dette UUID bruges til at identificere klienter under kommunikation
@socketio.on('connect')
def on_connect():
    # genererer UUID til peer
```

```

peer_id = str(uuid.uuid4())
# sæt UUID som key, og klientens session id som value
peers[peer_id] = request.sid
print(f"Peer '{peer_id}' : {peers[peer_id]} connected!")
print("Peers: ", peers)
# Oplys klienten om dens nye UUID
socketio.emit("peer_id", {"peer_id": peer_id})
print("Peer_id sent!")

# Resource håndtering, fjern frakoblede klienter fra listen over peers
@socketio.on('disconnect')
def on_disconnect():
    # variabel til midlertidig opbevaring af frakoblende peer
    disconnected_peer = None
    # led efter frakoblende peer i peers{}
    for peer_id, sid in peers.items():
        if sid == request.sid:
            disconnected_peer = peer_id
            break
    # fjern frakoblende peer fra peers{}
    if disconnected_peer:
        del peers[disconnected_peer]

@socketio.on('get_peers')
def get_peers():
    # Send liste over alle forbundne peers - undtaget den, der spørger efter
    peers (current_sid)
    current_sid = request.sid
    # iterer over peer_id, sid i peers{}
    for peer_id, sid in peers.items():
        # hvis de fundne peer_id, sid ikke er forespørgeren (current_sid)
        appendes de til available_peers[]
        if sid != current_sid:
            available_peers.append(peer_id)
    # send liste over tilgængelige peers til forespørgeren med event-tag
    'peer_list'
    emit('peer_list', {'peers': available_peers}, to=current_sid)
    print(f"""Sent list of available peers to {current_sid}.
- Available peers: {available_peers}""")

```

```

# læs offer pakke destination og afsender, tjek om destinationen er
# forbundet, og videresend til anden peer
@socketio.on('offer')
def handle_offer(offer):
    target_peer = offer["target"]
    sender_peer = offer["sender"]
    if target_peer in peers:
        socketio.emit("offer", offer, to=peers[target_peer])
        print(f"Forwarded offer from '{sender_peer}' to '{target_peer}'")
    else:
        print(f"Offer handling - Target peer '{target_peer}' not found")

# Læs answer pakke destination og afsender, tjek om destinationen er
# forbundet, og videresend til anden peer
@socketio.on('answer')
def handle_answer(answer):
    target_peer = answer["target"]
    sender_peer = answer["sender"]
    if target_peer in peers:
        socketio.emit("answer", answer, to=peers[target_peer])
        print(f"Forwarded answer from '{sender_peer}' to '{target_peer}'")
    else:
        print(f"Answer handling - Target peer '{target_peer}' not found")
        socketio.emit("error", {"message": f"Peer {target_peer} not found."}, to=answer["sender"])

@socketio.on('sdp_done')
def sdp_done(peer_id):
    sdp_exchange_done.append(peer_id)
    print(len(peers))
    print(len(sdp_exchange_done))
    if len(peers) == len(sdp_exchange_done):
        print(len(peers))
        print(len(sdp_exchange_done))
        socketio.emit('all_done')
        # socketio.stop()

# læs destination på ICE kandidat-pakker, tjek om destinationen er forbundet,
# og videresend til anden peer
# @socketio.on('candidate')

```

```
# def handle_candidate(data):
#     target_peer = data['target']
#     if target_peer in peers:
#         socketio.emit('candidate', data, to=peers[target_peer])
#         print(f"Forwarded ICE candidate from '{data['sender']}' to
# '{target_peer}'")
#     else:
#         print(f"Candidate handling - Target peer '{target_peer}' not
# found")

socketio.run(app, host="192.168.0.8", port=5000)
```

Bilag 12 - WebRTC klient (SØF)

12.1 Kode til test af WebRTC klient

Kan ses i bedre opklaring på følgende link: <https://imgur.com/a/80YpA1y>

Recv() 0 hz kode

```
class TestTrack(MediaStreamTrack):
    def __init__(self, audio_input):
        super().__init__()
        self.kind = "audio"
        self.input_track = audio_input
        self.sample_rate = 48000
        self.samples = 960
        self.timestamp = 0
        self.input_track.create_input_stream()

1 usage (1 dynamic)
async def recv(self):
    try:
        data = self.input_track.record_live()
        print(f"Captured audio frame size: {len(data)} bytes")
        if not data:
            print("No data available, skipping frame")
            return
        frame_in = AudioFrame(format="s16", layout="mono", samples=self.samples)

        print(f"Audio frame created: {frame_in} with {self.samples} samples")
        frame_in.sample_rate = 48000
        print(f"Sample rate: {frame_in.sample_rate}")
        # Audio data organiseres i planes - for mono audio er der kun én plane
        # [0] indikerer, at den første (og eneste) plane i 'frame' tilgås.
        # 'update(data)' lægger dataene ind i plane
        frame_in.planes[0].update(data)

        # frame.pts, pts står for 'presentation timestamp', som er det tidspunkt hvor
        # en frame skal afspilles, og repræsenteres i en eller anden form for tidsmåling.
        # Her er frame.pts repræsenteret gennem hvor mange frames, der er læs og behandlet.

        frame_in.pts = self.timestamp
        frame_in.time_base = Fraction(1, self.sample_rate) # Timebase er 1/sample_rate (1/48000)
        self.timestamp += self.samples # Forøg timestamp med antallet af samples i en frame
        # print(f"Read {len(data)} bytes of audio data")
        return frame_in
    
```

12.2 Endelig WebRTC klient kode

```
import socketio
import logging
import aiortc
from aiortc import MediaStreamTrack
import asyncio
from fractions import Fraction
from av import AudioFrame
import audio_handling as audio
```

```

# Enable logging til debugging
logging.basicConfig(level=logging.DEBUG)

# Initialize the Socket.IO client

# Connect to the signaling server

URL = "http://192.168.0.8:5000"

# OOP er benyttet for at gøre håndtering af bl.a. peer IDs nemmere - ved brug
af OOP opbevares
# informationer om hver peer direkte i deres egen instans, i stedet for at
skulle oprette
# globale variabler

class PeerClient:
    def __init__(self, server_url):
        self.io = socketio.AsyncClient(logger=True, engineio_logger=True)
        self.peer_id = None
        self.target_peer_id = None
        self.peer_connection = aiortc.RTCPeerConnection()
        # Når sdp_state er lig med 2 er SDP udvekslingen ovre, og hver peer
        kan gå i gang med
        # at samle ICE kandidater
        self.sdp_state = 0

        # test mediespor
        self.test_track = object
        self.audio_output = audio.Output()
        self.audio_output.create_output_stream()

        # event håndtering
        self.io.on('connect', self.on_connect)
        self.io.on('disconnect', self.on_disconnect)
        self.io.on('peer_id', self.set_peer_id)
        self.io.on('peer_list', self.handle_peer_list)
        self.io.on('offer', self.on_offer)
        self.io.on('answer', self.on_answer)

        # self.peer_connection.on("icegatheringstatechange",
        self.on_ice_candidate)
        # self.peer_connection.on("track", self.play_track)

```

```

async def connect(self):
    await self.sio.connect(URL, transports="websocket")
    await self.sio.emit('get_peers')

async def on_connect(self):
    print("Connected to signaling server!")

async def on_disconnect(self):
    print("Disconnected from signaling server!")

async def set_peer_id(self, data):
    if not self.peer_id:
        self.peer_id = data["peer_id"]
        print("Assigned peer id: ", self.peer_id)
    else:
        self.target_peer_id = data["peer_id"]
        print("Assigned target peer id: ", self.target_peer_id)

async def handle_peer_list(self, data):
    # sæt target_peer_id hvis der er tilgængelige peers
    print(f"Available peers: {data['peers']}")
    if data['peers']:
        self.target_peer_id = data['peers'][0]
        if self.target_peer_id is not None:
            if self.sdp_state == 0:
                await self.send_offer()

async def send_offer(self):
    offer_sdp = await self.create_offer()
    offer = {
        "sender": self.peer_id,
        "target": self.target_peer_id,
        "offer": offer_sdp
    }
    await self.sio.emit('offer', offer)
    self.sdp_state += 1

```

```

# opret offerpakke og sæt lokalbeskrivelsen til at være denne
async def create_offer(self):
    offer = await self.peer_connection.createOffer()
    await self.peer_connection.setLocalDescription(offer)
    return self.peer_connection.localDescription.sdp

async def on_offer(self, offer_sdp):
    offer_sdp = offer_sdp["offer"]
    await
self.peer_connection.setRemoteDescription(aiortc.RTCSessionDescription(offer_
sdp, "offer"))
    self.sdp_state += 1
    print("Offer received, set as remote description")
    answer_sdp = await self.create_answer()
    answer = {
        "sender": self.peer_id,
        "target": self.target_peer_id,
        "answer": answer_sdp
    }
    await self.sio.emit('answer', answer)
    print("Sent answer to ", self.target_peer_id)
    self.sdp_state += 1
    if self.sdp_state == 2:
        await self.sdp_done()

async def create_answer(self):
    answer = await self.peer_connection.createAnswer()
    await self.peer_connection.setLocalDescription(answer)
    print("Answer created, set as local description")
    return self.peer_connection.localDescription.sdp

async def on_answer(self, answer_sdp):
    answer_sdp = answer_sdp["answer"]
    await
self.peer_connection.setRemoteDescription(aiortc.RTCSessionDescription(answer_
sdp, "answer"))
    self.sdp_state += 1
    if self.sdp_state == 2:
        await self.sdp_done()

async def sdp_done(self):

```

```

print("SDP exchange completed")
await self.sio.emit('sdp_done', self.peer_id)

async def add_audio(self):
    audio_input = audio.Input()
    self.test_track = TestTrack(audio_input)
    self.peer_connection.addTrack(self.test_track)

# async def play_track(self, track):
#     print("Received track: ", track.kind)
#     if track.kind == "audio":
#         try:
#             frame = await track.recv()
#             audio_data = frame.to_ndarray()
#             print(f"Audio frame data received: {audio_data[:50]}...")
#             audio_bytes = audio_data.tobytes()
#             print(f"Audio frame data size: {len(audio_bytes)}")
#             self.audio_output.o_stream.write(audio_bytes)
#         except Exception as e:
#             print("Error :(", e)

async def wait(self):
    # vent på events
    await self.sio.wait()

async def terminate(self):
    print("Terminating resources...")

    if isinstance(self.test_track, TestTrack):
        self.test_track.close()

    # Stop og luk audio stream
    if self.audio_output and self.audio_output.o_stream.is_active():
        self.audio_output.terminate()
        print("Audio output stream closed.")

    # Luk peer forbindelse
    await self.peer_connection.close()
    print("Peer connection closed.")

    # Disconnect fra signaling serveren

```

```

    await self.sio.disconnect()

    print("All resources terminated.")

# async def disconnect(self):
#     await self.sio.disconnect()

class TestTrack(MediaStreamTrack):
    def __init__(self, audio_input):
        super().__init__()
        self.kind = "audio"
        self.input_track = audio_input

        self.samples = 960
        self.timestamp = 0

        self.input_track.create_input_stream()

    async def recv(self):
        return await asyncio.to_thread(self._recv_internal)

    # I et forsøg på at bedre håndtere RPI ressourcer er recv() lagt over i
    # en anden tråd
    # Dette hjalp ikke, men er inkluderet i den endelige version, da det var
    # her jeg stoppede
    def _recv_internal(self):
        try:
            data = self.input_track.record_live()
            # print(data)
            print(f"Captured audio frame size: {len(data)} bytes")
            if not data:
                print("No data available, skipping frame")
                return
            frame_in = AudioFrame(format="s16", layout="mono",
samples=len(data) // 2)
            frame_in.sample_rate = 48000

            print(f"Audio frame created: {frame_in} with {self.samples} samples")

            print(f"Sample rate: {frame_in.sample_rate} hz")

```

```

        frame_in.planes[0].update(data)

        # frame.pts, pts står for 'presentation timestamp', som er det
tidspunkt hvor
        # en frame skal afspilles, og repræsenteres i en eller anden form
for tidsmåling.
        # Her er frame.pts repræsenteret gennem hvor mange frames, der er
læs og behandlet.

        frame_in.pts = self.timestamp
        print(f"pts set: {frame_in.pts}")
        frame_in.time_base = Fraction(1, frame_in.sample_rate)  #
Timebase er 1/sample_rate (1/48000)
        print(f"Timebase: {frame_in.time_base}")
        self.timestamp += self.samples  # Forøg timestamp med antallet af
samples i en frame
        # print(f"Read {len(data)} bytes of audio data")
        return frame_in
    except IOError as e:
        print("recv error: ", e)
        return

def close(self):
    if self.input_track.i_stream:
        self.input_track.close()

async def main():
    client = PeerClient(URL)
    try:

        print("Created instance of Peer Client")

        print("Connecting...")
        await client.connect()

        await client.add_audio()

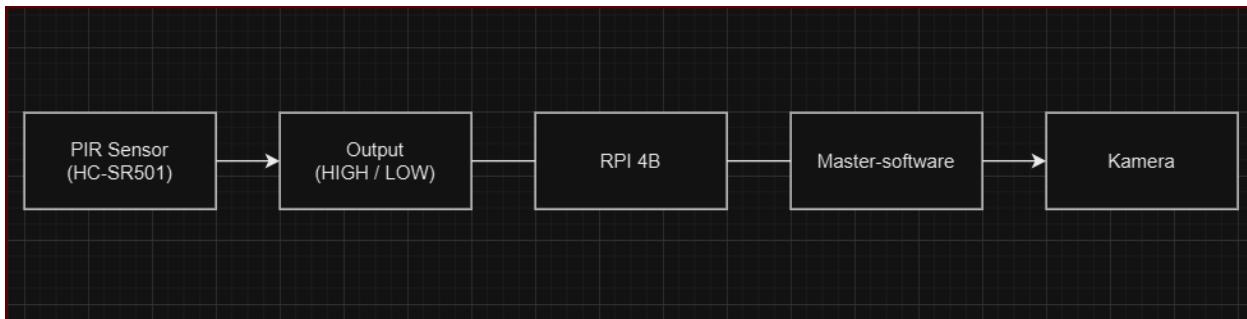
        await client.wait()

    except KeyboardInterrupt:
        print("Quitting")

```

```
    finally:  
        await client.terminate()  
  
asyncio.run(main())
```

Bilag 13 – Blokdiagram over PIR Sensorens sammenhæng i systemet. (HJL)



Bilag 14 – Sensor Videoer.

[Youtube Video af succesfuld test, PIR Sensor.](#)

11.1 Litteraturliste:

Indledning:

Artikel 2014 - <https://www.tv2nord.dk/nordjylland/aeldre-utrygge-efter-trickyverier>

Audio_handling:

PyAudio - <https://people.csail.mit.edu/hubert/pyaudio/docs/>

Python Wave bibliotek - <https://docs.python.org/3/library/wave.html>

WebRTC og signaling:

<https://webrtc.org/>

<https://www.nabto.com/what-is-a-webrtc-signaling-server/>

<https://aiortc.readthedocs.io/en/latest/>

Ansigtsgenkendelse:

SingleBoardBytes open CV instruktioner: <https://singleboardbytes.com/647/install-opencv-raspberry-pi-4.htm>

AI træning og Objekt detection - [Object detection](#)

Face detection using OpenCV using a Raspberry Pi Zero (2021):

https://www.youtube.com/watch?v=_amnSorlPaM

Install OpenCV and make Virtual env = <https://www.youtube.com/watch?v=QzVYnG-WaM4>

ffmpeg install: <https://www.youtube.com/watch?v=JR36oH35Fgg>

OpenCV course: <https://www.youtube.com/watch?v=oXlwWbU8l2o&t=11285s>

RP AI Kamera - <https://www.youtube.com/watch?v=U1lr6qV2o20&t=2494s>

PRI Schematic - <https://datacapturecontrol.com/articles/io-devices/single-board-computers/raspberry-pi-4b>

RPI AI camera - [RPI AI kamera dokumentation](#)

Git hub KM - https://github.com/kevinnmcaleer/object_detection

