Opsamling og Visualisering af Data om Luftkvalitet

Gruppe 2
Asbjørn R. Biering, Lucas Ta Jensen, Fabian Loki, Andreas L. Jonstrup,
Albert Gerlach Sylvest, Sebastian Lindau-Skands og Abdulrahman Kourbelal

AALBORG UNIVERSITY
STUDENT REPORT

Aalborg University

Title:
Opsamling og Visualisering af Data om Luftkvalitet

Theme:
Bæredygtighed

Project Period:
16.Oktober - 20 December

Project Group:
Gruppe 2

Participants:
Asbjørn R. Biering
Lucas Ta Jensen
Fabian Loki
Andreas L. Jonstrup
Albert Gerlach Sylvest
Sebastian Lindau-Skands
Abdulrahman Kourbelal

Supervisor:
Lene Tolstrup Sørensen

Date:
19-12-2024

Copies: 1

Abstract:
Denne rapport gennemgår i forbindelse med bæredygtighed, hvordan der kan måles og opstilles data om luftkvalitet i København, og hvordan det kan gøres let forståelig for befolkningen. Baggrunden for opgaven har taget udgangspunkt i rapporter fra organisationer som WHO og IPCC, som beskriver hvilke konsekvenser dårlig luftkvalitet har for verden og mennesker. Denne rapport undersøger desuden, hvordan der med forholdsvis simple værktøjer kan gøres en forskel, ved at give befolkningen mulighed for at have indsigt i luftkvaliteten på specifikke lokationer i København. Rapporten har gjort brug af en Arduino og et CLI-program, som giver mulighed for, henholdsvis at måle mængden af skadelige luftpartikler, og dernæst give det som input til CLI-programmet. CLI-programmet kan lave en grafisk repræsentation af luftkvaliteten på et givent tidspunkt og lokation, samt informere hvis niveauet af skadelige luftpartikler overstiger WHO's grænseværdier.

Indholdsfortegnelse

1 Indledning

Luftforurening er et problem i mange byområder over hele verden, og kan have betydelige sundhedskonsekvenser for deres befolkning. At kunne overvåge luftkvaliteten i forskellige områder er derfor afgørende både for at informere befolkningen om områder med høj luftforurening, og også for eventuelt at kunne bekæmpe problemet.

Derfor har denne rapport til formål at undersøge, hvordan der er udviklet en applikation til at opsamle, og fremvise data om skadelige partikler i luften. Hele projektet vises frem i rapporten fra problemanalyse til implementering og afsluttes med en diskussion og konklusion. I problemanalysen analyseres WHO's retningslinjer, for at skabe et overblik over problemet. I state of art undersøges hvad andre har gjort for at løse problemet, og om der er nogle der har lavet nogen lignende løsninger. I designprocessen gennemgås, hvordan applikation skal se ud, og hvilket udstyr der skal bruges til at måle partiklerne. Derefter gennemgås implementeringen, og til sidst en diskussion af projektet og projekt forløbet som helhed.

1.1 Problemformulering

Hvordan kan man med en Arduino indsamle data om $PM_{2.5}$, PM_{10}, NO_2 *og temperatur, og hvordan kan man håndtere og præsentere denne data for brugere i en brugervenlig terminalbaseret applikation?*

2 Problemanalyse

Problemanalysen ligger grundlaget for rapporten og dens relevans, set i lyset af omfanget af forureningen som også afdækkes. Der gennemgås en række forurenende stoffer, herunder hvordan de påvirker forskellige grupper mennesker, samt internationale retningslinjer for grænseværdier af disse stoffer. Ydermere undersøges det hvordan luftforurening forekommer.

2.1 WHO global air quality guidelines

WHO's air quality guidelines, AQG, er lavet for at informere mennesker over hele verden om luftforurening. Det gælder både individer, politikere og regeringer. Retningslinjerne, AQG, som er grundlagt af WHO er blot anbefalede værdier. Der er i øjeblikket kun retningslinjer for $PM_{2.5}$, PM_{10}, ozon (O_3), nitrogendioxid (NO_2), svovldioxid (SO_2) og carbonmonoxid (CO). PM_{10} står for alle partikler mindre end 10 μm, tilsvarende gælder det for $PM_{2.5}$. Disse er ikke de eneste luftforurenende stoffer, men dem som er blevet fokuseret på, i rapporten udgivet af WHO (World Health Organization, 2021). Eksperter ytrer, at der er bevis for at disse stoffer er skadelige for mennesker, og ved højere eksponering af disse stoffer, stiger risikoen og alvorlighedsgraden af påvirkningen på mennesket. (World Health Organization, 2021, kap. 2.3).

2.1.1 Udledningsprocessor

I forbindelse med udviklingen af, AQG's, blev der undersøgt hvilke processor udledte de forskellige stoffer. Særligt blev det fundet ud af, at den proces som udleder flest forurenende stoffer er forbrænding af fossile brændstoffer i forbindelse med produktionen af energi (World Health Organization, 2021, kap. 1.3.2). Både produktion af primære- og sekundære forureningsstoffer bliver udledt ved en sådan proces. Primære forureningsstoffer er stoffer udledt direkte af processen, fx black carbon som en del af PM2.5. Sekundære forureningsstoffer bliver dannet i atmosfæren ved kombination med andre primære forureningsstoffer.

2.1.2 Grunden til retningslinjerne

Til baggrund for retningslinjerne ligger en række årsager. Den primære årsag er at luftforurening er den største miljøtrussel for menneskers helbred. WHO estimerede i 2018 at 7 millioner dødsfald kunne ledes tilbage til luftforurening (World Health Organization, 2021, kap. 1.3.3).

Luftforureningens konsekvenser for menneskers helbred kan resulterer i omkostninger i forbindelse med hospitalsregninger og medicin, og koster på den måde udover menneskeliv også mange penge. Verdensbanken skriver som følger:

> This publication estimates that the global cost of health damages associated with exposure to air pollution is $8.1 trillion, equivalent to 6.1 percent of global GDP. (Bank, 2021)

— World Bank, 2019

Disse store sundhedsmæssige og økonomiske konsekvenser understreger, hvor vigtigt det er at overvåge luftkvaliteten i byer.

Nedenstående tabeller viser WHO's AQG af forskellige luftforenende stoffer.

Pollutant	Averaging time	Interim target				AQG level
		1	2	3	4	
$PM_{2.5}$, µg/m³	Annual	35	25	15	10	5
	24-hour[a]	75	50	37.5	25	15
PM_{10}, µg/m³	Annual	70	50	30	20	15
	24-hour[a]	150	100	75	50	45
O_3, µg/m³	Peak season[b]	100	70	–	–	60
	8-hour[a]	160	120	–	–	100
NO_2, µg/m³	Annual	40	30	20	–	10
	24-hour[a]	120	50	–	–	25
SO_2, µg/m³	24-hour[a]	125	50	–	–	40
CO, mg/m³	24-hour[a]	7	–	–	–	4

[a] 99th percentile (i.e. 3–4 exceedance days per year).
[b] Average of daily maximum 8-hour mean O_3 concentration in the six consecutive months with the highest six-month running-average O_3 concentration.

Figur 1: Tabellen viser hvad AQG niveauet er for $PM_{2,5}$, PM_{10}, O_3, NO_2 og SO_2 i $\frac{\mu g}{m^3}$ og CO i $\frac{mg}{m^3}$. (World Health Organization, 2021)

Pollutant	Averaging time	Air quality guideline that remain valid
NO_2, µg/m³	1-hour	200
SO_2, µg/m³	10-minute	500
CO, mg/m³	8-hour	10
	1-hour	35
	15-minute	100

Figur 2: Tabellen viser hvad AQG niveauet er for NO_2 og SO_2 i $\frac{\mu g}{m^3}$ og CO i $\frac{mg}{m^3}$. (World Health Organization, 2021).

2.2 Unges påvirkning af luftforurening

Alle mennesker påvirkes af luftforurening, men noget der ofte bliver overset er den forstærkede destruktive effekt som luftforurening har på unge og børn under udvikling, især fostre, som endnu ikke er blevet født. (European Environment Agency, 2023)

2.2.1 Hvorfor påvirkes unge i højere grad?

Unge, børn og fostre er mere følsomme overfor luftforurening end man kunne forvente. Der er mange årsager til dette, men i bund og grund er det primært det faktum at deres organer, immunsystem og krop i helheden, er under udvikling. Ved at blive udsat for luftforurening, forøges risikoen for sygdomme senere i livet blandt unge. (European Environment Agency, 2023)

Over 1200 dødsfald blandt folk under 18 år i EEA støttende, og medlems lande, anslås at være have været beskyldt luftforurening. (European Environment Agency, 2023)

Børn har højere åndedrætsfrekvens, samt de indånder mere luft ift. deres kropsvægt. Desuden er deres koncentration af luftforurening højere, da de har en lavere højde, og dermed trækker vejret tættere på jorden, hvor luftforureningsstoffer, især fra trafikudstødning udledning, bliver koncentreret. Da børn er mere fysisk aktive, ville de også kræve mere luft. Børn trækker derudover også vejret oralt, som resulterer i at luftforurening gennemtrænger dybt i deres nedre respirationstragt. Børns organer er desuden under udvikling, og deres immunsystem er svagere end den voksnes, som forstærker luftforureningens effekt.

Det kan altså ses at der er punkter, der pointerer at børn i langt højere grad har faktorer som påvirker at de optager mere luftforurenende stoffer pr. kilogram kropsvægt, de befinder sig steder med højere koncentrationer af luftforurenende stoffer, vejrtrækningsmetode, samt at de er under udvikling. Disse faktorer udmunder altså i at de er mere modtagelige overfor luftforurenende stoffer, og at de modtager væsentlig mere end den gennemsnittelige voksen. Grunden til det forholder sig til at være et problem beskrives næste afsnit. (European Environment Agency, 2023)

2.2.2 Hvilke effekter bidrager luftforurening til i børn og unge?

- Lav fødselsvægt, babyer af mindre størrelse (SGA), førtids fødsel, astma, reduceret lunge funktion, respirations infektioner, allergier i børn og unge, højere sandsynlighed for voksen kroniske sygdomme, generelt højere sandsynlighed for at udvikle forskellige sundhedsproblemer senere i livet. (Partikelforurening er knyttet til forøget sandsynlighed for spontan abort), Pneumoni, Mellemørebetændelse, Short term exposure -> allergi, eczema, snottet næse, øje kløen.
- Især O_3, NO_2, $PM_{2.5}$ påvirker på lang sigt lungefunktionen og lungeudvikling. Reduceret hjerneudvikling
- Stigende beviser for at luftforurening bidrager til kognitiv svækkelse (cognitive impairment), og spiller måske en rolle i udvikling af nogle typer fra autisme disorder spektrumet.
- Korrelation mellem benzen i trafik-relateret luftforurening og leukemi.

(European Environment Agency, 2023)

2.3 Særligt udsatte og ulighed

Ringe luftkvalitet påvirker ikke alle lige hårdt. Det er også en årsag til ulighed blandt mennesker på global skala. Siden 1990 er der set en reduktion af forurenede stoffer i byer og lande, som har taget initiativ til at forbedre luftkvaliteten. Disse områder tilhører i høj grad højindkomstlande. Samtidig med det er der også sket en forværring af luftkvalitet i lav- og middelindkomstlande. Dette skyldes hovedsageligt at produktionen af varer i høj grad er flyttet til disse lande, og da denne produktion ofte udleder mange forurenende stoffer, betyder det også en stigning på forening. Denne ulighed i forhold til luftkvalitet forstærker derved allerede eksisterende sociale og økonomiske uligheder (World Health Organization, 2021, kap. 1.3.4).

3 Metode

I metodeafsnittet gennemgås hvilke metoder der under projektarbejdet er blevet anvendt, og fortælle om hvilke virkninger disse metoder har haft for projektarbejdet.

Problemformuleringen blev fundet fra en diskussion om udledning af skadelige stoffer ved industrielle processor og trafik, herunder blev der diskuteret, hvordan luftforurening påvirker mennesker. Yderligere, hvorfor det er et stigende problem i samfundet.

Informationssøgning

Af anvendte metoder, er der til at starte med blevet gjort brug af specifikke nøgleordssøgninger i en søgemaskine, hvor nøgleordene var:

- "Luftforurening"
- "Luftforurening i København"
- "Air pollution"
- "Air pollutants"
- "Air pollution effects"
- "Air pollution WHO"
- "Air pollution EU"

At der søges på engelsk har til formål at få mere internationale kilder, som der også blev fundet, hvorimod søgningen på dansk frembragte mere lokale kilder, som kan bruges til enten af bekræfte eller afkræfte, at der er luftforurening i Danmark. Derudover er der også blevet valgt at søge efter kilder, som forventedes nyttige på baggrund af almen viden. Dette gælder kilder fra EU og WHO, som udviser forskellige interesser, men begge har information om emnet. Kilderne blev vurderet kildekritisk, og indholdet blev skimmet for at se om det ville være kompatible med projektarbejdet. Det kildekritiske arbejde gik hovedsageligt ud på at finde afsenderen, se kildens indhold, og faktatjekke hos en tredjepart. Derefter blev en der taget en beslutning om hvorvidt kilden skulle anvendes.

Metoder Anvendt i Designprocessen

Efter, at problemformuleringen blev fundet, begyndtes så småt en designproces. Her ville der søges efter nogle krav, for at indsnævre en bred ide om at løse et problem, til et konkret produkt. En del af kravene var blevet afledt fra indholdet på de fundne kilder, og nogle fra problemformuleringen. Disse krav ville stilles til det endelige produkt for at afhjælpe, løse eller på anden vis bidrage til en løsning. Hertil blev der primært kigget på den nye problemformulering som reference:

Hvordan kan man med en Arduino indsamle data om $PM_{2.5}$, PM_{10}, NO_2 *og temperatur, og hvordan kan man håndtere og præsentere denne data for brugere i en brugervenlig terminalbaseret applikation?*

Derudover blev der også erklæret krav, som virkede som begrænsninger, der ville mødes i forhold til udviklingen, et eksempel på dette ville være at et cirkeldiagram ville være meget svært at udarbejde ved kørsel af et C-program, især mens der forsøges at skildres noget ud fra det.

Udarbejdning af produktet er udført ved at dele gruppen op i tre delgrupper, hvorved de tre delgrupper fokusere på forskellige hovedelementer af koden, user interface, graf og data. Der blev aftalt såkaldte code reviews hvor hver delgruppe præsenterede det udførte arbejde og besvarede de eventuelle spørgsmål de andre gruppemedlemmere havde. Disse møder var essentielle for gennemsigtighed og forståelse af koden i gruppen. Undervejs i processen blev prototyper skabt, som blev evalueret på deres mangler og det de havde opnået, for at blive til det endelige produkt. Prototyperne har til formål at virke som milepæle, og hver gang opnå noget nyt. Til slut når hver delgruppe var færdig udførtes integrering af koden, så delene kunne spille sammen i en større sammenhæng og blive betegnet som et produkt.

Testning af produktet er udført både i fællesskab i gruppen, og med andre individer som ikke havde prøvet programmet før. I fællesskab blev den testet for:
- Om det kan hvad den skal, til den grad det var intenderet.
- Om der er fejl som skal rettes op på.
- Om brugerfladen er intuitiv nok, til hvis en ny bruger anvender koden.

Der kan læses mere om brugertesten i Afsnit 8.

3.0.1 Udviklingsværktøjer

Til selve udviklingen af applikationen er der også anvendt forskellige værktøjer og teknologier for at effektivisere denne proces. Dette er ikke direkte en del af det færdige produkt, men det er stadig relevant da det har haft en indflydelse på hvordan udviklen har fundet sted.

Text-editor og IDE

Der er blevet anvendt forskellige text-editorer/IDE'er til at skrive selve koden. I gruppen er der blevet anvendt Visual Studio Code, CLion og Zed, hvor det har været op til hvert gruppemedlem hvilken text-editor eller IDE de ville bruge. Der er desuden blevet brugt Arduino IDE til at skrive kode til Arduinoen.

Git og Github

Blandt andet er der anvendt Git til versionskontrol og Github til at lagre de forskellige Git-repositories. Git er et versionskontrolsystem som gør det muligt at spore ændringer i koden, vende tilbage til tidligere versioner og arbejde på forskellige dele af projektet parallelt. Github er en cloud-baseret platform hvor der kan lagres Git-repositories.

Ved at anvende Git og Github har det gjort os i stand til at lagre alt koden til applikationen et centralt sted, nemlig i et Github-repository. Dette har gjort det nemt for alle gruppemedlemmer at få adgang til den seneste version af koden

uanset hvor de befinder sig. Hvert gruppemedlem har kunnet downloade (pull) koden til deres lokale maskine, implementere den del de arbejder på, og derefter uploade (push) de ændrede filer tilbage til Git.

Git håndterer også automatisk sammenfletning af ændringer der er lavet af forskellige personer. Hvis der opstår konflikter, fx hvis to personer har ændret den samme linje i en fil, giver Git mulighed for manuelt at løse denne konflikt. Alt dette har samlet set har gjort det muligt for os at arbejde meget mere effektivt på den samme kodebase.

4 Baggrund
I dette kapitel gennemgås de teknologier og værktøjer, som der er blevet anvendt til at løse problemformuleringen.

4.1 Arduino
Arduino er en væsentlig del af projektet, da den bruges til at opsamle data fra sensorer, der kan bruges til appen. Arduino er en general-purpose microcontroller, og Arduino IDE'en gør det nemt at programmere og opstille elektriske kredsløb. Sammen med et breadboard kan der opstilles og testes sensorer nemt og hurtigt. Et breadboard er et bræt med huller, hvor der kan tilsluttes pins. På et breadboard løber der to hovedskinner på hver side til henholdsvis voltage og ground. Hver skinne har huller, som er forbundet, så der kan laves parallelle kredsløb. Der løber desuden mindre skinner på tværs af breadboardet, som også er forbundet, og som gør det nemmere at lave prototyper på elektriske systemer, uden at ledningerne skal loddes i manuelt.

4.1.1 Hvad er en microcontroller?
En microcontroller kan opfattes som en mini-computer, der kan bruges til at kontrollere sensorer og aktuatorer. En microcontroller har ikke noget operativ system, så det kræver en anden computer til at programmere den. En microcontroller har tre hovedkomponenter: CPU, hukommelse og IO-udstyr (Asadi, 2023). CPU'en er hjernen bag systemet, der holder styr på instruktionerne i programmet, og foretager alle beregningerne, og så modtager og sender den signaler til alle de andre komponenter i systemet.

En microcontroller har ikke særlig meget hukommelse, men den skal have noget hukommelse til at opbevare programmet, og potentiel data fra sensore, som den skal holde styr på.

Til sidst har en microcontroller noget IO-udstyr, i form af sensorer (f.eks. temperaturmåler, fugtighedsmåler, lysmåler) og aktuatorer (f.eks. elektrisk motor, LED, buzzer). Sensorer sender signaler til microcontrolleren, og aktuatorer modtager signaler fra microcontrolleren.

4.1.2 Arduino UNO R3
Til projektet bruges en Arduino UNO R3, som har følgende specifikationer (Arduino, 2024):

- AVR CPU 16 MHz: Refererer til dens "clock speed" det er hvor mange ticks den laver i sekundet. 16 MHz betyder at CPU'en opererer med i gennemsnittet 16.000.000 ticks i sekundet.
- 32 kB flash memory: Flash memory er den hukommelse, som programmet "flasher" på. Der kan altså laves et program, som fylder op til 32 kB.
- 2 kB SRAM: SRAM bliver brugt under en process til at opbevare midlertidig hukommelse, som der hurtigt skal haves fat i.

Alle disse specifikationer er mere end rigeligt, for at udføre, det som den er intenderet at anvendes til under dette projekt. I Arduino UNO R3 er der desuden 32 GPIO's. GPIO står for General-purpose input/output (Butler, 2022), og det er de pins, som skal bruges til at tilslutte sensorerne og aktuatorerne. De vigtigste pins er:

- VCC: outputter konstant 3.3V eller 5.5V.
- GND: Ground
- Digital IO: Kan fungere som input og output at et signal som er enten 1 eller 0
- Analog IO: Også kendt som ADC (Analog-to-digital converter) og DAC (Digital-to-analog converter)

CPU'en fungerer kun med digitale signaler, så hvis der besiddes en sensor, som har et analogt input eller output, så skal disse pins bruges til at konvertere signalet. Arduino gør det ved at konvertere analoge signaler til et 10-bit tal, det vil sige at et analog signal bliver repræsenteret som et tal fra $2^0 - 1$ til $2^{10} - 1$ altså 0-1023.

4.2 Programmeringssproget C

Til at udvikle appen bruges C. C er et kompileret programmeringssprog. Det vil sige, at det compiler direkte til maskinsprog, som kan køres på den pågældende computer. C er tæt integreret med hardwaren, og tillader f.eks. programmøren at manipulere computerens hukommelse. Dette gør sproget optimalt til at lave lavniveau programmering af for eksempel operativsystemer og compilere.

C har desuden et stærkt <stdio> bibliotek (standard in/out), som har en række værktøjer, der kan hjælpe med at lave CLI-programmet (command-line interface). For eksempel har den funktioner som "printf", der tager flere argumenter, der benyttes til at printe tekst ud i terminalen på den måde det er intenderet at ville gøres. Den har også, funktioner til at læse input fra brugeren som "scanf" og "fgets". En anden ting C har af udbytte, er funktioner til at læse og skrive henholdsvis fra og til filer. Dette er vigtigt både, når der skal skrives og gemmes data fra Arduinoen over i en fil, og når der skal læses data fra filerne over i programmet.

5 State of art

I dette kapitel gennemgås lignende projekter og teknologier, og ser på hvordan det tidligere er forsøgt at løse problemformuleringen. Der undersøges særligt data indsamlings projekter fra Aarhus universitet og Københavns Kommune.

Valget af rapporterne fra Aarhus Universitet og Københavns Kommune er nøje truffet, da de begge belyser luftforureningens mange facetter og betydningen af bæredygtige løsninger.

Aarhus Universitets rapport leverer en omfattende analyse af luftkvaliteten på nationalt niveau ved at anvende avancerede metoder som målestationer og modelberegninger. Rapporten giver dybdegående indsigt i forureningskilder som fine partikler (PM_{10} og $PM_{2.5}$), nitrogendioxid (NO_2) og svovldioxid (SO_2), som ikke kun stammer fra trafik, men også fra industrielle processer og energiproduktion. Denne tilgang sikrer en bred forståelse af, hvordan forskellige former for luftforurening påvirker miljøet og folkesundheden. Afsnit 5.1 Københavns Kommunes rapport fokuserer på en lokal kontekst og fremhæver de mange kilder til luftforurening i bymiljøer. Ud over trafik og brændeovne belyser rapporten også forurening fra bygge- og industrisektorer samt andre aktiviteter, der bidrager til emissioner af skadelige stoffer som kulilte (CO) og ozon (O_3). Rapporten analyserer forureningens rumlige fordeling og dens komplekse interaktioner med byens klima og beboernes sundhed. Ved at kombinere disse rapporters nationale og lokale perspektiver opnås en helhedsorienteret forståelse af luftforureningsproblematikken og dens mange kilder. Dette er afgørende for at kunne udvikle effektive og målrettede bæredygtige løsninger. Afsnit 5.2

Sammen supplerer de to rapporter hinanden da de hver især fokuserer på forskellige dele af forurening i Danmark. Det er derfor en fordel at undersøge data fundet i begge reporter i stedet for blot en enkelt af dem.

5.1 Aarhus Universitet

5.1.1 Introduktion

Rapporten "Luftforurening, udledninger og effekter" fra Århus universitet præsenterer deres arbejde med at overvåge luftkvaliteten i Danmark. Formålet er at levere præcise data om forureningsniveauer og deres udvikling over tid med udgangspunkt i internationale standarder og grænseværdier. Overvågningen af luftforurening spiller en central rolle i vurderingen af forureningens indvirkning på miljø og folkesundhed og bidrager til nationale og internationale miljøpolitikker.

5.1.2 Formål og fokusområder

Aarhus Universitets har fokus på den aktuelle status for luftkvalitet baseret på gældende grænseværdier, og kigger samtidig på tendenser i udviklingen af

luftforurening med hensyntagen til internationale tiltag og ideelle værdier for menneskelig sundhed

5.1.3 Overvågningsmetoder

Luftkvalitetsdata indsamles ved hjælp af 18 stationer som er placeret strategisk over hele landet for at få en større udspredning og varians. Der bliver herefter foretaget modelberegninger på baggrund af den indsamlede data, ved anvendelse af avancerede luftkvalitetsmodeller, der dækker geografiske områder fra den nordlige halvkugle til specifikke adresser og naturområder.

Målestationerne er nøje placeret for at dække forskellige miljøer:
• Trafikerede gader med høj forurening.
• Tage og gårde i byområder for at repræsentere generel byluftkvalitet.
• Åbne, fjerntliggende områder for at måle baggrundsforurening.

Luftkvalitet analyseres gennem to hovedmetoder, en af disse er straksanalysen, hvorved der foretages luftprøver der straks analyseres på målstationen, og data herfra updloades direkte til en central database i Roskilde. Dens modstykke langtidsanalysen foregår ved at luft og partikler opsamles over længere tid og sendes til laboratoriet for detaljeret analyse.

Partikelforurening måles med fokus på:
• PM_{10}: Partikler med en diameter på under 10 µm.
• $PM_{2.5}$: Partikler med en diameter på under 2.5 µm.

Der er mange forskellige metoder til partikelmåling, og dem der bliver kigget på er:

1. Gravimetrisk referencemetode
   • Måler den daglige gennemsnitlige partikelmasse over 24 timer.
   • Metoden sammenligner vægten af et rent filter med vægten efter partikelfiltrering, divideret med luftvolumenet i prøvetagningsperioden.
2. Betaabsorption
   • Måler partikelmasse ved at anvende betaabsorptionsteknikker på filtrerede partikler.
   • Partikelvægt bestemmes straks ved indsamling, og ugentlige prøver sendes til DCE's laboratorier for yderligere analyse.
   • Samme teknik benyttes af brandalarmer
3. TEOM (Tapered Element Oscillating Microbalance)
   • Måler gennemsnitlige partikelniveauer hvert 30. minut.
   • Teknikken bruger vibrationer i et konisk element, som ændrer resonansfrekvens baseret på vægten af opsamlede partikler.
   • Begrænsninger: Høj driftstemperatur kan føre til et 20-30 % massetab på grund af fordampning.
   • Fordele: Høj opløsning og følsomhed i tidsbestemte målinger.

5.2 Københavns Kommune

Overvågning af luftkvalitet i Københavns Kommune årlig afrapportering for 2022

5.2.1 Introduktion

Denne årsrapport giver en omfattende oversigt over luftkvaliteten i København og understreger vigtigheden af at overvåge forurening og dens indvirkning på folkesundheden. (World Health Organization, 2021) Luftkvalitet er en af de mest presserende miljøudfordringer, der påvirker byens beboere, da forurenende stoffer bidrager til respiratoriske problemer og kroniske sygdomme. Rapporten behandler centrale faktorer som trafikforurening, påvirkningen af brænde og industriaktivitet, og fremhæver byens bestræbelser på at forbedre luftkvaliteten gennem datamonitorering og implementering af miljøløsninger.

5.2.2 Rapportens mål

Formålet med rapporten er at evaluere luftkvaliteten for 2022, identificere de forskellige forureningskilder og give præcise data om forureningens fordeling i byens forskellige kvarterer. Disse data skal hjælpe beslutningstagere i Københavns Kommune med at designe effektive miljøpolitikker og øge bevidstheden om forurening og dens konsekvenser.

5.2.3 Hovedemner:

Forureningsniveauer: Rapporten angiver, at kvælstofdioxid (NO_2) og fine partikler (PM_{10} og $PM_{2.5}$) er de primære forurenende stoffer. (*EU's luftkvalitetsdirektiv 2008/50/EF.*, 2008) Niveauerne af NO_2 overstiger WHO's anbefalede grænser i områder med høj trafik, hvilket øger risikoen for respiratoriske problemer.

Forureningskilder: Rapporten fokuserer på flere centrale kilder til forurening, herunder:

- Biludledninger: Biler er den største kilde til kvælstofdioxid, især i befolkningstunge områder.
- Brænding af træ: Brænding af træ er en hovedkilde til fine partikler, som har alvorlige indvirkninger på åndedrætssystemet. Brænding af træ sker blandt andet i forbindelse med produktion af trækul og energiproduktion.(Københavns Kommune, 2022)

Rumlig fordeling af forurening: Forurening er mest koncentreret i travle områder og hovedveje med høj trafik. (Københavns Kommune, 2022) Rapporten fremhæver de kvarterer, der har brug for hurtig indgriben for at forbedre luftkvaliteten.

5.2.4 Københavns indsats for bedre luftkvalitet

Københavns Kommune har iværksat flere initiativer for at reducere forurening og forbedre luftkvaliteten i byen. Blandt de tiltag, der er blevet truffet, er fremme af elektrisk transport, hvor der blandt andet er blevet støttet brugen af elbiler og

elektriske busser. Denne indsats skal bidrage til at reducere udledningen af skadelige stoffer fra fossile brændstoffer og fremme miljøvenlige transportmuligheder. Derudover er der gjort en indsats for at udvide grønne områder og plante træer i byen, hvilket hjælper med at absorbere forurening og samtidig forbedre luftkvaliteten. Kommunen har også implementeret strenge miljøpolitikker, herunder opdaterede lovgivninger, der sigter mod at reducere udledninger fra fabrikker, samt et forbud mod brænding af træ i visse områder.

For at opnå yderligere forbedringer og reducere forurening endnu mere, afsluttes rapporten med flere anbefalinger. En vigtig anbefaling er at øge miljøovervågningen i de mest forurenede områder for at få præcise og opdaterede data om luftkvaliteten. Dette vil kunne hjælpe med at målrette indsatsen mod de områder, hvor forureningen er størst. Derudover anbefales det at intensivere oplysningskampagner for at øge samfundsbevidstheden om vigtigheden af luftkvalitet og de handlinger, borgerne kan tage for at reducere forurening, såsom at vælge miljøvenlige transportmuligheder. Endelig bør der være øget støtte til miljøinnovation, hvor der arbejdes på at udvikle nye luftrensningsteknikker og bæredygtige metoder inden for både transport og industri. Disse tiltag vil sammen kunne skabe en mere bæredygtig fremtid for Københavns borgere og miljøet.

5.2.5 Avancerede værktøjer

For at nå frem til rapportens resultater benyttede forskerne avancerede værktøjer og teknikker til at overvåge luftkvaliteten, herunder:

- Luftovervågningsstationer: Der blev installeret faste overvågningsstationer i forskellige områder for at overvåge forureningsniveauerne i løbet af året, såsom kvælstofdioxid (NO_2) og fine partikler (PM_{10} og $PM_{2.5}$).
- Præcise sensorer: Bærbare sensorer blev brugt til at måle de øjeblikkelige ændringer i luftkvaliteten i specifikke områder.
- Computermodeller: Simuleringsmodeller, som f.eks HYSPLIT, blev anvendt til at analysere, hvordan forureningerne spreder sig og identificere hotspots med høj forurening.
- Klimadata: Dette inkluderer vejroplysninger som vind og temperatur for at vurdere, hvordan forureningens fordeling påvirkes af vejrfaktorer.

5.3 Delkonklusion

Gennem rapporterne er der opnået en dybere forståelse af de forskellige kilder til luftforurening og deres konsekvenser for både miljø og sundhed. Aarhus Universitets rapport understregede betydningen af avancerede overvågningsmetoder og dataanalyse til at identificere forureningsmønstre på tværs af forskellige kilder som energiproduktion, industri og transport.

Københavns Kommunes rapport gav et værdifuldt indblik i de specifikke udfordringer ved luftforurening i bymiljøer, herunder bidrag fra flere sektorer og deres indvirkning på lokalsamfundet.

Denne viden er afgørende for projektet, da der kan anvendes de metodologier og indsigter fra rapporterne til at designe løsninger, der imødekommer udfordringerne ved forskelligartede forureningskilder. Ved at forstå hvordan forurenende stoffer som $PM_{2.5}$, NO_2 og O_3 påvirker luftkvaliteten og sundheden, kan projektet skræddersys teknologier og anbefalinger, der bidrager til bæredygtighed og bedre livskvalitet for borgerne.

Rapporternes fokus på tværgående samarbejde mellem forskning, teknologi og politiske tiltag fremhæver også vigtigheden af en integreret tilgang til at løse luftforurenings problematikken.

6 Designproces

I dette kapitel gennemgås selve designprocessen, og hvilke tanker og overvejelser, der er blevet gjort undervejs.

6.1 Konceptet Bag Appen

Applikationen skal visualisere den data, som er blevet opsamlet via sensorer forbundet til en Arduino. Den indsamlede data består af en række målinger for henholdsvis PM_{10}, $PM_{2.5}$, NO_2 og temperatur, hvor gennemsnittet beregnes for hver time. Eftersom det er meget besværligt at finde den data der leder efter i dataens, rå form, er der brug for en mere brugervenlig løsning. Det er her programmet kommer ind i billedet. Ideen bag programmet er, at det skal være en terminal-baseret applikation som skal hjælpe brugeren med at få et overblik over den data, der er indsamlet om luftkvalitet. Appen skal have et user interface, som både er nemt at anvende, og som hurtigt kan fremvise brugeren, den information de har brug for. Programmet skal desuden gøre brugeren opmærksom på, hvornår der er en sundhedsskadelig mængde af luftforurening, baseret på WHO's grænseværdier.

Generelt er målet med appen at visualisere de store mængder af data, til noget mennesker kan bruge, og tage stilling til. Måden hvorpå dette skal opnås er, blandt andet ved opsætte data på en mere overskuelig måde, samt gøre det nemmere at vælge hvilken lokation og dato der vil ses data fra. Meningen er, at der skal være to forskellige måder at visualisere data på. Den første fremvisningsform er en simpel tabel, der viser niveauet af de forskellige målinger ud fra et bestemt tidspunkt. Den anden fremvisningsform er et søjlediagram, som viser niveauet af et bestemt målingsparameter over et antal dage. Hver søjle skal vise niveauet for en bestemt dag, samt vise, hvornår der er en sundhedsskadelig mængde af luftforurening vha. farver og ratings.

Udover bare at visualisere data, skal appen også anvendes til nemt at navigere mellem de forskellige visualiseringstyper, samt hvilke data der gerne vil ses. Ideen er, at terminalen anvendes sammen med en række simple kommandoer, til nemt at skifte mellem forskellige veje og tidspunkter, og dermed hurtigt kan finde frem til den korrekte fremvisningen af ønskede data. Det er intentionen, at appen skal fungere med forskellige skærme, som brugeren kan navigere imellem ved brug af bestemte kommandoer. Derved er det muligt at at tilgå de forskellige visualiseringstyper, samt andre eventuelle dele af programmet. Udover det vil der også være andre kommandoer, som kan bruges til at interagere med de forskellige skærme, for eksempel hvis der vil ændres på datoen eller lokationen. Der vil blive dykket mere i dybden af de forskellige funktioner og specifikke kommandoer i *Krav Til Appen* (Afsnit 6.1.3) og *Interface Design* (Afsnit 6.2).

6.1.1 Bruger og målgruppe

Applikationen er primært rettet mod borgere der bor eller arbejder i København, eller mere generelt personer, som bruger meget tid i byen. Udover det skal det

også være borgere, som er interesserede i at overvåge luftkvaliteten i deres nærområde. Målgruppen omfatter både personer der færdes meget udendørs i byen, og personer hvor dårlig luftkvalitet kan have en stor indflydelse på deres helbred. Dette er især sårbare grupper som børn, ældre og personer med åndedrætsproblemer. Da målgruppen er ordinære personer og ikke forskere eller lignende, er det vigtigt at applikationen derved er brugervenlig, hvilket der vil tages højde for i designet.

Som beskrevet i Afsnit 2.2.1 (Unges påvirkning af luft forening), er børn særligt sårbare over for luftforurening. I Afsnit 5 (State of Art), viser Københavns kommunes rapport desuden at der er en højere koncentration af forenende stoffer i visse bydele, hvilket kan have sundhedsmæssige konsekvenser for beboerne (Københavns Kommune, 2020). Derfor er det relevant at udvikle et værktøj der kan hjælpe de her borgere med at overvåge og forstå luftkvaliteten i deres nærområde.

Applikationen kan anvendes i en række forskellige scenarier. Nedenstående er eksempler på forskellige use-cases, hvor appen blandt andet kunne bruges:

- **Planlægning af udendørsaktiviteter:** En person, der planlægger en løbetur, gåtur eller lignende gennem København, kan bruge applikationen til at se, hvilke områder der har høj luftforurening, og planlægge en rute, der så vidt muligt ungår disse områder. Derved kan de minimere deres eksponering for luftforurenende partikler.
- **Valg af bopæl:** En familie der overvejer at flytte til København, kan bruge applikationen til at undersøge luftkvaliteten i forskellige bydele, og vælge et område med mindre luftforurening. Dette er især relevant, hvis familien har børn, da de er særligt følsomme over for luftforurening (European Environment Agency, 2023).
- **Vurdering af arbejdspladsplads placering:** En person der søger nyt job i København, kan anvende applikationen til at vurdere luftkvaliteten omkring potentielle arbejdspladser, og derved sikre, at de ikke udsættes for sundhedsskadelige niveauer af forurening i løbet af deres arbejdsdag.

Disse eksempler illustrerer, hvordan applikationen kan give dens brugere værdifuld information om luftkvalitet, der kan hjælpe dem med at træffe forskellige beslutninger, og beskytte deres helbred.

6.1.2 Bruger Scenarier

Afsnittet her handler om hvordan mere specifikke scenarier hvor applikationen kan bruges. Det tager udgangspunkt i personer i situationer hvor applikationen kan bruges til at løse et problem.

Live information om forurening på specifik gade

Hr. Hansen plejer at cykle ned ad Beckersvej, når han skal til og fra arbejde, men han har på det seneste oplevet, at luften på hans rute ikke føles helt normalt. Han

åbner derfor applikationen for at se en graf over luftkvaliteten på den vej i løbet af den sidste uge. Han ser, at den i øjeblikket er meget ringe. Han anvender derfor en kommando, for at se data for en anden vej, som han også kan bruge til at tage på arbejde. Han ser, at niveauet for alle forureningsstoffer ligger gevaldig under deres tilhørende grænseværdi på den nye vej. Han beslutter sig derfor for at tage den nye vej til arbejde i stedet for den gamle.

Overblik over generelt forurening i en tidsperiode
Familien Hansen skal købe et hus. Familien Hansen har små børn og ved, at luftforurening påvirker særligt børn og unge. De går derfor ind på applikationen, og finder frem til data der tilhører vejnavnet for det hus de overvejer at købe. De kigger rundt i forskellige datoer, og kan se, at i løbet af det seneste år har været et enkelt tilfælde med uhensigtsmæssig høj luftforurening. De vælger derfor at spørger ejendomsmægleren ind til den periode. Ejendomsmægleren fortæller, at dette kunne skyldes en ombygning i huset, som kan forklare det forhøjede luftforureningsniveau.

Overblik over et specifikt forurenings stof
Fru Hansen har læst om $PM_{2.5}$. Hun er meget nervøs overfor påvirkningen af netop $PM_{2.5}$. Hun åbner derfor applikationen, og navigerer til den vej hun bor på. Efter at have undersøgt forskellige data for forskellige datoer, ser hun, at niveauet gentagne gange i løbet af det seneste år, har været over det grænseværdien. Hun ser, at de grænseværdien er fastsat af WHO som hun synes er en valid kilde. Hun tager derfor kontakt til kommunen, og oplyser dem om problemet. Kommunen forsikrer hende, at de vil undersøge nærmere og komme op med en løsning.

6.1.3 Krav Til Appen
For det er muligt at designe selve appen, er det nødvendigt at opstille en række krav til appens funktionalitet. For kravene er der kategorier, hvori kravene er kategoriseret på baggrund af hvor nødvendige de er. I dette afsnit gennemgås de forskellige krav mere overordnet, hvorefter det mere konkrete design af, hvordan kravene implementeres på vil gennemgås i afsnittet Interface Design (Afsnit 6.2). Kravene til applikationen er udarbejde på baggrund af State of Art, og det førnævnte brugsscenarier.

- **"Must-have" krav:** Disse krav *skal* være implementeret i appen for, at den fungerer på et fundamentalt niveau. Dette betyder, at brugeren skal være i stand til at åbne applikationen, og se data for bestemte lokationer på bestemte tidspunkter.
- **"Should-have" krav:** Disse krav vil forbedre appen med, for eksempel, udvidet eller forbedret funktionalitet. Dog er de ikke strengt taget "nødvendigt" for, at appen fungerer, men det vil stadig være en prioritet at implementere dem.
- **"Could-have" krav:** Disse krav vil forbedre appen, men er mindre vigtige end "should-have" kravene. Dette er enten funktioner, som ikke vil bidrage meget til

appen eller funktioner, som vil være meget tidskrævende at implementere, og som derfor burde prioriteres efter andre funktioner.
- **"Won't-have" krav:** Dette er krav som i teorien kunne implementeres, men som er valgt at blive ekskluderet. Dette kan for eksempel være fordi, de er for komplicerede at udvikle, eller måske vil tage for lang tid, og derfor er det blevet besluttet, at de er out-of-scope. Disse ting kunne dog tilføjes i en eventuel videreudviklet version af programmet.

Ud fra disse kategorier er de forskellige krav til applikationen blevet opdelt.

Must-have (Essentielt for fundemental funktionalitet)
Simpelt terminal-baseret interface Appen kræver et simpelt funktionelt terminal-baseret interface, hvor brugeren skal kunne skrive kommandoer, og derved interagere med appen. Dette er grundlaget for, at alle de andre funktioner i appen kan anvendes af brugeren, og er derfor den vigtigste del af appen.
Visning af data for bestemt tidspunkt Appen skal kunne vise de forskellige datapunkter for et bestemt tidspunkt og lokation til brugeren. Dette skal være i form af en simpel tabel. Dette er den mest simple måde at vise data på, og eftersom brugeren er nødt til at se data på en eller anden måde er dette et must-have krav.
Navigations-kommandoer For, at appen kan anvendes, skal der være implementeret kommandoer, til at navigere mellem de forskellige skærme. Disse kommandoer vil blive implementeret, som følge af de skærme de skal navigere hen til.
Lokations-kommando Denne kommando skal anvendes til at vælge, hvilken lokation der vil fremvises data for. Uden denne kommando vil brugeren ikke være i stand til at vælge en lokation, hvilket er en af de centrale formål med appen.
Dato-kommando Denne kommando anvendes til at vælge hvilket tidspunkt der vil fremvises data for. Den er vigtig af samme årsag som lokations-kommandoen, blot med tidspunkt i stedet.

Should-have (Vigtigt, men ikke nødvendigt for den første version)

Visualisering med søjlediagram

Giver mulighed for et visuelt overblik over data for en række dage. Selvom dette er en værdifuld funktion at have i appen, er det ikke som sådan nødvendigt.Brugeren kan allerede se alt den data, som de har lyst til ved brug af tabellen for specifikke tidspunkter. Det vil dog være den første prioritet, efter de nødvendige funktioner er implementeret.

Udvidet søjlediagram

Søjlediagrammet kan udvides, så det blandt andet viser, om målingerne er over det anbefalede niveau. Dette skal ske i forhold til WHO's grænseværdier.

Generel brugervenlighed

I sammenhæng med alle de forskellige funktioner er det også vigtigt, at appen overordnet er brugervenlig. Dette kan være ting som følgende:

- Hjælp-kommando: En kommando, som viser brugeren hvilke kommandoer de kan anvende.
- Tydelige error-meddelelser: Hvis der opstår en fejl, skal brugeren nemt kunne se hvorfor den er opstået
- Input-validering: Appen skal håndtere uventede bruger input på en effektiv måde

Could-have (Kan implementeres hvis der er tid)

Tilføj mulighed for personlig tilpasning af graf og UI

Programmet kunne gøre brugeren i stand til at tilpasse forskellige dele af det. Dette kunne være ting som:

- Tilpasning af grafens udseende, fx forskellige teksttegn til at tegne den, eller ændre grafens farver.
- Ændre på de forskellige toleranceværdier for grafen. Personer med astma kunne fx sætte tolerancen lavere, da de er mere følsomme overfor luftforurening end den gennemsnitlige person.

Flere diagramtyper

Andre diagramtyper kunne give brugeren en bedre forståelse af mængden af luftforurening, ved at give dem en alternativ måde at visualisere dataene på. Dette kunne fx være som et kurvediagram, eller i form af et boksplot.

Won't-have (Ting som er out of scope)
Udvikling til en app I stedet for en terminal-baseret app, kunne den i stedet udvikles som en web-app eller mobilapp med en grafisk brugerflade. Dette ville gøre den mere brugervenlig og nemmere tilgængelig for den bredere befolkning, da størstedelen er vant til at bruge denne type applikation og ikke terminal-baserede applikationer. Det vil også gøre det nemmere at implementere mere avancerede funktioner. Det vil dog kræve et at bygge applikationen op fra bunden igen, og er derfor out of scope for dette projekt.
Advarselsnotifikation En af de funktioner som kunne implementeres i en web-app/mobilapp er notifikationer. Her kunne brugeren blive kunne advaret, hvis luftforureningen overstiger et bestemt niveau.
Heatmap En grafisk applikation kunne også indeholde et heatmap til at se geografisk, hvor der er ringe luftkvalitet. Dette kræver dog også adgang til målingsdata fra en stor mængde af lokationer, da kortests information ellers ikke vil være særlig brugbart.
Live data I stedet for at læse dataene fra en fil med historisk data, kunne applikationens data automatisk blive opdateret sådan, at ny målt data automatisk tilføjes.

6.2 Interface Design

I dette afsnit beskrives designet af den terminalbaserede applikation ud fra de opstillede krav. Dette inkluderer applikationens struktur, brugergrænseflade og mockups af de forskellige skærme.

6.2.1 Applikation struktur

Som tidligere nævnt, skal applikationen overordnet fungere med forskellige skærme, som brugeren skal kunne navigere imellem. Disse skærme kan anvendes til at vise brugeren forskellige ting. Det forventes, at applikationen skal indeholde følgende skærme:

- **Velkomst-skærm:** Dette er den første skærm, som brugeren ser når de anvender applikationen. Dette skal være en meget simpel skærm, der måske indeholder noget grundlæggende information om appen, samt nogle af de forskellige kommandoer, som brugeren kan gøre brug af.

- **Hjælp-skærm**: Dette er en skærm, som indeholder alle de forskellige kommandoer samt en beskrivelse af hvad de gør. Dette kan brugeren anvende for at finde ud af, hvilke kommandoer de skal bruge for at udføre deres ønskede handling.
- **Data-skærm**: Meningen med denne skærm er at vise detaljerede data for en lokation for et specifikt tidspunkt. Det forventes, at det ønskede data skal opstilles i form af en tabel eller lignende.
- **Graf-skærm**: Denne skærm skal vise søjlediagram, som visualiserer niveauet af et målingsparameter over et antal dage.

Brugeren skal så være i stand til at skifte mellem de forskellige skærme ved at skrive bestemte kommandoer. Bagvedliggende skal applikationen fungere ved at lagre forskellig information som skal bruges af de forskellige skærme. For eksempel skal den huske, hvilket tidspunkt der skal vises data fra, hvilket så bruges af data-skærmen når den er aktiv. Denne information kan også modificeres ved brug af forskellige kommandoer.

6.2.2 Kommando interface

Som nævnt adskillige gange indtil videre, er det meningen at brugeren skal kunne interagere med applikationen ved brug af forskellige kommandoer. For at skabe et overblik over de forskellige kommander, er følgende tabel blevet lavet:

Kommando	Beskrivelse
`quit`, `q`	Afslutter programmet
`help`, `h`	Skifter til hjælp-skærmen
`reset`, `r`	Genstarter programmet til dens originale tilstand
`data`, `d`	Skifter til data-skærmen
`graph`, `g`	Skifter til graf-skærmen
`location`, `l`	Anvendes til at skifte til en anden lokation at se data for
`time`, `t`	Bruges til at ændre det tidspunkt som data vises for
`timespan`, `span`, `s`	Indstiller tidsperioden for dataen der ønskedes vist
`info`, `i`	Viser generel info om programmet, samt visse standard indstillinger

Tabel 1: Kommandoer i applikationen

Alle kommandoer skal kunne anvendes globalt. Det giver derfor ikke mening at producere et flowchart over the overordnede program, da det bare ville resultere i et rod af pile, som alle sammen peger på alle de forskellige elementer. Der vil dog i implementerings afsnittet fremstilles et flowchart over enkelte kommandoer, når de er blevet konkretiseret i det endelige program.

6.2.3 Applikation mockups

Inden, at udviklingen af de egentlige program begynder, er det en god ide at lave såkaldte mock-ups. Dette er en slags skitse på, hvordan forskellige dele af

programmet kunne se ud. De vil ikke nødvendigvis være fuldstændig identiske med det færdige produkt, da der sagtens kan findes på bedre ideer eller opdage, at noget ikke virker lige så godt som der havde været forventet. Dog er det stadigvæk en god ting at have som reference. Følgende ses nogle af de mock-ups, der er blevet lavet i forbindelse med design-processen.

Det første mock-up ses i Kode Blok 1. Dette er velkomstskærmen, som er det første brugeren ser når de åbner for programmet. Meningen er, at der skal være en meget enkelt skærm med en kort velkomst tekst, samt et par enkelte essentielle kommandoer.

```md
*Fancy Welcome Message*

Enter h for help
Enter q to quit

--------------------------------------------
Enter Command:
```

Kode Blok 1: Mock-up af velkomst-skærm

I Kode Blok 2 ses der et forslag på hvordan `location` kommandoen kunne fungere. Meningen er at brugeren skriver `location` eller `l` i kommandobaren, hvorefter de bliver vist frem til denne menu. Derefter kan de skrive et tal for at vælge, hvilken lokation de gerne vil bruge.

```md
(Selecting Road)
----------------------------------------------

 [1] Vej              05-09-2022   05-09-2024
 [2] Vej              05-09-2022   05-09-2024
 [3] Vej              05-09-2022   05-09-2024
 [4] Vej              05-09-2022   05-09-2024
 [5] Vej              05-09-2022   05-09-2024
 [6] Vej              05-09-2022   06-02-2024
 [7] Vej              05-09-2022   12-01-2024
 [8] Vej              05-09-2022   06-01-2024
 [9] Vej              05-09-2022   06-01-2024
 [0] Vej              05-09-2022   06-01-2024

--------------------------------------------
Enter Command:
```

Kode Blok 2: Mock-up af lokation vælger menuen

I Kode Blok 3 ses et mock-up af, hvordan data-skærmen kunne se ud. Øverst vises en header, som indeholder information som, hvilken lokation der vises data fra og, hvilket tidspunkt det er taget fra. Nedenunder er kroppen af skærmen som viser selve dataen.

```md
(Showing General Data For Location)
Vejvej            05-09-2024            14:00
-------------------------------------------
[1] Pm2.5          [2] Pm10          [3] No2
    xxx                xxx               xxx

[4] CO             [5] CH4           [6] Temp
    xxx                xxx               xxx
-------------------------------------------
Enter Command:
```

Kode Blok 3: Mock-up af data-skærmen

6.3 Design af data opsamling

I dette afsnit gennemgås overvejelser i forbindelse med design af hardware og software, til at opsamle data om PM_{10}, $PM_{2.5}$ og NO_2 vha. en Arduino. Målet med designet er at lave præcise målinger af disse luftpartikler og gemme det i en CSV-fil, så det kan bruges i appen til at analysere luftforureningen over tid.

6.3.1 Overordnet design

Der skal designes et system, som kan forvandle luftpartikler i luften til data på computeren. Det er valgt at måle PM_{10}, $PM_{2.5}$, NO_2 og temperatur, da det var de partikler de tilgængelige sensoroer var i stand til at måle. En Arduino stilles op med sensorer, som kan omdanne koncentrationen af disse luftpartikler, samt temperaturen i luften, til spænding i selve Arduino kredsløbet. Derefter kan viden om forholdet mellem koncentrationen eller temperaturen, og spændingen bruges til at udregne den præcise data. Dette forhold afhænger af, hvilke sensorer der bruges, og hvordan de foretager deres målinger. Til sidst skal denne data gemmes i en csv-fil på computeren, så det kan bruges i applikationen.

6.3.2 Udfordringer

Det første skridt er at finde de sensorer, som skal bruges for at lave præcise målinger af PM_{10}, $PM_{2.5}$ NO_2, og temperatur. Derefter skal de tilsluttes korrekt i et kredsløb, og der skal skrives et program, som kan overføre den data til en csv fil, så det kan bruges i appen. Udfordringerne er opsummeret i nedenstående skema.

Udfordringer i designet
Indsamle data vha. Arduino og sensorer Finde præcise sensorer, der kan måle henholdsvis PM_{10}, $PM_{2.5}$, NO_2 og temperatur.
Tilslutte sensorer til Arduinoen og behandle data i en Arduino sketch Sensorerne skal tilsluttes korrekt, og en sketch skal behandle deres input.
Overføre data til en CSV-fil på computeren Data fra Arduinoen skal overføres til en CSV-fil på computeren, så det kan bruges i appen.

6.3.3 Løsninger

Sensorerne skal fungere med 3.3V eller 5V, da det er de spændinger, Arduinoen understøtter. Sensorerne skal være nemme at tilslutte, f.eks. via et breadboard, og understøtte de ønskede måltyper: PM_{10}, $PM_{2.5}$, NO_2 og temperatur. Disse sensorer matcher de data, som Københavns Kommune også måler, og som appen er designet til at vise.

De fleste Arduino-kompatible sensorer fungerer ved at måle spænding eller strøm, som ændrer sig afhængigt af de fysiske forhold sensoren registrerer. For eksempel bruger en $PM_{2.5}$-sensor en laser til at tælle partikler i luften, mens en termistor måler temperatur ved at ændre sin modstand.

I projektet skal der være flere sensorer forbundet til arduinoen på en gang, her er breadboardet en stor fordel. Der laves en såkaldt "sketch", som kan måle og behandle data fra alle sensorerne på samme tid. Der bruges Arduino's officielle IDE til at lave sketchen. Fordelen ved at bruge den er, at den har et indbygget bibliotek, som gør det nemt at programmere Arduinoen.

Arduino-sketchens kode anvender Serial-klassen til at etablere en kommunikationsport mellem Arduinoen og en computer. Via denne port kan data sendes til computeren i realtid. For at konvertere data til en CSV-fil laves der et separat C-program, der læser fra porten og gemmer data i CSV-format, med tidsstempler i unix-time.

Løsninger til designet
Arduino-kompatible sensorer Der vælges sensorer, som er kompatible med Arduino og nemt kan tilsluttes via breadboard.
Arduino IDE til programmering af sensorerne Arduino's officielle IDE bruges til at programmere Arduinoen til at opsamle data.
Serial-port og C-program til CSV-konvertering `Serial`-klassen bruges til at kommunikere med computeren, og et C-program bruges til at læse fra porten og skrive til en CSV-fil.

7 Implementering

Designet som blev beskrevet i det forrige kapitel er derefter blevet brugt til at lave en konkret terminal-baseret applikation, samt en dataopsamler til at indsamle data om luftforurening. I dette kapitel gennemgås selve udviklingen af disse ting, herunder de mere tekniske detaljer, alle de udfordringer det blev stødt på, og de løsninger der blev fundet frem til under implementeringen af projektet.

7.1 Implementering af applikationen

I dette under-kapitel beskrives implementeringen af den terminalbaserede applikation. Først præsenteres de anvendte teknologier og værktøjer, efterfulgt af en beskrivelse af applikationens modulære opbygning. Derefter gennemgås implementeringen af de forskellige dele af koden.

7.1.1 Anvendte teknologier og værktøjer

Programmeringssprog

Implementeringen af applikationen var udviklet med programmeringssproget C, da dette var et af kravene til projektet. Der var dog stadigvæk flere fordele ved at bruge C til udviklingen. C er kendt for dens effektive ydeevne eftersom det er et lavt niveau programmeringssprog. Dette betyder at det er tættere på hardwaren, og giver mere kontrol over hvad computeren helt præcist gør. Denne ydeevne betyder at applikationen vil være i stand til at behandle de potentielt store mængder data hurtigere end andre højere niveau programmeringssprog som Python. Det betyder også at applikationen opbruger færre resurser fra den computer den kører på.

Dog er dette på en bekostning af at mange opgaver er mere komplekse i C end i mange andre programmeringssprog, og derved tager udviklingen også længere tid end ellers. Dette er blandt andet string manipulation. Her tilbyder fx Python mange funktioner der gør det betydeligt nemmere, hvorimod disse ting skal gøres manuelt i C. Da disse opgaver skal løses manuelt, giver det også mulighed for at introducere mange fejl som ikke ville have opstået ved brug af andre sprog. Derfor kan det diskutters hvorvidt C er det bedste programmeringssprog til denne applikation. I en applikation som denne hvor datamængden på nuværende tidspunkt ikke er overvældende stor, og hvor udviklingstiden er begrænset, kunne et højere-niveau sprog være et mere optimalt valg. Den hurtigere udviklingstid ville gøre det muligt at implementere hurtigere og føre til et program med flere funktioner. Udover det ville der også være mindre risiko for fejl, hvilket også fører til længere udviklingstid, og eventuelt dårligere brugeroplevelse hvis de ikke bliver løst.

Dog giver C også en mere fremtidssikret løsning. Hvis datamængden skulle vokse betydeligt i fremtiden, eller hvis resursekrævende funktioner skulle implementeres, kunne C's effektivitet og kontrol over computeren være værdifuld.

C Standard Biblioteker

Under udviklingen af applikationen er der også blevet anvendt forskellige standard C biblioteker.

Bibliotek	Beskrivelse
`<stdio.h>`	"Standard Input/Output". Giver adgang til funktioner som gør brugeren i stand til at interegere med applikationen. Nogle af disse er `printf()` som bruges til at skrive ting ud til terminalen, samt `scanf()` og `fgets()` som kan bruges til at tage imod brugerens input.
`<stdlib.h>`	"Standard Library". Inkluderer funktioner til mere generelle operationer, blandt andet `strtol()` som konverterer en string til en integer. Det indeholder også funktioner for dynamisk hukommelse allokering som `malloc()`, hvilket var anvendt i tidligere versioner af applikationen
`<string.h>`	"String Handling". Dette bibliotek af funktioner som har at gøre med at arbejde med strings. Blandt de inkluderede funktioner er `strcpy()`, som kopirer en string fra en variabel til en anden og `strtok()` som splitter en string op i tokens baseret på et separator-tegn. `strtok()` er især relevant når det kommer til at parse CSV-data, da det er separeret af et bestemt tegn.
`<time.h>`	"Time". Indeholder funktioner til at arbejde med dato og tid. Dette er blandt andet `time_t` datatypen som bruges til at lagre et bestemt tidspunkt i unixtime. Dette gør det nemmere at lave tidsrelaterede udregninger og konvertering.

Tabel 2: Standard biblioteker anvendt i applikationen (ISO, 2024)

7.1.2 Datahåndtering

Programmet som er udviklet under dette projekt er afhængig af præcise data. I dette afsnit vil det beskrives hvordan data fra Arduinoen og sensorerne bliver struktureret og behandlet i programmet.

Data indeholder værdier for $PM_{2.5}$, PM_{10} og NO_2. Data bliver sat op i en matrice hvor hver række repræsentere data for én time. Matricen ser ud som følger:

Tid[unix]	**PM$_{2.5}$** $[\frac{\mu g}{m^3}]$	**PM$_{10}$** $[\frac{\mu g}{m^3}]$	**NO$_2$** $[\frac{\mu g}{m^3}]$	Temperatur [℃]
1733378400	11,4	30	45,8	12

Tabel 3: Datastruktur

Når data er sat op således er det for at gøre det let at tilgå et enkelt datapunkt eller en række udvalgte punkter til visning. For at gøre det nemmere at arbejde med data, er hvert sæt af data (for hver vej), opdelt i hver sin matrice.

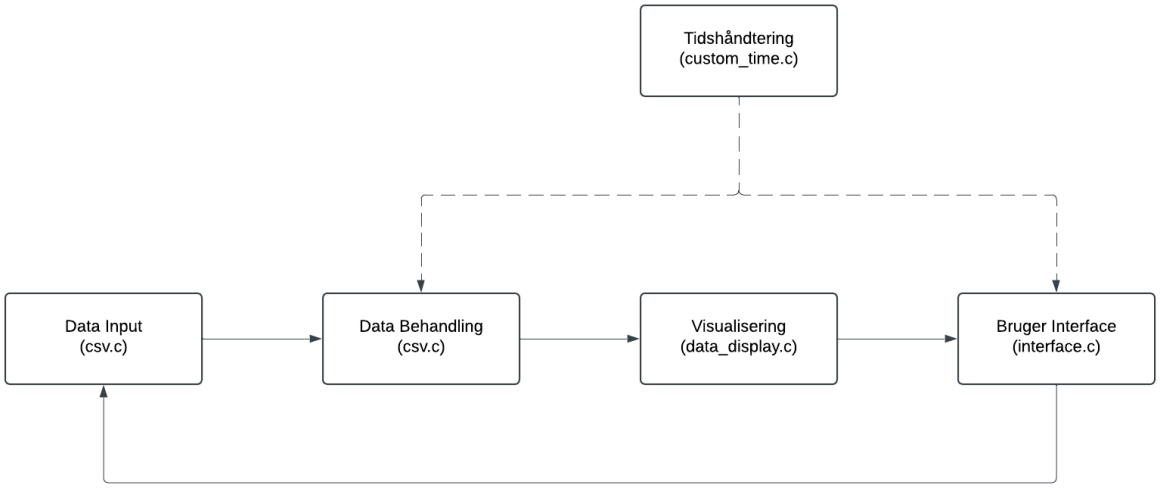
Den første kolonne indeholder tid i formatet unix-time. Unix-time er valgt for at gøre programmet i overensstemmelse med international standarder. Unix-time

er også let at regne med og mindre tilbøjeligt til at blive misforstået. Selvom unix-time er lettere at regne med, er det svært at læse for mennesker, og derfor kræver det at der kan konverteres mellem unix time, og "almindelig tid", for at kunne vise dette til brugeren i programmet.

Enheden, for $PM_{2.5}$, PM_{10} og NO_2, er $\mu\frac{g}{m^3}$ da dette er denne enhed der anvendes på tværs af fundne kilder, og desuden er den enhed som data fra Københavns kommune leveres i. Enheden er implicit og ikke i matricen. Tilsvarende har den sidste kolonne grader celsius som enhed.

7.1.3 Applikationens Arkitektur

Applikationens kode er opbygget efter et modulært design. Dette betyder at de forskellige dele af programmet er splittet op i forskellige moduler baseret på hvad deres formål er. I dette tilfælde er applikationen opdelt efter datainput og datahåndtering, visualisering, bruger interface, og tidsrelaterede funktioner. På denne måde er det nemmere at have et overblik over applikationens kode samt at implementere nye funktioner, da det er nemmere at finde ud af hvor forskellige funktioner kan findes henne. Udover det er det også nemmere at debugge applikationen, da der nemmere kunne findes frem til, hvor en eventuel fejl er opstået. På Figur 3 kan der ses et diagram over de forskellige moduler og hvordan de relaterer til hinanden.



Figur 3: Diagram over de forskellige moduler

Det primære modul er bruger interface modulet, lokaliseret i `interface.c` filen. Det er her den primære logik i forhold til applikationen finder sted, blandt andet håndering af de forskellige skærme og kommandoer. Med vise kommandoer og skærme vil der så muligvis gøres brug af de andre moduler.

Visualiseringsmodulet indeholder funktioner som anvendes til visualisere data. Dette modul er placeret i `data_display.c` filen, og er blandt andet funktioner til at generere grafen over et specifikt tidsrum.

Data behandlings modulet, hvis funktioner kan findes i `csv.c` filen, består af funktioner som har til formål at behandle data på forskellige måder. Dette kan blandt andet være at filtrere data så det kun inkluderer det der er målt indenfor et bestemt tidspunkt.

Data input modulet indeholder funktioner som anvendes til at parse data fra en CSV-fil til et format som data behandlings modulet kan anvende. Selvom data behandlings modulet og data input modulet er i den samme fil `csv.c`, kan det stadig argumenteres at de logisk set er separate moduler. Funktionerne i filen er enten udelukkende til data input eller til data håndtering, og derfor er de stadigvæk adskilte. Det er dog valgt at have dem i samme fil grundet den relativt lille størrelse på funktionerne, samt at de er tæt relaterede.

Til sidst er der også tidshåndtering modulet. I modsætning til de andre moduler fungerer det mere som et hjælper-modul der bruges i flere forskellige moduler. Det indeholder funktioner brugt i flere forskellige moduler, og er anvendt til tidsrelaterede opgaver som ikke kunne løses udelukkende med funktioner fra `<time.h>` biblioteket.

Som set på Figur 3 er der generelt et dataflow der går fra interface modulet, igennem data input modulet, videre til data behandlingsmodulet, hen til visualiseringsmodulet og til sidst tilbage til interface modulet. Et eksempel på dette dataflow ses i forbindelse med kommandoer der involverer data visualisering. Kommandoen eksekveres fra interface modulet, hvorefter den data der skal bruges bliver hentet fra CSV-filen med data input modulet. Data behandlingsmodulet behandler dataene til det som kommandoen anmoder, hvorefter visualiserings modulet skaber den visuelle repræsentation af det. Til sidst vises dette på terminalen tilbage i interface modulet. Under denne proces anvendes til tider funktioner fra tidshånderings modulet som vist med de stiplede linjer på Figur 3.

7.1.4 Implementering af modulerne

Hvordan data display, data input, visualisering, og (måske) tidshåndtering fungerer. Interface modulet beskrives senere i sit eget afsnit da det er det centrale modul og her alle de forskellige moduler anvendes.

Visualisering

Til at visualisere grafen anvendes funktionen `draw_graph()` som set i Kode Blok 4.

```c
void draw_graph(int size, double *data, float max_val, float threshold)
{
    ...
    for (int step = steps; step >= 0; step--) {
        float i = step * stepVal;

        printf("%s", i >= 10.0 ? "" : " ");
        printf("%s%.1f||", i <= threshold ? GREEN : RED, i);

        for (int j = 0; j < size; j++) {
            printf("%s", data[j] >= i ? "### " : "    ");
        }
        printf("\n%s", RESET_COLOUR);
    }
    printf("      ");
    for (int i = 0; i < size; i++) {printf("====");}
    printf("\n       ");
    for (int i = 1; i <= size; i++) {printf("%2d  ", i);}
    printf("date");
}
```

Kode Blok 4: `draw_graph()` funktionen (delvist)

Siden programmet er begrænset til en terminal, kan der kun tegnes fra øverste venstre til nederste højre. Derfor er det nødvendigt at iterere gennem matricen bagfra, startende med det sidste element i hver række.

Til at printe søjlerne, vælge farver og indsætte mellemrum bruges ternaries operatore inde i en print-sætning, for at holde koden lidt mere overskuelig. ternaries operatore er en form for kompakt if-statement som kan skrives i en enkelt linje. Selvom ternaries operationer generelt ikke anbefales til mere komplekse kodestykker, er de perfekt egnede til disse simple betingelser.

Værdierne for akseinddelingen defineres tidligere i koden for at tillade dynamisk skalering af grafen. Dette gøres simpelt ved at sammenligne det største tal, grafen skal tegne, med en fast float-værdi, og bestemme om det er større eller ej.

For hver værdi kontrolleres det, om den overstiger en bestemt tærskelværdi. Hvis dette er tilfældes ændres farven. Dette gøres gennem ANSI escape-koder som “x1b[31m”, der er defineret som konstantvariabler i starten af koden. Dernæst tjekkes værdien mod y-aksen. Hvis værdien er lig med eller over y-akseværdien, printes en søjle, ellers printes der 4 mellemrum for at efterlade et tomrum. Efter hver iteration nulstilles farven i forberedelse til den næste. Selvom dette kunne optimeres yderligere, bruger print-sætninger generelt ikke nok CPU-cyklusser til, at eventuelle optimeringer ville være mærkbare, før der rammes en milliard plus datapunkter, hvilket ingen af datasættene kommer i nærheden af.

Til sidst printes x-aksen ved simpelthen at printe en linje "=" og x-tal nedenunder. En bedre tilgang ville have været at printe datoerne nedenunder, men der var ikke tid til det.

Grafsystemet, da dette var en af de større dele af projektet, havde også en Python-prototype, som er vedhæftet denne fil. Koden er meget mindre struktureret og optimeret, men gør stort set det samme som koden ovenfor (mere eller mindre).

Data Input

Projektet her benytter det bredt brugte ".csv" format til data opbevaring.

CSV står for "comma seperated values" og er et format som oftest bruges i industrien til at opbevare store mængder indsamlet data på en hurtig, effektiv og simpel måde, hvilket gør det perfekt til dette formål. CSV består af rækker og koloner. Hver kolone er defineret af kommaer, som navnet antyder, men kan også være defineret af punktum, kolon og semikolon, osv. Standarden er bare komma. Rækker er blot defineret af enter.

I C læses en CSV fil ved først at definere "delimiteren" altså om det er komma, punktum, osv; derefter åbnes filen i "read only" for at undgå data korruption i tilfælde af fejl. Filen vil nu blive læst linje efter linje, og hver linje læst delimiter efter delimiter. Dette giver to lykker, hvor summen bliver et index [row, column] og en værdi, som nemt kan appendes til en matrix for yderligere brug i programmet.

```c
1 if (fgets(line, sizeof(line), data) == NULL) {
2     fclose(data);
3     return -1;  // Failed to read header line
4 }
```

Kode Blok 5: Fejlhåndtering i tilfælde af tom fil

Denne linje er af særlig betydning for denne Proof of concept, da den skipper header dataen i CSV-filen (linje 1). Header data er "hardcoded" i programmet, og header data ville som resultat være ukendte værdier som kunne forårsage fejl eller forkerte værdier.

7.1.5 Bruger Interface Implementering

Som forklaret i designproces kapitlet (Afsnit 6), er applikationens bruger interface designet til at give brugeren en mere visuel og intuitiv måde at se den indsamlede data på. Målet er at brugeren nemt skal kunne navigerer til den data de har brug for at undersøge, og få en klar visualisering af den ønskede data.

Dette afsnit beskriver implementeringen af det terminalbaserede bruger interface, og hvordan designet beskrevet tidligere er blevet oversat til et konkret program ved brug af programmeringssproget C.

Data Struktur og organisering

For at gøre koden til interfacet mere overskueligt og læsbart, er der gjort brug af enums, structs og lookup-tables til at organisere forskellige dele af programmet.

Enums er brugt til at repræsentere et heltalsværdi som en navngiven variabel, og kan også kategoriseres efter bestemte data typer baseret på hvad de er brugt til. I denne applikation er der anvendt enums for at repræsentere forskellige kommandoer, skærme, lokationer og målingstyper. Dette forbedrer kodens læsbarhed da de numeriske værdier erstattes med navne der angiver hvad værdierne repræsenterer. Et eksempel på en enum er `Command_id`, som kan ses i Kode Blok 6. Denne enum repræsenterer de forskellige kommandoer som brugeren kan anvende i applikationen.

```c
typedef enum {
  CMD_QUIT,
  CMD_HELP,
  CMD_RESET,
  CMD_LOCATION,
  CMD_GRAPH,
  CMD_TIME,
  CMD_DATA,
  CMD_TIMESPAN,
  CMD_INFO
} Command_id;
```

Kode Blok 6: Erklæring af kommando enums

I dette eksempel er `CMD_QUIT` lig med 0, `CMD_HELP` lig med 1 osv. I stedet for at bruge disse tal i koden kan der i stedet skrives disse navne hvilket er meget mere overskueligt.

En anden data struktur anvendt i applikationen er structs, som tillader dig at groupere variabler af forskellige datatyper under en enkelt datatype og navn. Structs kan blandt andet anvendes til at repræsentere objekter med flere attributter. Dette er endu en ting der gør koden mere læsbar og overskuelig, da der i stedet for at skulle holde styr på mange forskellige variabler, bare kan have en enkelt struct som altid har den data der er brug for.

Et eksempel på structs anvendt i denne applikation er `command_entry` og `location_entry` som set i Kode Blok 7. Disse structs bliver til at forbinde en enum med den kommando eller lokation den repræsenterer i form af en string. Dette er nødvendigt for at programmet senere hen skal kunne aflæse hvilken kommando brugeren har skrevet. De bliver også brug i sammenhæng med lookup-tables som vil blive beskrevet senere i dette afsnit.

```c
// Struct for each command entry
struct command_entry {
    char* command_string;
    Command_id command_id;
};

// Struct for location entries
struct location_entry {
    char* filename;
    Location location_id;
};
```

Kode Blok 7: Erkæring af structs for kommandoer og lokationer

Den struct der bruges til kommandoer har to variabler. Den første er den string der skal skrives for at anvende kommandoen, den anden er den enum variabel den bliver repræsenteret af. Det samme gælder for den struct der anvendes til lokationer, her er string variablen blot navnet på filen til lokationen i stedet.

Et eksempel på en større struct er `program_state` som set i Kode Blok 8. Denne struct er brugt til at holde styr på programmets state, såsom hvilken skærm applikationen viser, eller hvilket tidpunkt den viser data for.

```c
// Struct for program state
struct program_state {
    Screen current_screen;
    time_t current_time;
    Location current_location;
    Measurement_type current_measurement;
    struct timespan current_timespan;
    int running;
};
```

Kode Blok 8: Struct for programmets state

Ved at have denne information i en struct, kan der i stedet for at holde styr på seks forskellige variabler bare have denne ene struktur som altid indeholder alt information om programmet som der kunne være brug for.

Til sidst er der også anvendt lookup-tables, som er en data struktur der anvendes til at implementere en associativt array som knytter nøgler til værdier. Det fungerer på samme måde som en dictionary i JavaScript eller Python, hvor der kan bruges en nøgle til at retunere den værdi den knytter sig til. I dette program er det brugt i sammenhæng med kommandoer som set i Kode Blok 9 samt filassociering i en lignende array. Det lookup-table som er brugt til kommandoer er implementeret med en array af den tidligere nævnte `command_entry` struct. Et

lookup-table kan også implementeres med andre strukturer såsom et hash-table, men disse metoder er mere komplicerede end en simpel array. De giver typisk kun mening hvis der besiddes meget store lookup-tables, eller hvis optimering er ekstremt vigtigt. Ingen af delene er tilfældet her.

```
1 [sprog]
2 [kode]
```

Kode Blok 9: titel

Ved at bruge et lookup-table her, er det meget enkelt at finde ud af hvilken enum variabel der er knyttet til en bestemt kommando. For eksempel hvis en bruger indtaster `quit` kommandoen kan programmet meget nemt finde ud af at der er tale om `CMD_QUIT` kommandoen uden at skulle gøre brug af en lang if-else kæde eller lignende. Det gør det også meget nemmere at udvide koden med nye kommandoer.

Program flow

Logikken til hele programmet kan findes i `main()` funktionen, som indeholder initialiseringen af programets state, samt programmets primære loop. Koden til denne funktion kan ses i Kode Blok 10.

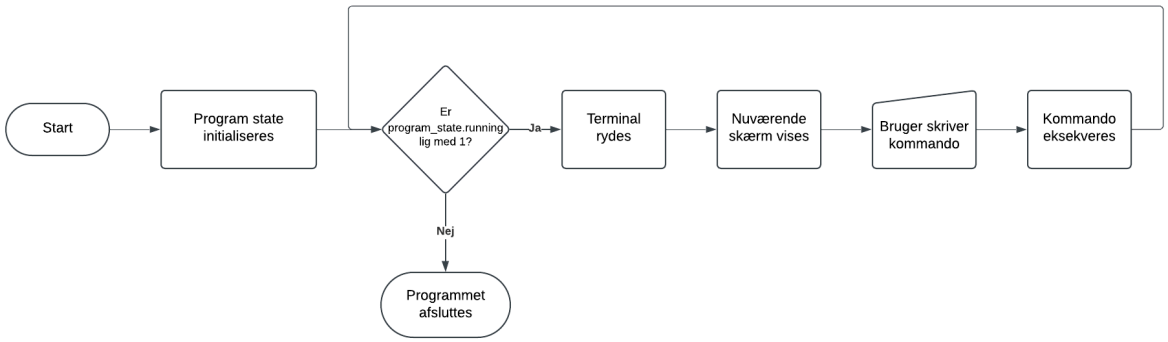
```c
int main() {
    struct program_state program_state = {
            SCREEN_MAIN,
            1728932400,
            FOLEHAVEN,
            PM2_5,
            {
                1727992800,
                1728856800
            },
            1
    };
    char user_input[50];

    while(program_state.running) {
        struct entered_command entered_command;
        clear_terminal();

        display_screen(program_state);
        get_input(user_input);
        entered_command = get_command(user_input);

        execute_command(entered_command, &program_state);
    }

    return 0;
}
```

Kode Blok 10: `Main()` Funktionen

`Main()` funktionen starter med at initialisere programmet med nogle standard værdier. Dette sker på linje 2 hvor en `program_state` struct bliver erklæret med de værdier programmet skal have når brugeren åbner det. Dette er blandt andet at programmet skal state på `SCREEN_MAIN` skærmen, og skal starte med at vise data fra `FOLEHAVEN` lokationen.

Senere på linje 15 ankommes til programmets primære loop, som kører så længe `running` variablen i `program_state` er lig med 1. Denne variabel kan også skrives som `program_state.running`. Hele programmets levetid vil bestå af at gå over dette loop igen og igen. Loppet begynder med at erklære en `entered_command` struct der skal lagre brugerens indtastede kommando, hvorefter terminalen bliver rydet på linje 17 for at gøre plads til at printe den nuværende skærm til brugeren. Dette gøres med `display_screen()` funktionen som tager programmets state som argument så den har information om hvilken skærm den skal vise. Bagefter at have vist skærmen afventer programmet brugerens input i form af en

kommando ved brug af `get_input()`. Dette input lagres i `user_input` variablen. For at finde ud af hvilken kommando brugeren har indtastet anvendes `get_command()` funktionen på linje 21, hvor dette lagres i `entered_command` fra tidligere. Til sidst eksekveres den pågældende kommando med `execute_command()` funktionen, hvor disse kommandoer ofte ændrer på programmets state, og derved vil programmet være forandret i det næste loop. Efter at kommandoen er eksekveret starter loopet forfra, og dette fortsætter indtil `program_state.running` ikke længere er lig med 1. Der vil dykkes mere ned i hvordan disse forskellige funktioner fungerer i de næste afsnit. Loopet kan også forklares i form af et flow chart som set på Figur 4



Figur 4: Flow chart over det primære loop

Kommando Parsing

Den første del af kommando håndteringen er at finde ud af hvilken kommando brugeren har skrevet. Til dette bruges `get_command()` funktionen til at finde den enum konstant som kommandoen passer til. Udover det håndterer den også eventuelle argumenter, og retunerer det samlet i form af en `entered_command` struct. Funktionen kan ses i Kode Blok 11.

```c
struct entered_command get_command(char *input_string) {
    int command_id = -1;
    char command_string[50];
    char argument_string[50];

    // Creates a pointer to the first instance of a space.
    char *space_pointer = strchr(input_string, ' ');

    if (space_pointer != NULL) {
        // Copies the command part of the input to command_string.
        strncpy(command_string, input_string, space_pointer - input_string);

        // Copies the argument part of the input to argument_string
        strcpy(argument_string, space_pointer + 1);
    } else {
        strcpy(command_string, input_string);
    }

    // Finds the command id of the command_string (if it exists)
    for (int i = 0; i < sizeof (command_table) / sizeof (command_table[0]); i++) {
        if (strcmp(command_string, command_table[i].command_string) == 0) {
            command_id = command_table[i].command_id;
            break;
        }
    }

    struct entered_command entered_command = {command_id, argument_string};

    return entered_command;
}
```

Kode Blok 11: `Main()` Funktionen

Funktionen starter med at se om der er et mellemrum i den indtastede input, da dette indikere at brugeren har indtastet et argument sammen med kommandoen. En pointer til dette mellemrum lagres i `space_pointer`. Hvis et mellemrum blev fundet køres koden i linje 10-15. Her tager programmet den del af inputtet der kommer før mellemrummet (selve kommandoen) og kopier det til `command_string` variablen. Delen efter mellemrummet bliver ligeledes kopiret til `argument_string` variablen. I tilfælde af at inputtet ikke indeholder et mellemrum, bliver hele inputtet bare kopiret til `command_string`.

Den næste del af funktionen er der hvor programmet finder ud af hvilken kommando værdien i `command_string` er lig med. Denne del forgår i linje 20-25, og her anvendes in linear search algoritme til at søge igennem det lookup-table der er oprettet til kommandoerne (`command_table`). Dette linear search betyder at den går igennem hver eneste struct i `command_table` indtil den finder en hvor værdien af `command_string` er det samme som værdien af `command_string` i den pågældende struct. Hvis den finder sådan en struct, tages `command_id` variablen fra den struct og lagres i den lokale `command_id` variabel. Til sidst retuneres dette id sammen med et eventuelt argument i en samlet struct.

Kommando Eksekvering

Efter at `get_command()` har identificeret den indtastede kommando, sendes dette til `execute_command()` funktionen hvis job det er at eksekvere denne kommando. Denne funktion kan ses i Kode Blok 12.

```c
// Executes a command based on its id using a switch statement
void execute_command(struct entered_command entered_command, struct program_state *program_state) {
    switch (entered_command.command_id) {
        case CMD_QUIT:
            command_quit(program_state);
            break;
        case CMD_HELP:
            command_help(program_state);
            break;
        case CMD_RESET:
            command_reset(program_state);
            break;
        case CMD_LOCATION:
            command_location(program_state);
            break;
        // [SNIP]
    }
}
```

Kode Blok 12:

```
    execuAarhus Universitets rapport leverer en omfattende analyse af
  luftkvaliteten på nationalt niveau ved at anvende avancerede metoder som
  målestationer og modelberegninger. Rapporten giver dybdegående indsigt i
      forureningskilder som fine partikler (PM#sub[10] og PM#sub[2.5]),
  nitrogendioxid (NO₂) og svovldioxid (SO₂), som ikke kun stammer fra trafik,
men også fra industrielle processer og energiproduktion. Denne tilgang sikrer
en bred forståelse af, hvordan forskellige former for luftforurening påvirker
                          miljøet og folkesundheden.
                                    @AUData
  Københavns Kommunes rapport fokuserer på en lokal kontekst og fremhæver de
  mange kilder til luftforurening i bymiljøer. Ud over trafik og brændeovne
  belyser rapporten også forurening fra bygge- og industrisektorer samt andre
  aktiviteter, der bidrager til emissioner af skadelige stoffer som kulilte
  (CO) og ozon (O₃). Rapporten analyserer forureningens rumlige fordeling og
  dens komplekse interaktioner med byens klima og beboernes sundhed. Ved at
     kombinere disse rapporters nationale og lokale perspektiver opnås en
  helhedsorienteret forståelse af luftforureningsproblematikken og dens mange
    kilder. Dette er afgørende for at kunne udvikle effektive og målrettede
                  bæredygtige løsninger. @KUDatate_command()
```

Funktionen

Funktionen fungerer med en simpel switch statement som kalder en bestem funktion baseret på hvilken enum konstant der blev givet i funktionens argument. Hver af disse funktioner indeholder instruktionerne for den pågældende kommando. Et par eksempler på enkle kommandoer kan ses på Kode Blok 13.

```c
void command_data(struct program_state *program_state){
    program_state->current_screen = SCREEN_DATA;
}

void command_quit(struct program_state *program_state) {
    clear_terminal();
    program_state->running = 0;
}
```

Kode Blok 13: `execute_command()` Funktionen

Den første kommando `command_data()` har det formål at ændre den nuværende skærm til data skærmen. Dette gøres på linje 2, hvor `current_screen` variablen i `program_state` ændres til den korrekte enum variabel. Det andet eksempel er `command_quit()` som er den funktion der kører når programmet skal lukkes. Det fungerer cirka på samme princip med at ændre en variabel i `program_state` i dette tilfælde `running` variablen.

Nogle kommandoer er mere komplicerede og kan blandt andet tage imod argumenter eller kræver yderligere input fra brugeren. Et eksempel på sådan en kommando er `command_time()` som er funktionen der bruges når brugeren indtaster en kommando til at ændre det tidspunkt de ser data for. Kommandoen kan bruges med et argument for et bestemt tidspunkt, men den kan også indtastes uden argument, hvor brugeren så bliver spurgt tidspunktet som et separat input i stedet. Den første del af kommandoen kan ses i Kode Blok 14, og er den del af koden der håndterer et eventuelt argument.

```c
1  void command_time(struct program_state *program_state, char* argument)
   {
2      char input[32];
3      time_t new_time;
4      clear_terminal();
5
6      // if the user gave an argument, attempt to change the time to that
       argument, otherwise continue as if no argument was given
7      if (argument[0] != '\0') {
8          new_time = string_to_unixtime(argument);
9          if (new_time != -1) {
10             program_state->current_time = new_time;
11             return;
12         }
13     }
14     // [Print statement for user input]
…                                        …
31 }
```

Kode Blok 14: Del 1 af `command_time()`

Funktionen starter på linje 7 med at tjekke om der er en string værdi i argument variablen. Hvis brugeren ikke har inkluderet et argument vil variablens første værdi være en null terminator (`\0`) hvilken indikerer at den er tom. Hvis der er et argument konverteres denne string værdi til unixtime og hvis argumenteret konverteres korrekt, sættes `program_state.current_time` til denne værdi og funktionen retuner. Hvis konverteringen fejlede (typisk grundet et forkert format på argumentet) fortsætter programmet som hvis der ikke blev givet et argument.

Dette leder videre til den anden del af funktionen, som beder brugeren om et nyt input som til at ændre tidspunktet. Dette ses i Kode Blok 15.

```c
void command_time(struct program_state *program_state, char* argument)
{
…                                       …
    while(1) {
        // use fgets as the input requires a space (which scanf isnt able to handle)
        fgets(input, sizeof(input), stdin);
        input[strcspn(input, "\n")] = '\0';
        clear_terminal();

        // converts the string to unixtime
        new_time = string_to_unixtime(input);

        // if there are no errors, set the current time to the new time and exit loop
        if (new_time != -1) {
            program_state->current_time = new_time;
            break;
        }
        // [Print statement for user input]
    }
}
```

Kode Blok 15: Del 2 af `command_time()`

Dette fungerer på samme måde som den første del af koden. Først tager programmet brugerens input, konvertere det til unixtime, og sætter `program_state.current_time` til denne værdi hvis den er valid. Hvis inputtet har et forkert format spørges brugeren om et nyt input hvilket fortsætter indtil inputtet er korrekt.

Skærm håndtering

For at vise en bestemt skærm til brugeren anvendes funktionen `display_screen()`. Denne funktion fungerer på samme måde som `execute_command()` fra tidligere, i det den indeholder en switch statement som kører en bestemt funktion baseret på hvilken enum konstant den nuværende skærm er sat til. Funktionen kan ses i Kode Blok 16.

```c
void display_screen(struct program_state program_state) {
    switch (program_state.current_screen) {
        case SCREEN_MAIN:
            screen_main();
            break;
        case SCREEN_HELP:
            screen_help();
            break;
        case SCREEN_DATA:
            screen_data(&program_state);
            break;
            // [SNIP]
    }
}
```

Kode Blok 16: `diplay_screen()` Funktionen

Funktionerne for de forskellige skærme er implementeret på samme måde som kommandoerne, men de er typisk simplere og består primært af kald til printf() eller til funktioner fra de andre moduler. Et eksempel på sådan en funktion kan ses i Screen Data.s Dette funktion er kaldt når data skærmen skal vises, og indeholder information om målinger for et bestemt tidspunkt. Hello

```c
void screen_data(struct program_state *program_state) {
    char *filename = "none";

    // gets the filename for the location
    for (int i = 0; i < sizeof (location_table) / sizeof (location_table[0]); i++) {
        if (program_state->current_location == location_table[i].location_id) {
            filename = location_table[i].filename;
            break;
        }
    }

    double data[5];
    if (get_data_for_date(filename, data, program_state->current_time) == -1) {
        printf("Couldnt load data for file: %s\n", filename);
        return;
    }
    print_data(data);
}
```

Kode Blok 17: `diplay_screen()` Funktionen

Funktionen starter med at bruge det lookup-table der hedder `location_table` til at finde ud af hvilken fil den skal hente data fra. Derefter bruger den `get_data_for_date()` funktionen til at hente data for dette tidspunkt. Derefter bliver denne data givet til `print_data()` funktionen som står for at printe skærmen til terminalen.

7.2 Implementering af data opsamling

I dette afsnit gennemgås hvordan der opsamles data til projektet, hvilke sensorer der er blevet brugt, og hvilke udfordringer der har været havde undervejs.

7.2.1 Sensorer

Der skal findes sensorer som kan måle: PM_{10}, $PM_{2.5}$, NO_2 og temperatur. Der er kommet frem til disse sensorer:

- SEN54-SDN-T Miljøsensor til måling af PM_{10} og $PM_{2.5}$
- MICS-2714 gas sensor til måling af NO_2
- Til temperatur bruges der en simpel termistor

SEN54-SDN-T (RS, u.å.) er egentlig ikke lavet til Arduinio, og den kan tilsluttes via et USB, men det var den eneste let tilgængelige sensor, som målte PM_{10}, $PM_{2.5}$. Den gør det ved at sende laser ind i luften, og måle på refleksionen. Dog var det ikke til at få fat i denne sensor, så springes PM_{10} og $PM_{2.5}$ målingerne over.

Den næste sensor MICS-2714 er en lille sensor, som kan sættes i et breadboard og måle forskellige gasser som for eksempel NO_2. Dog kunne den heller ikke bruges, da den kun kunne måle gasser når de kom op på højere niveauer end normalt målt på gade niveau i København. Sensoren var primært lavet til at måle gasudslip f.eks. i et laboratorie. Den kan måle NO_2 i intervallet 0,05-10ppm (DFRobot, 2021) (parts per million). Det vil sige at den ikke kan måle niveauer under 0,05ppm. Hvis der antages en temperatur på 25 grader celsius og et tryk på 1 atmosfære, så kan det omregnes til $\mu\frac{g}{m^3}$ vha. denne formel:

Molekylvægten for NO_2 er 46,01 g/mol og 0,05ppm = 50ppb

så:

$$K = \frac{46.01 * 50}{24.45} \approx 94,09$$

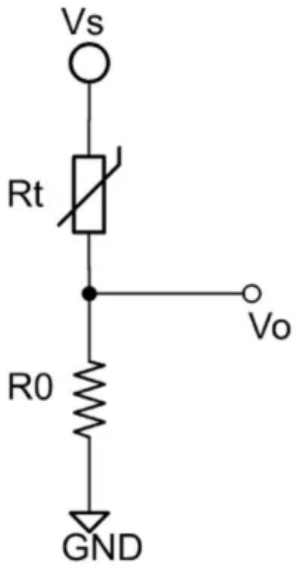
94,09 $\frac{\mu g}{m^3}$ er højere end det normale NO_2 niveau i atmosfæren, så hvis den ikke kan måle niveauer lavere end det, kan den desværre ikke bruges til dette projekt. Det er derfor kun muligt at implementere temperatur målingen, og til dette bruges en simpel NTC termistor.

7.2.2 Termistor opsætningen

En NTC termistor er en termistor hvis modstand reduceres når temperaturen stiger. Den har altså en negativ temperaturkoefficient. Det betyder at spændingen stiger når temperaturen stiger. Målet er derfor at måle spændingen, og finde proportionerne mellem temperatur og spænding. Til dette bruges konstanter fundet hos producenten. Der skal bruges termistorens modstand ved 25°C, og dens B-konstant, som repræsenterer forholdet mellem modstand og temperatur over et bestemt interval. Konstanterne til vores termistor (Elextra, u.å.) er:

Rt ved 25°C = 10 kOhm
B = 3977K

Da termistoren er meget sensitiv til temperaturen, sættes en 10kOhm resistor (R0) parallelt med outputtet for at gøre det mere lineært, som vist på Figur 5:



Figur 5: Skitse af kredsløb (Kane, u.å.)

Med dette kredsløb kan temperaturen udregnes ud fra Vo ved at bruge en version af Steinhart-Hart (Kane, u.å.) ligningen:

$$\frac{1}{T} = \frac{1}{\mathrm{T0}} + \frac{1}{B} * \ln\left(\frac{\mathrm{Rt}}{\mathrm{R0}}\right)$$

T er temperaturen som skal findes.
T0 er temperaturen 25°C i kelvin: 25°C = 298.15K
B er konstanten 3977K
Rt er termistorens modstand

Rt kan findes ved at gange ændringen i spænding med R0 så:

$$\mathrm{Rt} = \mathrm{R0} * \left(\left(\frac{\mathrm{adcMax}}{\mathrm{adcVal}}\right) - 1\right)$$

Indsættes det i ligningen fås:

$$\frac{1}{T} = \frac{1}{\mathrm{T0}} + \frac{1}{B} * \ln\left(\mathrm{R0} * \frac{\left(\frac{\mathrm{adcMax}}{\mathrm{adcVal}}\right) - 1}{\mathrm{R0}}\right)$$

Det kan ses at R0 går ud, så det reduceres til:

$$\frac{1}{T} = \frac{1}{\mathrm{T0}} + \frac{1}{B} * \ln\left(\left(\frac{\mathrm{adcMax}}{\mathrm{adcVal}}\right) - 1\right)$$

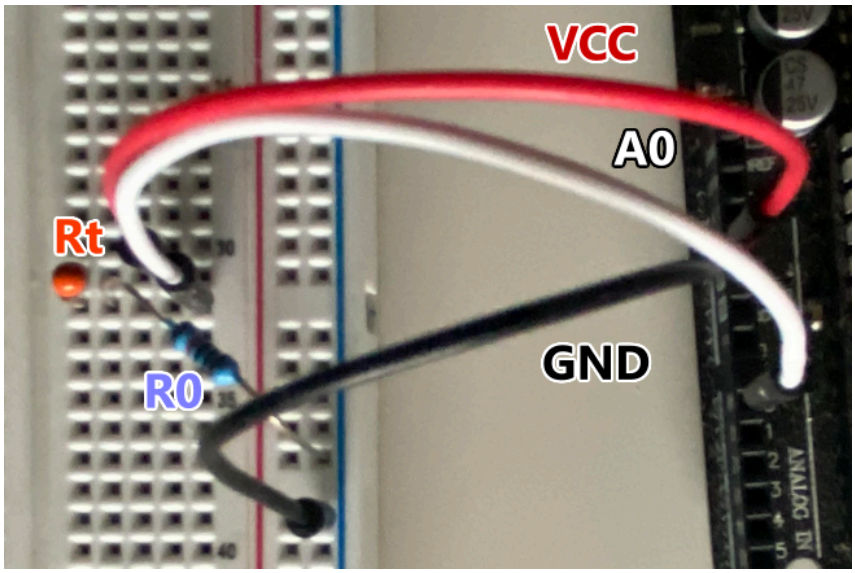
Til sidst isolerers T og konstanterne indsættes, så:

$$T = \frac{1}{\frac{1}{298.15} + \frac{1}{3977} * \ln\left(\left(\frac{1023}{\mathrm{adcVal}}\right) - 1\right)}$$

Det er altså denne formel som skal implementeres i programmet for at finde temperaturen. Det analoge input skal blot sættes ind på adcVal's plads, og omregnes til celsius inden resultatet bliver udskrevet.

7.2.3 Udførelse

Kredsløbet opstilles ud fra skitsen på, og det implementerede kredsløb kan ses på ref til billede



Figur 6: Opstilling af skitsen

Resistoren R0 er 10 kOhm, da ligningen kun virker hvis R0 og Rt ved 25°C er det samme. Outputtet er Vo, som går ind i A0 porten i Arduinoen. A0 er en ADC pin (se evt. Afsnit 4.1), og det er her "adcVal" aflæses i programmet.

```cpp
int tmpPin = A0;
...
void loop() {
  ...
  float reading = analogRead(tmpPin);

  float B = 3977;
  float T0 = 298.15;

  // Temperature in kelvin based on the Steinhart-Hart equation
  Temp = 1 / (1 / T0 + 1 / B  log((1023.0 / reading) - 1));

  // Convert to celcius
  Temp = Temp - 273.15;
```

Kode Blok 18: Fra Sketch.ino

I Kode Blok 18 kan det ses hvordan variablen tmpPin forbindes med A0, aflæser spændingen, og udregner temperaturen ved at bruge ligningen fra sidste afsnit.

Testene fra denne opstilling gav nogle resultater som var meget realistiske, så det antages at konstanterne og opstillingen er korrekt.

7.2.4 Aflæsning og lagring af målinger

Når en Arduino og en computeren skal snakke sammen bruges Arduino's indbygget Serial-klasse, som laver en seriel port, som den kan overføre data til. For at gøre det skal den initialisere den med en baud rate i setup delen af vores sketch. Baud rate er kommunikationshastigheden, hvor en baud rate på f.eks. 9600 betyder 9600 symboler i sekundet. Derefter kan resultaterne printes gennem denne port, som vist her:

```cpp
void setup() {
  Serial.begin(9600);
}

void loop() {
  ...
  // Print in the csv format: PM25,PM10,NO2,Temp
  // right now we only have a temp sensor, so we will
  // print 0 for the rest
  Serial.print(PM25);
  Serial.print(",");
  Serial.print(PM10);
  Serial.print(",");
  Serial.print(NO2);
  Serial.print(",");
  Serial.println(Temp);

  delay(1000);
}
```

Kode Blok 19: Fra Sketch.ino

Som det også fremgår i kommentaren, er de andre målinger initialiseret til 0, og det er kun temperaturen som rent faktisk bliver målt. Dog skal alle værdierne printes ud, så det er kompatibelt med CLI-programmet, da målingerne skal have formatet:

```c
"PM25,PM10,NO2,Temp\n"
```

hvor alle variablerne er af typen float, og slutningen er markeret med et \n.

Det næste mål er at aflæse data og gemme det i en CSV-fil. For at gøre dette gøres der brug af et speciallavet C-program. Det virker ved at læse fra device filen, som er den fil Arduinoen skriver til, så tager den en måling per minut, og efter 60 minutter udregner den gennemsnittet af de 60 målinger og skriver det ind i en CSV-fil med et unixtime tidsstempel.

Programmet er speciallavet til et Linux styresystem. Det er ikke nødvendigt at gøre programmet kompatibelt med andre styresystemer, da det blot er en del af backend, og det bruges kun til at opsamle data i en CSV-fil. Desuden skal selve koden ikke integreres med koden til CLI-programmet.

```c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#include <time.h>

#define SAMPLE_COUNT 60
#define BAUD_RATE B9600 // Adjust for your arduino baud rate
#define DATA_FILE "data.csv"
#define DEVICE_FILE "/dev/ttyACM0" // Adjust for your arduino device file

int configureSerialPort(int fd);
float getAverage(float data[], int size);
void collectDataLoop(FILE* dev_file, FILE* data_file);
```

Kode Blok 20: Fra collect_data.c

Programmet er sat til at have en baud rate på 9600, ligesom i Arduino sketchen. Device filen er sat til /dev/ttyACM0 som er den fil Arduino automatisk skriver til på Linux. Det ses også at der tages 60 data opsamlinger per time og CSV-filen hedder "data.csv". Programmet består generelt set af 2 dele.

1. Konfigurer den serielle port og åben/opret CSV-filen
2. Start et loop som indsamler data hvert minut og skriver gennemsnittet ind i CSV-filen hver time.

For at opnå det første bruges C biblioteket <terminos> som giver funktioner til at indstille serielle porte i forhold til f.eks. baud rate og read permission. Funktionen 'configureSerialPort' indstiller porten til baud rate 9600 og 8N1 mode hvilket betyder at den sender 1 start bit, 8 data bits (1 byte) og 1 stop bit, det vil sige at den sender $\frac{9600}{10} = 960 \frac{\text{bytes}}{\text{sek}}$. Foruden <terminos> bruges en række andre biblioteker som f.eks. <fcntl> til at åbne og lukke filer samt læse og skrive til filer på computeren, og <time> til at indlæse tidsstempler og time programmet.

```c
time_t startUTC = time(NULL);
printf("Data collection starts in: %ld minutes\n", 60 - (startUTC % 3600) / 60);
while (startUTC % 3600 != 0) {
    usleep(1000);  // Wait 1 milli-second, to avoid overclocking the cpu
    startUTC = time(NULL);
}
printf("Collecting data...\n");

clock_t start, end;
while (1) {
    start = clock();

    ...

        count++;

        end = clock();
        usleep(60000000 - (end - start));  // Wait 1 minute, minus the cpu time used
```

Kode Blok 21: Fra collect_data.c

Her ses en del af funktionen 'collectDataLoop'. Den første del sørger for at loopet først starter når timen er rund. Et print statement fortæller hvor lang tid der er til den næste time.
Til dette bruges et while loop med en sleep funktion. Uden sleep funktionen ville computeren køre loopet med 100% hastighed, og det ville være spild af ressourcer. I stedet ventes et millisekund før der igen tjekkes om den næste time er nået. Det vil sige at programmet har en tidsmæssig nøjagtighed på 1 millisekund.
Når programmet kommer ud af loopet sættes startiden, som også bruger til at lave tidsstempler i CSV-filen, og så laves et printstatement som siger at nu starter data indsamlingen.

Det næste loop kører for evigt, eller indtil programmet bliver stoppet manuelt. Det ses desuden at der er to clock_t variabler "start" og "end". Disse variabler er med til at sørge for at der ikke kommer forsinkelser i programmet, og at de altid er synkroniseret. På den måde sikres at alle tidsstempler er præcis på sekundet.

```c
        if (count == SAMPLE_COUNT) {
            fprintf(data_file, "%ld,%3.1f,%3.1f,%3.1f,%3.1f\n",
                (long)startUTC,
                getAverage(pm2_5_samples, SAMPLE_COUNT),
                getAverage(pm10_samples, SAMPLE_COUNT),
                getAverage(no2_samples, SAMPLE_COUNT),
                getAverage(temp_samples, SAMPLE_COUNT));
            fflush(data_file);
            startUTC = time(NULL);
            count = 0;  // Reset count for next batch
        }
```

Kode Blok 22: Fra collect_data.c

Inde i selve loopet ses at hver gang der er 60 samples, hvilket altid vil være efter præcis 1 time, så skrives gennemsnittet ud sammen med tidsstemplet ved at bruge "fflush". Hvad der ikke kan ses i denne kode blok er, at hvis der sker en fejl under aflæsning af en måling fra device filen, så bliver den markeret med −1, og så bliver den måling ikke talt med af getAverage funktionen. På den måde sørger programmet for at det altid er synkront, samtidig med at der undgås fejlmålinger.

8 Testing
I dette afsnit gennemgås hvordan programmet er blevet testet både i forhold til at finde bugs, men også for at forbedre brugeroplevelsen.

8.1 Testing af data opsamling
Data opsamlingen er en volatil del af systemet, da den kræver kommunikation mellem flere enheder, som ikke nødvendigvis er lavet til at være kompatible med hinanden. Mere specifikt kræver det kommunikation mellem en Arduino, elektriske sensorer og en computer.

Det er generelt et vanskeligt program at teste af flere grunde. For det første afhænger det af funktioner fra flere andre biblioteker, og det er en risiko i sig selv, hvis funktionerne ikke virker som forventet. For det andet skal programmet opsamle data præcis en gang i timen, og der skal sørges for at programmet konstant er synkroniseret, og at den ikke bliver forsinket efter f.eks. 1000 opsamlinger. Dette blev testet ved at sætte programmet til at lave en opsamling efter hvert minut, og efter et par hundrede opsamlinger var det stadig synkroniseret med præcise unix tider. Derefter blev programmet testet ved at lade det opsamle data efter hver time, ligesom det var beregnet til. Efter at have kørt i to døgn, og lavet mere end 50 opsamlinger var programmet stadig synkront, og resultaterne var som forventet. Det kan ses fordi der er 3600 sekunder (1 time) i mellem hver opsamling, og temperaturen ligger ved stuetemperatur, hvilket giver mening efter temperaturen blev målt indenfor. se bilag 3 (Afsnit 12.3), der indeholder testdata fra CSV-filen.

8.2 Brugertest af applikationen
Til testning af programmets brugerflade blev der anvendt to forskellige brugerteste hvor formålet var at observere hvordan brugere brugte programmet, samt at finde ud af hvordan deres oplevelse af det var. Den første brugertest ustruktureret og fungerede ved at give programmet til vilkårlige personer og få dem til at gøre hvad end de havde lyst til i programmet. Dette belyste et par åbenlyse fejl, men det blev hurtigt gjort klart, at metoden ikke var en god tilgang, da testpersonerne var usikre på hvad de skulle gøre, og indtastede bare tilfælge kommandoer ind uden noget mål. Udover det, var det også svært at få dem til at gøre ting på en måde der simulerede hvordan en rigtig bruger ville anvende programmet, da instruktøren var nødt til konstant at guide dem.

På baggrund af denne suboptimale test, blev det besluttet også at lave en mere struktureret brugertest. Denne test bestod af tre forskellige testpersoner. Testpersonerne var alle borgere i København, og blev udvalgt tilfældigt blandt individer der var til rådighed.

Til dette nye forsøg på en forbedret brugertest blev der udviklet en række simple opgaver som testpersonen skulle udfører. Disse opgaver blev givet i tekstform som testpersonen kunne læse. Opgavernes formål var at sørge for at

testpersonerne fik testet alle dele af programmet på en mere naturlig måde, uden at instruktøren direkte skulle instruere dem. Et eksempel på en opgave kunne være at finde data for en specifik lokation og tidspunkt. En fuld liste af opgaver kan ses i Bilag 2, (Afsnit 12.2).

Testpersonerne skulle så vidt muligt forsøge at gennemføre opgaverne uden hjælp fra instruktøren, men hvis testpersonen sad fast et sted i en længere periode ville instruktøren hjælpe dem så minimalt som muligt. Alle tilfælde af hjælp blev noteret som potentielle problemer i programmet.

Efter testens færdiggørelse blev testpersonen bedt om at svare på et spørgeskemaet set i Bilag 1, (Afsnit 12.1). Spørgeskemaets formål var at spørge dem ind til deres overordnede oplevelse af applikationen, og om de havde nogle ønsker til ting kunne tilføjes eller gøres anderledes. Spørgeskemaet bestod af følgende spørgsmål:

- Er det brugervenligt?
- Kan du forstå tabellerne?
- Kan du finde rundt i programmet?
- Hvor nemt er det at finde bestemte tidspunkter?
- Hvordan kan appen blive bedre?

Alt i alt blev der i løbet af testen indsamlet en del information om hvordan applikationen kunne forbedres. Dette inkluderede både fejl i programmets kode, men også sub-optimalt design af brugergrænsefladen. Følgende ses en liste over forskellige fejl der blev fundet under brugertesten:

- Flere testpersoner kommenterede på at det ikke var muligt at forlade menuer og input skærme uden at indtaste et validt input. Et eksempel på dette er den input skærm der vises når brugeren skal indtaste et nyt tidspunkt. Den eneste måde hvorpå brugeren kunne forlade denne skærm er ved at indtaste en valid dato. Dette ledte til en del frustration hvis testpersonen ikke ønskede at ændre datoen og indtastede kommandoen ved en fejl.
- Flere testpersoner oplevede at der manglede en muligheden for at gå tilbage til den senest viste skærm.
- Der blev stillet spørgsmål til grafen og hvad den viste, da der ikke var navn på akserne. Desuden blev det anbefalet at der som note blev beskrevet, at der kunne ses mere præcise værdier for specifikke tidspunkter på dataskærmen.
- Der blev også fundet en række stavefejl og misvisende sætninger blandt andet på timespan-menuen, hvilke skabte en smule forvirring blandt nogle testpersoner.

Der blev også bekræftet mange positive aspekter af programmet. Blandt andet blev det bevist at appen funktionelt fungerer og at det er muligt at tilgå de forskellige data, og at det overordnede design er velfungerende udover de fejl der blev fundet.

9 Diskussion

Der er i dette projekt blevet set ind til et globalt problem, der bidrager til miljømæssig og sundhedsskadelige komplikationer. Der mistes liv, og livskvaliteter nedsættes på grund af problemet. Der er gennem projektarbejdet fundet frem til en løsning til at udarbejde og fremvise data om luftforurening niveauer.

9.1 Tidsbegrænsning

Til dette projekt har der været en tidsbegrænsning på 2,5 måneder. Dette har gjort, at programmet, som er udviklet under projektet, har en del manglende funktioner. Som beskrevet i Afsnit 6.1.3, er der flere funktioner, som ikke er implementeret, men som kunne være implementeret hvis der havde været tid. I forbindelse med testen, som blev lavet sent i projektet, blev det klart, at der manglede en funktion. Denne funktion har til formål at gå tilbage i programmet, Afsnit 8, men er ikke blevet implementeret pga. tidsbegrænsningen. Med mere tid kunne programmet være mere fyldestgørende, og tilpasset til den enkelte bruger.

9.2 Måles de rigtige luftforurenende stoffer?

Siden en essentiel del af produktet er måling af luftforurening, er det relevant at kigge på hvilke sensorer der er brugt i forbindelse med projektet. Som beskrevet i Afsnit 7.2.1, har der været problemer med leveringen af sensorerne, hvilket har gjort, at setuppet ikke er i stand til at måle hverken PM_{10}, $PM_{2.5}$ eller NO_2. Det der kan måles er dog temperatur, se evt. afsnit Afsnit 9.4.

En af de ting der er blevet valgt ikke at måle specifikt på, er “black carbon”, BC, som er en måling på sodpartikler i luften. Når der måles på PM2.5 er en af de inkluderede stoffer BC. Siden projektet her ikke har til intention at reducerer specifikke stoffer, men blot at give indblik i hvorvidt et område har generel forhøjet forurening er det ikke relevant at måle specifikt på BC. Hvis målet med projektet var at reducerer luftforurening, ville det være interessant at kigge på specifikke stoffer, for at kunne arbejde specifikt på at reducerer dem.

Det er i projektet også valgt ikke at fokuserer på CO_2 siden dette ikke har en direkte effekt på menneskets helbred, såfremt der ikke udsættes for store mængder. Hvilket også er grunden til at bl.a. WHO ikke inkluderer CO_2 i anbefalinger for luftforurening.

Ozon, O_3, er også en relevant faktor for direkte helbredseffekter, se evt. Afsnit 2.2.2, men er ikke medtaget, da en sensor for specifikt ozon er dyr.

9.3 Problemet med sensorer

Som tidligere nævnt i Afsnit 7, så der problemer relateret med indkøbet af de to sensorer. Den ene sensor, SEN0441, til at måling af nitrogenoxider, er intenderet højere koncentrationer af nitrogenoxider, og ikke de, som befinder sig i et alment miljø, men i højere grad i led af industrielle processor. Dette betyder at sensoren

ikke ville kunne anvendes under projektarbejdet. Den anden sensor, SEN54-SDN-T, til måling af fine partikler, altså PM2.5 og PM_{10} er ikke blevet leveret under den tidsrammen for projektet. Den effekt disse forsinkelser og uanvendeligheder har haft for projektet er at der ikke ville kunne måles på de stoffer som fremgår i programmets visualiseringen ved søjlediagrammet. Derfor vil der anvendes en temperatursensor til at få proof of concept til at arduinoen ville kunne anskaffe data via en temperatursensor, og lagre det i en fil. Men er det egentlig det samme som hvis der var en sensor som måler på luftforurenende stoffer? Da temperatur også er relevant, som nævnt Afsnit 9.4 når der snakkes om luftforurening, så er det relevant, og en måler til luftforurenende stoffer ville virke på samme måde ved at levere en eller anden data til arduinoen som lagre data.

9.4 Forholde mellem temperatur og luftforureningen

så luftforureningen har et interessant for til temperatur fordi den måde influencer luftforureningen og temperaturen har med hinanden, et eksempel er drivhusgasser som gøre at varmen var solen ikke kan kommer ud af jorden som så gøre det varmer, og med at temperaturet stiger, stiger chancen for ting som skov brand skær som gøre at der kommer med luftforureningen, og et andet eksempel er at temperaturen kontroller vind, ved at lave lav-tryk og høj-tryk zoner, vind bevæger så luftforureningen rundt i luften og det vil så spade luftforureningen mere rundt eller konkreter mere. (IPCC, 2023).

9.5 Diskussion af testning

I testing af data opsamling blev der opnået nogle gode resultater, men det er ikke nok til at vide med sikkerhed at det ville fungere ligeså godt, hvis den stod i længere tid, og især hvis den stod i et mindre beskyttet miljø. Noget som kunne blive implementeret og testet for at gøre data opsamlingen mere sikker, er automatisk udsmidning af fejlmålinger. Hvis der for eksempel fås en temperaturmåling, som er urealistisk høj eller lav, så får målingen et flag og bliver smidt ud hvis den står alene, da det i så fald kun kan have været en fejlmåling.

Til testen blev der brugt 3 testpersoner, som blev observeret af en instruktør, som kunne give vejledning i nødstilfælde. Det kan diskuteres om dette er en tilstrækkelig mængde testpersoner. Eftersom denne type test tager en betydelig mængde tid i forhold til mere kvantitative test-metoder. Dog er kvaliteten af resultaterne også bedre, hvilket betyder at en stor mængde af resultater ikke er lige så nødvendigt.

I testen, se evt. Afsnit 8, blev det afklaret at der var en række problemer med programmet, som gjorde det mindre intuitivt og forværrede brugerens oplevelse af programmet. En af problemerne der blev fundet var stavefejl og forvirrende sætninger. Dette gjaldt bl.a. på hjælpeskærmen hvor en linje var skrevet på en anden måde end alle de andre, hvilket gjorde det ikke var klart hvad der skulle indtastes for at eksekverer kommandoen til at gå til indtastning af timespan.

Flere testpersoner oplevede også at de manglede en tilbage kommando, til at gå tilbage til den tidligere skærm, hvis det var, de ved en fejl, kom hen på en forkert skærm. Derfor kunne der i fremtiden med fordel implementeres en tilbagefunktion.

Grafens mangel på enheder gjorde du uklart helt præcist hvad det var den vist. Dette er blevet fikset ved den nyeste version, så brugeren får noget ud af grafen.

Desuden var der en bug, som nu er fikset, som gjorde, at brugeren ikke kunne finde hjælpeskærmen når der skulle indtastes en tid eller et timespan. Der kunne heller ikke kommes væk fra denne skærm medmindre der blev indtastet en valid dato, men dette er blevet fikset ved implementeringen af tilbage kommandoen.

Opgaverne som testpersonen skulle fuldfører var i form af en liste af spørgsmål, se Brugertest af applikationen (Afsnit 8.2). Det kan diskuteres hvor gode disse spørgsmål er. Blandt andet er de første tre spørgsmål meget lukkede. Dette resulterede i at størstedelen af svarende var en blanding af "ja", "nej" og "nogenlunde", hvilket ikke giver særlig meget information om hvordan det kan forbedres. Desvære var der en mangel på kvalitetskontrol på spørgeskemaet inden testen, så det var det ikke muligt at forbedre den i tide. Trods dette gav spørgeskemaet stadig en hvis viden om brugerens oplevelse, især det sidste spørgsmål. Der var heller ikke en større negativ effekt på testens overordnede resultater, da opgavedelen af testen gav en stor mængde information. En fuld liste af de forskellige svar til spørgeskemaet kan findes i bilag 1 (Afsnit 12.1).

Brugertesten gav værdifuld indsigt i applikationens svagheder, hvilket har givet mulighed for at videreudvikle den for at forbedre disse dele. Grundet tidspres var det ikke muligt at løse alle problemerne, men mindre ting som stavefejl og forkert information blev hurtigt løst.

9.6 Troværdighed

Da der er forskellige sammenhænge mellem doser og tidsmængder når der snakkes om luftforurenende stoffer, så er det svært at fremstille en reel grænseværdi, som alt efter om den aktuelle værdi er over eller under, skildrer om det er ansvarligt at befinde sig i området, i form af et søjlediagram. Dermed, vil det i det rigtige liv blive meget svært at afgøre om det er ansvarligt eller uansvarligt at befinde sig under specifikke koncentrationer, og der er flertallige faktorer der spiller med til at give en estimering om det er ansvarligt, dette kunne blandt andet være, tiden der anvendes i et luftforurenet miljø, hvor luftforurenet miljøet er, hvilke diagnoser vedkommende har, alder og sundhed. Men det er mere eller mindre umuligt at lave en app som kan tage højde for alle disse faktorer og give et samlet overblik. Hvis der var blevet aflagt mere tid til projektarbejdet, ville det have blevet fokuseret på at arbejde med produktet, så der kunne implementeres at grænseværdierne er skræddersyet til den enkelte bruger, ved at nedsætte grænseværdierne korrekt i relation til hvordan f.eks. WHO ville nedsætte dem for befolkning der er særligt svækkede. Hvis sporet

følges, kunne der også ved programmets start af kørsel, spørge brugeren noget i stil af "særligt svækket?", og lade brugeren besvare ja, nej, så der kun er to tilstande, som programmet tilpasser sig til.

9.7 Støj

I forbindelse med opsamlingen af data er der mulighed for støj, som kan nedsætte kvaliteten af data. Ved støj skal der forstås en uønsket påvirkning af sensorer eller andet udstyr, som resulterer i misvisning af den reelle luftkvalitet.

Nogle sensorer påvirkes af temperatur og luftfugtighed, men denne fejlkilde kan bekæmpes ved kalibrering af sensoren. Elektrisk interferens er anden mulig støjkilde som særligt vil blive mere relevant jo mere der tilføjes til det elektriske system.

Siden luftkvaliteten er afhængig af temperaturen, kan dette også betragtes som en støj kilde. Se evt. Afsnit 9.4

Det er ikke undersøgt hvorvidt sensorerne beskrevet i Afsnit 7.2.1 har indbygget beskyttelse af støj, og om Københavns Kommune har håndteret støj ved udgivelsen af deres data, men dette ville tildeles også kunne håndteres i programmet beskrevet i projektet her.

I det, det er støjens natur at påvirke resultater, til at give forkerte værdier, og det er svært definitivt sige at alt støj er elimineret, kunne en funktion udvikles til at håndterer værdier som ligger uden for bestemte grænseværdier for luftkvalitet.

9.8 Videreudvikling

Hvis der i fremtiden skulle videreudvikles på produktet er der samtlige punkter der kunne kigges på. Der kunne laves en mobilapp med grafisk brugerflade ud af det terminalbaserede program, da data ville blive mere brugervenligt og lettilgængeligt, desuden kunne der fås flere diagramtyper ved dette, samt advarselsnotifikationer til høje koncentrationer der overstiger grænseværdien ville kunne advare brugere.

I øjeblikket er data fra målinger fra et enkelt af nogle meget udspredte punkter. I fremtiden kunne der tænkes, at der kunne måles fra flere punkter, så der eventuelt kunne laves et heatmap, således at der grafisk ville kunne ses forskellen og koncentrationen på luftforurening over et bestemt områder som er delt op i tern, dette ville f.eks. kunne bruges til, på længere sigt at vide hvilken rute, der kan tage for en mindre eksponering af luftforurenende stoffer. Derudover ville det også tilføjes at data fra sensor kan blive tilføjet til eksisterende data.

Endnu en essentiel tilføjelse ville være en mulighed for at skræddersy programmets visuelle egenskaber til brugerens æstetetiske præferencer, dette gøre programmet mere visuelt nydeligt i brugerens øjne, og under nogle omstændigheder, får brugeren til oftere at tjekke luftforureningsniveauet et bestemt sted.

10 Konklusion

Ud fra det valgte tema om luftforurening, blev der udarbejdet en applikation som kan bruges til orientering samt overvågning af luftkvalitet. Udover dette blev en arduino Proof of Concept data indsamler også udviklet.

Der kan konkluderes ud fra Arduino eksperimenterne at det er muligt at indsamle data på den ønskede måde, da den var i stand til korrekt at indsamle data om temperatur, og eksportere dette til en CSV fil af ønsket format. Dog var der desværre ikke mulighed for at indsamle data om de forskellige typer partikler grundet mangel af sensorere og udstyr. Målingen af temperaturen er dog Proof of Concept på at det er muligt, da processen med at indsamle denne information er identisk med indsamling af temperatur data.

Udviklingen af applikationen startede med en designproces som indebar at opstille de krav programmet skulle opfylde. Dernæst blev forskellige mock-ups til programmets design lavet, og ud fra dette kunne applikationen implementeres.

Applikationen endte med at opfylde de nødvendige og ønskede krav. Den fungerede ved at spørge brugeren om en ønsket lokation, og læse den tilsvarende CSV fil. Data fra filen bliver indsat i en 2D matrix, for yderligere brug i programmet. Denne data kan nu bruges til tabeller og grafer i programmet, og brugeren kan vælge tidspunkter samt tidsintervaller efter ønske. Applikationens funktionalitet og brugervenlighed blev testet ved hjælp af en brugertest, hvor en mængde testpersoner fik forskellige opgaver at løse i programmet, imens deres adfærd blev observeret. Udover det blev de også bedt om at udfylde et spørgeskema om deres oplevelse. Brugertesten var hovedsageligt en succes, dog var der plads til forbedring i forbindelse med spørgeskemaet.

Derved kan ydermere kan konkluderes at selv et simpelt terminalbaseret program er udmærket i stand til at informere brugeren omkring de omliggende områders luftforureninger, som de udførte brugertest beviste. Brugertesten påviste også at applikationen er nogenlunde brugervenlig, dog med plads til forbedring. Nogle af problemerne med brugervenlighed blev dog løst og implementeret i den færdige version. Programmet kan klart videreudvikles med udvidet funktionalitet og brugervenlighed, men grundet tidsrestriktioner blev nogen ønskede funktioner udeladt.

11 Referencer

Arduino. (2024,). *Arduino UNO R3*. https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf

Asadi, F. (2023). *Essentials of Arduino boards programming*. Apress.

Bank, W. (2021,). *The Global Health Cost of PM2.5 Air Pollution: A Case for Action Beyond 2021*. https://openknowledge.worldbank.org/server/api/core/bitstreams/550b7a9b-4d1f-5d2f-a439-40692d4eedf3/content

Board, C. air resources. *Inhalable Particulate Matter and Health (PM2.5 and PM10)*. https://ww2.arb.ca.gov/resources/inhalable-particulate-matter-and-health

Butler, S. (2022,). *What is GPIO?*. https://www.howtogeek.com/787928/what-is-gpio/

DFRobot. (2021,). *SKU:SEN0441*. https://wiki.dfrobot.com/Fermion_MEMS_Gas_Sensor__MiCS-2714_SKU_SEN0441

Elextra. *Vishay - NTC termistor*. https://www.elextra.dk/p/ntc-termistor-10kohm-3977k-500mw-40-125c/H58384

European Environment Agency,. (2023,). *Air pollution and children's health*. https://www.eea.europa.eu/publications/air-pollution-and-childrens-health/air-pollution-and-childrens-health

EU's luftkvalitetsdirektiv 2008/50/EF. (2008,). https://eur-lex.europa.eu/legal-content/DA/TXT/?uri=CELEX:32008L0050

IPCC. (2023,). *CLIMATE CHANGE 2023 Synthesis Report*. https://www.ipcc.ch/report/ar6/syr/downloads/report/IPCC_AR6_SYR_FullVolume.pdf

ISO. (2024). *Information technology — Programming languages — C* (5. udg.). https://www.iso.org/standard/82075.html

Kane, P. *Temperature Measurement with NTC Thermistors*. https://www.jameco.com/Jameco/workshop/TechTip/temperature-measurement-ntc-thermistors.html

Københavns Kommune,. (2020,). *monitorering af luftkvaliteten*. https://www.kk.dk/borger/affald-og-miljoe/stoej-stoev-og-luft/luftforurening/monitorering-af-luftkvaliteten

Københavns Kommune,. (2022,). *monitorering af luftkvaliteten*. https://www.kk.dk/sites/default/files/2024-02/%C3%85rsrapport%20for%20overv%C3%A5gning%20af%20luftkvalitet%20i%20K%C3%B8benhavns%20Kommune%202022.pdf

Laust, K. (2023,). *AAU Typst template*. https://github.com/krestenlaust/AAU-Typst-Template

RS. *SEN54-SDN-T*. https://dk.rs-online.com/web/p/miljosensor-ic-er/2389413

Technologies, B. (2021,). *How to convert between mg/m3, µg/m3 and ppm, ppb*. https://www.breeze-technologies.de/blog/air-pollution-how-to-convert-between-mgm3-%C2%B5gm3-ppm-ppb/

WMO. *WMO Bulletin: heatwaves worsen air quality and pollution*. https://wmo.int/news/media-centre/wmo-bulletin-heatwaves-worsen-air-quality-and-pollution

World Health Organization,. (2021,). *Who global air quality guidelines: particulate matter (PM2.5 and PM10), ozone, nitrogen dioxide, sulfur dioxide and carbon monoxide*. https://www.who.int/publications/i/item/9789240034228

Århus Universitet,. (2024,). *luftforurening udledninger og effekt*. https://envs.au.dk/om-instituttet-1/faglige-omraader/luftforurening-udledninger-og-effekter/overvaagningsprogrammet

12 Bilag

12.1 Svar på spørgeskema

Spørgeskema

Testperson/ spørgsmål	Er det bruger venlig?	Kan du forstå tablerne?	Kan du finde rundt i programmet?	Hvor nemt er det at finde bestemt tidspunkter?	Hvordan kan Appen blive brede?
Testperson 1	nogenlunde	nogenlunde	Ja og nej den der timespan er lidt svær at forstå	Ja og nej den der timespan er lidt svær at forstå	• Visualisering af graf er ikke præcis, for præcise værdier se specifikke værdier • Mangler enhed på graf • Rettelse af P2.5 på graf • Kan ikke bruge hjælp inde på time og timespan • Ret xxxx-xx-xx - xxxx-xx-xx i timespan • Kan ikke gå ud fra time og timespan uden at skrive en hel dato • Ret span til s i hjælp
Testperson 2	Ja	De giver mening, man skal bare bære i mende at man			• Ret xxxx-xx-xx - xxxx-xx-xx i timespan • Tilbage funktioin
Testperson 3	ja	ja	ja	ja	• Ændrer data til date

12.2 Spørgsmål til spørgeskema

Opgaver

- Display hjælpeskærmen
- Hvad var koncentrationen af PM_{10} 2. maj 2024 kl. 11 på Folehaven
- Vis for Beckersvej en graf over $PM_{2.5}$ i tidsperioden 1. Jan 2024 til 15. Jan 2024
- Afslut programmet

12.3 Resultater fra dataopsamling

csv

```
StartUtc,PM2_5,PM10,NO2,Temp
1732449600,,,,20.4
1732453200,,,,20.7
1732456800,,,,20.8
1732460400,,,,20.6
1732464000,,,,20.5
1732467600,,,,20.5
1732471200,,,,19.2
1732474800,,,,19.1
1732478400,,,,19.1
1732482000,,,,19.1
1732485600,,,,19.1
1732489200,,,,19.1
1732492800,,,,18.6
1732496400,,,,18.6
1732500000,,,,18.6
```

```
1732503600,,,,18.6
1732507200,,,,18.6
1732510800,,,,18.6
1732514400,,,,18.8
1732518000,,,,18.8
1732521600,,,,18.8
1732525200,,,,18.8
1732528800,,,,18.8
1732532400,,,,18.8
1732536000,,,,18.7
1732539600,,,,18.7
1732543200,,,,18.7
1732546800,,,,18.7
1732550400,,,,18.7
1732554000,,,,19.1
1732557600,,,,20.5
1732561200,,,,20.5
1732564800,,,,20.5
1732568400,,,,20.5
1732572000,,,,20.5
1732575600,,,,20.9
1732579200,,,,21.8
1732582800,,,,21.8
1732586400,,,,21.8
1732590000,,,,21.8
1732593600,,,,21.8
1732597200,,,,21.3
1732600800,,,,20.4
1732604400,,,,20.4
1732608000,,,,20.4
1732611600,,,,20.4
1732615200,,,,20.4
1732618800,,,,20.5
1732622400,,,,20.7
1732626000,,,,20.7
1732629600,,,,20.7
1732633200,,,,20.7
1732636800,,,,20.7
```