



SW2PRJ2 Projekt

Energistyring og embeddede systemer til overvågning/fjernstyring

Gruppe 16

Navn	Studienummer	Studieretning
Daniel Naddaf	Studie Nr. 202106189	SW
Jonas Kirkegaard Skyum	Studie Nr. 202306414	SW
Khaled Rami Omar	Studie Nr. 202307853	SW
Simon Eifer	Studie Nr. 202306749	SW
Filip Allan Alberg	Studie Nr. 202307156	SW
Jakob Worm Albertsen	Studie Nr. 202305906	SW
Nichlas Vu	Studie Nr. 202305371	SW

Afleveringsdato: 31/05/2024.
Vejleder: Kristian Nybye Lund.
Medbedømmer: Torben Gregersen.

Omfang: 66536 anslag inkl. mellemrum.

1 Resumé/Abstract

1.1 Resumé

Denne rapport dokumenterer udfærdigelsen af et temperatur reguleringssystem. Systemet kan regulere temperaturen i et lokale baseret på brugerens ønskede temperatur og et prisloft for elprisen under opvarmningen. Systemet bruger temperaturen i lokalelet, elpriser og brugerindstillede tærskelværdier til at bestemme reguleringstidspunktet. Systemet skulle overvåge hjemmets aktuelle energiforbrug, for at medtage dette i dets overvejelser. Dette essentielle krav til systemet blev ikke opfyldt, da kommunikationen mellem de ansvarlige delsystemer ikke blev etableret. Under bestemmelsen af reguleringstidspunktet, kan den bedste løsning for programmet være, at vente en eller to timer, til elprisen er faldet. Der har været problemer med at få programmet til at vente den eksakte tid, så denne del af regulerings-algoritmen er utilstrækkelig.

1.2 Abstract

This report documents the development of a temperature control system. The system can regulate the temperature in a room based on the user's desired temperature and a price cap for electricity during heating. The system uses the room temperature, electricity prices, and user-set thresholds to determine the regulation timing. The system was supposed to monitor the home's current energy consumption to include this in its considerations. This essential requirement for the system was not met because communication between the responsible subsystems was not established. When determining the regulation timing, the best solution for the program might be to wait one or two hours until the electricity price has dropped. There have been issues with getting the program to wait the exact time, so this part of the regulation algorithm is insufficient.

Indhold

1 Resumé/Abstract	2
1.1 Resumé	2
1.2 Abstract	2
2 Indledning	6
2.1 Problemformulering	6
2.2 Elementer	7
2.2.1 Hub	7
2.2.2 Arduino mega2560	7
2.2.3 Temperatur sensor	7
2.2.4 Lys sensor	7
2.2.5 Dataopsamling	7
2.2.6 Temperatur regulator	7
2.2.7 ”Smart”kontakt	7
3 Kravspecifikation	8
3.1 Funktionelle krav	8
3.2 Ikke-funktionelle krav	17
3.2.1 Temperaturregulering	17
3.2.2 Temperatursensor	17
3.2.3 Varmeaktuator	17
3.2.4 Hub	17
3.2.5 Smartkontakt	17
3.2.6 Touchskærm	17
3.2.7 Energimåler	18
4 Accepttestspezifikation	18
4.1 Funktionelle krav	18
4.2 Ikke-funktionelle krav	22
5 Metode og Proces	24
5.1 Proces	24
6 Systemarkitektur	25
6.1 Hardware arkitektur	25
6.1.1 BDD	25
6.1.2 IBD	27
6.1.3 Note om reducering af projektets størrelse	30
6.2 Software arkitektur	33
6.2.1 Domænemodel	33
6.2.2 Sekvensdiagram	33
6.2.3 State machine diagram	34
6.2.4 Protokoller	35

7 Systemdesign	36
7.1 Energy consumption calculator	36
7.1.1 Hardware design	36
7.1.2 Realisering	37
7.1.3 Software design	38
7.2 Temperatur regulering	39
7.2.1 Hardware design	39
7.2.2 Software design	41
7.3 Hub	45
7.3.1 Software design	45
7.3.2 Hardware design	51
8 Implementering	54
8.1 Temperatur regulering	54
8.1.1 I2C/TWI kommunikation	55
8.1.2 Reflektering over implementering	55
8.2 HUB	55
8.2.1 Frontend:	55
8.2.2 API	63
8.2.3 Backend	64
8.2.4 Raspberry Pi 4 og Raspberry 7" Touchscreen	65
8.3 ECC / Elmåler	65
8.3.1 Software implementation	65
8.3.2 Hardware implementation	66
9 Test	66
9.1 Modultest	66
9.1.1 Temperature regulation	66
9.1.2 HUB	67
9.1.3 ECC / Elmåler	68
9.2 Integrationstest	69
9.3 Acceptttest	69
9.3.1 Noter til accepttest	74
10 Resultater	74
10.1 Temperature Regulation	74
10.2 HUB	74
10.3 Energy Consumption Calculator	74
10.4 Hele systemet	74
11 Diskussion af resultater	75
11.1 Temperature Regulation	75
11.2 Hub	75
11.3 Energy Consumption Calculator	76
11.4 Hele systemet	76
12 Konklusion	76

13 Fremtidigt arbejde	77
13.1 UART Kommunikation	77
14 Kildeliste	77
15 Bilagsoversigt	79

2 Indledning

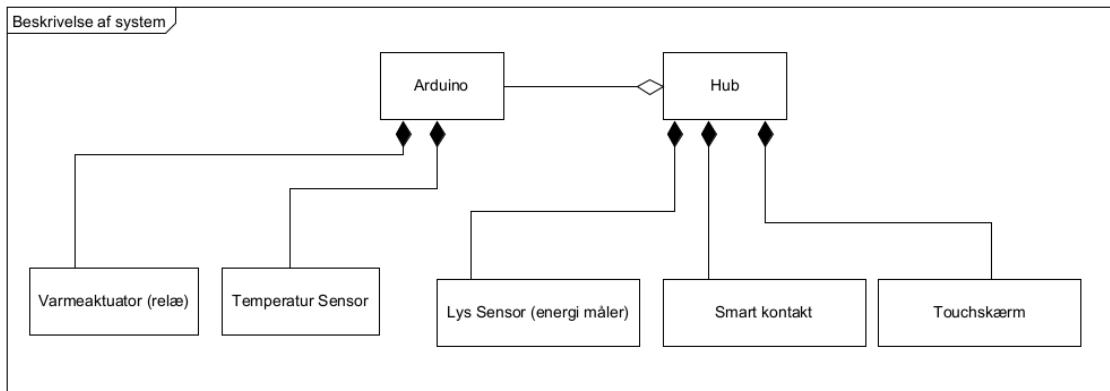
af hele gruppen

2. semsterprojektet på Softwareteknologi-uddannelsen i 2024 er udviklet med fokus på embeddede systemer, som skal fjernstyre og overvåge energistyrings-enheder som aktuatorer og sensorer. Denne rapport skal klargøre processen bag udførslen af dette projekt, fra idé til realisering og test. Der er i dette projekt valgt at fokusere på opvarmning samt vedligeholdelse af temperaturen i et hjem, som opvarmes med el. Elektrisk opvarmning er lettere at gå til uden forhåndsviden, da implementeringen af aktuatorer i et varmelegeme, som f.eks. sidder i en stikkontakt, virker mere håndgribeligt end alternativerne.

2.1 Problemformulering

Projektet tager sit udgangspunkt i optimeringen af energi- og varmeforbruget i et hjem, lokale, eller sommerhus, som anvender elektrisk varme. En hub placeres centralt i hjemmet og modtager hjemmetes temperatur samt energiforbrug, fra hhv. en temperatur- og lys-sensor (impulssensor på elmåler). Dertil skal hub'en indhente den aktuelle energipris, som skal kunne skaleres, så den passer med hjemmetes el-abonnement (f.eks. tillæg fra leverandøren). Brugeren skal på denne hub kunne indtaste den ønskede temperatur for hjemmet, samt en øvre grænse for elprisen, som de vil betale, for at opvarme hjemmet. Ud fra dette vil hub'en tænde og slukke for hjemmetes varmelegeme på de mest fordelagtige tidspunkter, for at spare på strøm, samt vedligeholde den ønskede temperatur. Ved høje energipriser og lave temperaturer, vil brugeren skulle tage en beslutning om hvad der er vigtigst for dem. Systemet skal derfor implementeres med prioriterings funktioner, og eventuelt data, der kan hjælpe brugeren med at træffe den rigtige beslutning.

Brugeren kan også vælge at energioptimere en almindelig stikkontakt i hjemmet, sådan at denne kun tænder, indenfor et givet interval af energipriser. Denne smartkontakt vil tilmed kunne indstilles med en given tidsperiode (f.eks. den tid, det tager at oplade et batteri), og ud fra dette, stoppe helt med at tænde, når den samlet set har været tændt i den givne periode.



Figur 1: Beskrivelse af systemet.

2.2 Elementer

Systemet vil bestå af en hub, som, sammen med en arduino, modtager og sender signaler til resten af systemet. Dertil skal der anvendes en temperaturmåler, som snakker sammen med arduinoen, der regulerer temperaturen. Hub'en skal kunne vise data om energipriser fra nettet, samt vise ens eget energiforbrug, som måles med en lys-sensor. Alle komponenter og elementer er beskrevet i dette afsnit.

2.2.1 Hub

En (touch)skærm med et interface, som viser temperaturen i hjemmets områder, temperaturen udenfor (fra vejrudsigtten), el priserne og strømforbruget. Denne implementeres med en Raspberry Pi, som modtager input fra brugeren og snakker sammen med resten af systemet - herunder sensorer, aktuatorer og arduino.

2.2.2 Arduino mega2560

Arduino implementeres i form af temperaturreguleringen, hvor dens rolle er at snakke sammen med sensorerne og Raspberry pi. Ud fra informationerne fra disse, vil den sørge for at regulere temperaturen til den ønskede temperatur eller udføre energibesparelsesfunktionen. Dertil anvendes der også en arduino til at modtage signaler fra fotodioden på elmåleren. Denne skal beregne det aktuelle forbrug og periodisk sende det til Raspberry'en.

2.2.3 Temperatur sensor

Der installeres en temperatur sensor i de rum, hvor temperaturen kan reguleres. Disse sender data til en arduino.

2.2.4 Lys sensor

Der installeres en lys-sensor på elmålerens LED. Denne skal aflæse frekvensen på lyset. For at indsamle data om, hvor meget el der bruges.

2.2.5 Dataopsamling

Data indsamles lokalt fra hjemmets elmåler, og sendes til hub'en, sådan at man kan se hjemmets forbrug.

Ekstern dataopsamling af prisdata fra energiværker, som bruges til at optimere energiforbruget i hjemmet.

2.2.6 Temperatur regulator

En aktuator i form af en kontakt, som tændes og slukkes alt efter hvad temperaturen i hjemmet er. Denne får sin ordre fra arduinoen, som også holder øje med temperaturen.

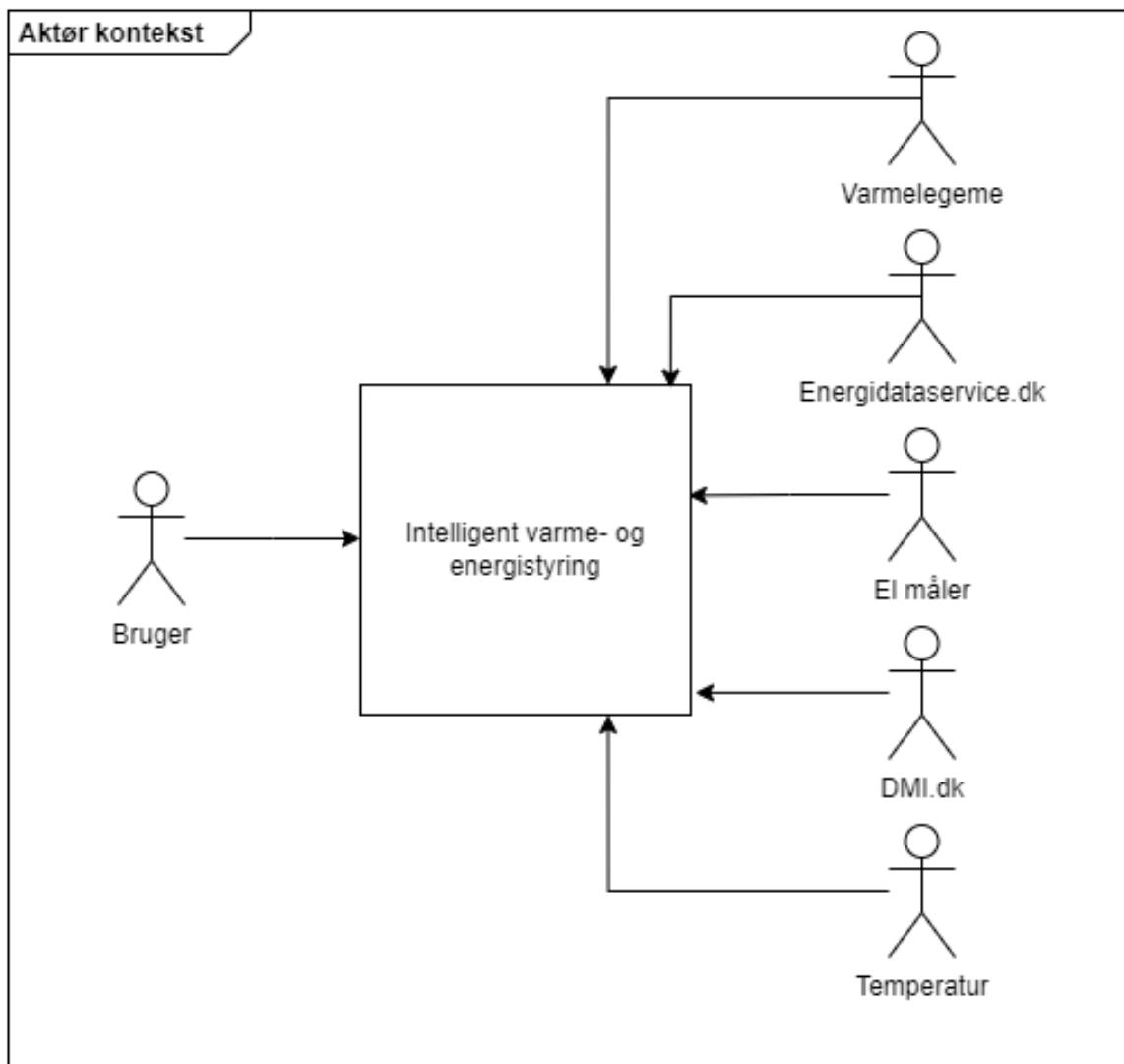
2.2.7 "Smart" kontakt

En aktuator i form af en kontakt, som tændes og slukkes alt efter hvad elpriserne er. Denne får sin ordre fra HUB'ens raspberry pi, som modtager info om el-priser og -forbrug.

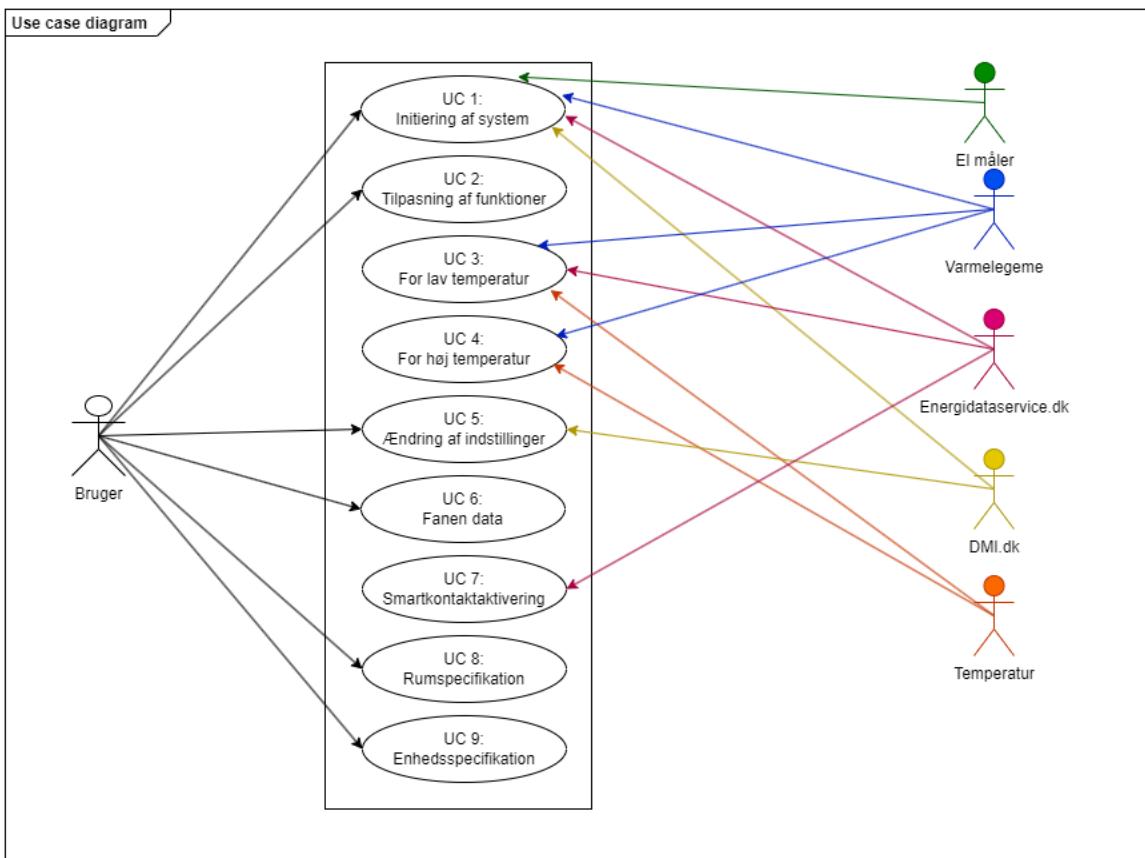
3 Kravspecifikation

af hele gruppen

3.1 Funktionelle krav



Figur 2: Aktør-kontekst diagram for hele systemet.



Figur 3: Use case diagram for hele systemet.

Navn	Initiering af system.
ID	1
Mål	At indstille systemet med brugerens præferencer og informationer.
Initiering	- Brugeren tænder hub'en for første gang.
Aktører	Brugeren, El-måler, Varmelegeme, Energidataservice.dk, dmi.dk
Antal samtidige forekomster	1.
Forudsætninger	- Systemet er forbundet strøm. - Alle komponenter er operationsdygtige.
Resultat	Klargøre systemet til standard tilstand.
Hovedscenarie	<ol style="list-style-type: none"> 1. Brugeren tilslutter eksterne sensor/aktuator systemer. 2. Brugeren opretter forbindelse til netværk. 3. Brugeren indtaster informationer iht. husstandens elleverenadør. <ol style="list-style-type: none"> (a) EXT1: Brugeren springer handlingen over. 4. Brugeren indtaster deres ønskede temperatur for hjemmet, eller for de individuelle rum. <ol style="list-style-type: none"> (a) EXT1: Brugeren springer handlingen over. 5. Brugeren indtaster sit postnummer. <ol style="list-style-type: none"> (a) EXT1: Brugeren springer handlingen over. 6. Brugeren godkender sine valgte præferencer og indstillinger. <ol style="list-style-type: none"> (a) EXT2: Brugeren godkender ikke.
Udvidelser/undtagelser	<ul style="list-style-type: none"> - EXT1: Brugeren springer handlingen over; Der sker ikke noget. Brugeren går videre til næste indstilling. - EXT2: Brugeren godkender ikke; Systemet nulstiller og brugeren gennemgår indstillerne igen.

Tabel 1: Use case 1

Navn	Tilpasning af funktioner
ID	2
Mål	At tilpasse systemets funktioner efter brugerens behov
Initiering	Brugeren tilgår fanen ”funktioner”.
Aktører	Brugeren
Antal samtidige forekomster	1
Forudsætninger	- Systemet er operationsdygtigt.
Resultat	Systemets funktioner er tilpasset brugerens behov.
Hovedscenarie	<ol style="list-style-type: none"> 1. Brugeren vælger funktionen der skal tilpasses. 2. Brugeren trykker på ”Tilpas funktion” 3. Brugeren tilpasser parameterene for funktionen. <ol style="list-style-type: none"> (a) EXT: Brugeren trykker på ”tilbage” 4. Brugeren gemmer den tilpassede funktion. 5. Skærmen viser oversigten over funktionen.
Udvidelser/undtagelser	EXT: Brugeren trykker ”tilbage”; Ingen ændringer gemmes. Brugeren kommer tilbage til den forrige fane i systemet.

Tabel 2: Use case 2

Navn	Temperatur regulering - For lav temperatur
ID	3
Mål	At regulere temperaturen, baseret på brugerens indstillinger
Initiering	Temperaturen falder til under den ønskede værdi.
Aktører	Energidataservice.dk, varmelegeme, temperatur
Antal samtidige forekomster	1
Forudsætninger	- Systemet er operationsdygtig. - Temperaturen er under det givet interval
Resultat	Temperaturen er reguleret til det ønskede.
Hovedscenarie	<ol style="list-style-type: none"> 1. Elprisen er under det acceptable prisloft. <ol style="list-style-type: none"> (a) El prisen er over det acceptable prisloft. 2. Varmelegeme tændes. 3. Temperaturen når den ønskede værdi 4. Varmelegeme slukkes.
Udvidelser/undtagelser	EXT: Prioriterings funktion træder i kraft.

Tabel 3: Use case 3

Navn	Temperatur regulering - For høj temperatur
ID	4
Mål	At regulere temperaturen, baseret på brugerens indstillinger
Initiering	Temperaturen overstiger det ønskede interval.
Aktører	Varmelegeme, temperatur
Antal samtidige forekomster	1
Forudsætninger	<ul style="list-style-type: none"> - Systemet er operationsdygtig. - Temperaturen er over det givet interval
Resultat	Temperaturen er reguleret til det ønskede.
Hovedscenarie	<ol style="list-style-type: none"> 1. Varmelegeme slukkes.
Udvidelser/undtagelser	

Tabel 4: Use case 4

Navn	Ændring af indstillinger.
ID	5
Mål	At indstille systemet med brugerens præferencer og informationer.
Initiering	Brugeren trykker på feltet ”indstillinger” på hub’ens startside.
Aktører	Brugeren
Antal samtidige forekomster	1.
Forudsætninger	- Hub’en er operationsdygtig.
Resultat	Indstille systemet.
Hovedscenarie	<ol style="list-style-type: none"> 1. Hub’en viser fanen ”Indstillinger”. 2. Brugeren opretter forbindelse til netværk. <ul style="list-style-type: none"> (a) EXT1: Brugeren foretager ingen handling ved feltet. (b) EXT2: Brugeren trykker ”tilbage”. 3. Brugeren indtaster informationer iht. husstandens leverenadør. <ul style="list-style-type: none"> (a) EXT1: Brugeren foretager ingen handling ved feltet. (b) EXT2: Brugeren trykker ”tilbage”. 4. Brugeren indtaster sit postnummer. <ul style="list-style-type: none"> (a) EXT1: Brugeren foretager ingen handling ved feltet. (b) EXT2: Brugeren trykker ”tilbage”. 5. Brugeren godkender sine valgte præferencer og indstillinger. <ul style="list-style-type: none"> (a) EXT2: Brugeren trykker ”tilbage”. (b) EXT3: Brugeren godkender ikke.
Udvidelser/undtagelser	<ul style="list-style-type: none"> - EXT1: Brugeren foretager ingen handling ved feltet; Der sker ikke noget. - EXT2: Brugeren trykker tilbage; Brugeren går tilbage til startside. - EXT3: Brugeren godkender ikke; Indstillinger gemmes ikke. Brugeren går tilbage til startside.

Tabel 5: Use case 5

Navn	Fanen data
ID	6
Mål	At give brugeren et overblik over indsamlet data
Initiering	Brugeren trykker på feltet "Data" på hub'ens startside.
Aktører	Brugeren
Antal samtidige forekomster	1
Forudsætninger	- Hub'en er operationsdygtig
Resultat	Vise brugeren den indsamlede data
Hovedscenarie	<ol style="list-style-type: none"> 1. Fanen data vises på hub'en. 2. Brugeren trykker "tilbage".
Udvidelser/undtagelser	

Tabel 6: Use case 6

Navn	Smart kontakt aktivering
ID	7
Mål	At aktivere smartkontakten, baseret på brugerens indstillinger for elpriser.
Initiering	Elprisen er under brugerens valgte prisloft.
Aktører	Energidataservice.dk
Antal samtidige forekomster	1
Forudsætninger	Systemet er operationsdygtigt.
Resultat	Smartkontakten leder strøm til dens tilsluttede enhed.
Hovedscenarie	<ol style="list-style-type: none"> 1. Kontaktet tænder. 2. Der løber strøm til kontaktens tilsluttede enhed. <ol style="list-style-type: none"> (a) EXT: Elprisen er over prisloftet.
Udvidelser/undtagelser	EXT: Elprisen er over prisloftet: Kontakt slukker.

Tabel 7: Use case 7

Navn	Rum Specifikation
ID	8
Mål	Indstille specifikationer for de individuelle. rum
Initiering	Brugerens tilgårs fanen ”Rum”
Aktører	Brugerens
Antal samtidige forekomster	1
Forudsætninger	- Brugerens har valgt fanen rum - Systemet er operationsdygtigt
Resultat	Temperaturen justeres baseret på den oplyste data.
Hovedscenarie	<ol style="list-style-type: none"> 1. Brugerens klikker på et specifikt rum <ol style="list-style-type: none"> (a) EXT1: Brugerens annullerer og går til startside. 2. Brugerens indtaster data for prisloft og temperatur. <ol style="list-style-type: none"> (a) EXT2: Brugerens indtaster for prisloft men ikke temperatur. (b) EXT3: Brugerens indtaster for temperatur men ikke pris. (c) EXT4: Brugerens vælger en funktion med bestemte værdier på forhånd. (d) EXT5: Brugerens annullerer og går til startside.
Udvidelser/undtagelser	<ul style="list-style-type: none"> - EXT1: Brugerens klikker annullerer; Går tilbage til startside. - EXT4: Brugerens vælger en funktion; Data fra funktionen sættes til den bestemte rum - EXT5: Brugerens annullerer og går tilbage til starten.

Tabel 8: Use case 8

Navn	Enhedsspecifikation
ID	9
Mål	Indstille enheder for systemet
Initiering	Når fanen ”enheder” vælges på forsiden
Aktører	Brugeren
Antal samtidige forekomster	1
Forudsætninger	- Brugeren på valgt fanen enheder - Systemet er operationsdygtigt
Resultat	Styrer de forskellige enheder i systemet.
Hovedscenarie	<ol style="list-style-type: none"> 1. Brugeren vælger en specifik enhed. <ol style="list-style-type: none"> (a) EXT1: Brugeren annullere og går til startside. 2. Brugeren justerer data for prisloft til den specifikke enhed. <ol style="list-style-type: none"> (a) EXT2: Brugeren indtaster ikke prisloft. (b) EXT3: Brugeren annullere og går til startside.
Udvidelser/undtagelser	<ul style="list-style-type: none"> - EXT1: Brugeren klikker annuller; Går tilbage til startside. - EXT2: Brugeren indtaster ikke prisloft; Der sker ikke noget - EXT3: Brugeren annuller og går tilbage til starten.

Tabel 9: Use case 9

3.2 Ikke-funktionelle krav

3.2.1 Temperaturregulering

- Skal implementeres med en Arduino Mega2560.
- Arduinoen kan anvendes med et shield fra PR Electronics, med påmonteret LM75 temperatursensor.
- Arduinoen bør kommunikere med Hub'en gennem en half-duplex linje.
- Arduinoen skal kunne regulere temperaturen, baseret på et givet temperatur-interval, uden nogen forbindelse til hub'en.
 - Dvs. at temperaturreguleringen skal kunne operere uden brug af hub'en, dvs. udelukkende med arduinoen, hvis elprisen ignoreres.
- Den samlede størrelse af reguleringsenheden, dvs. controller+sensor, må ikke overstige målene LxBxH: 10x5x4 (i centimeter).

3.2.2 Temperatursensor

- Kan implementeres med en LM75 temperatursensor.
- Skal kunne måle temperaturer i intervallet 0°C-30°C.
- Målingerne skal maks have en afvigelse på ±2°C.

3.2.3 Varmeaktuator

- Skal implementeres i form af en MOSFET.

3.2.4 Hub

- Hub'en skal implementeres med en Raspberry Pi 4.
- Hub'en skal kunne kobles til internettet.
 - For at kunne hente data for elpriser, og vise vejrudsigten for brugerens lokalitet.

3.2.5 Smartkontakt

- Smartkontakten skal tænde hvis energiprisen er under det indstillede prisloft i hub'en
- Smartkontakten skal slukke hvis energiprisen er over det indstillede prisloft i hub'en
- Kunne initieres med den samlede tid, den ønskes tændt.
 - F.eks. den tid, det tager at oplade et batteri, som er sluttet til kontakten.

3.2.6 Touchskærm

- Touchskærmen skal implementeres via Raspberry pi 4.
- Bør implementeres med en Raspberry Pi 7" touchskærm.

3.2.7 Energimåler

- Skal kunne måle strømforbrug
- Skal implementeres med en lyspuls-måler der vender ind mod elmåleren i hustanden.
- Lyspuls-måleren skal kunne måle frekvenser i området 10-100Hz.

4 Accepttestspezifikation

af hele gruppen

4.1 Funktionelle krav

Use Case 1: Initiering af system.			
Punkt	Præ-kondition	Action	Forventet resultat
1	Hub slukket. Systemet er operationsdygtigt.	Brugeren trykker på tænd knappen.	Hub'en tænder
2	Sensor/aktuator systemer er tilgængelige og operationsdygtige.	Brugeren tilslutter komponenterne ud fra manual.	Man kan se, at komponenterne er forbundet.
3	Netværksfobindelse tilgængelig.	Brugeren opretter forbindelse til netværket.	Hub'en viser at der er oprettet forbindelse.
4	Brugeren kender til informationer iht. priser fra elleverandør.	Brugeren udfylder pris-informationer om husstandens elleverandør.	Informationer indtastet i Raspberry pi
5	Brugeren kender sin ønskede temperatur for hjemmet (eller området).	Brugeren indtaster en ønsket temperatur for ønsket rum.	Informationer indtastet i Raspberry pi
6	Brugeren kender postnummeret for husstanden, som systemet installeres i.	Brugeren indtaster husstandens postnummer.	Informationer indtastet i Raspberry pi
7	Indstillingerne er korrekt udfyldt.	Brugeren trykker godkendt på hubben.	Indstillinger godkendt i Raspberry pi

Tabel 10: Use Case 1, accepttest

Use Case 2: Tilpasning af funktioner			
Punkt	Præ-kondition	Action	Forventet resultat
1	Systemet er operationsdygtigt.	Brugeren trykker på funktionen, som ønskes ændret.	Fanen for den valgte funktion åbnes
2	Brugeren er tilgået funktionen som skal ændres.	Brugeren trykker ”Tilpas funktion”.	Fanen ”tilpas funktion” åbnes for den valgte funktion
3	Parametre, der kan ændres, vises på skærmen.	Brugeren tilpasser parametrene efter eget behov.	Parametrene bliver ændret og vises på skærmen.
4	Brugeren er tilfreds med sit valg af parametre.	Brugeren trykker gem.	Systemet gemmer indstillinger.
5	De nye indstillinger er gemt.	Systemet sender brugeren til fanen med oversigten over funktionen.	Skærmen viser oversigten over funktionen.

Tabel 11: Use Case 2, accepttest

Use Case 3: Temperatur regulering - For lav temperatur			
Punkt	Præ-kondition	Action	Forventet resultat
1	Systemet har adgang til elpriser.	Tjekker elpris.	Elpris under prisloft.
2	Systemets varmeaktuator er operationsdygtigt og forbundet til resten af systemet. Varmelegeme er operationsdygtigt.	Varmeaktuator/ relæ aktiveres	Varmelegeme tændes.
3	Temperatursensor er operationsdygtigt og forbundet til retsen af systemet.	Temperatursensor mäter temperatur.	Temperaturen har den ønskede værdi.
4	Systemets varmeaktuator er operationsdygtigt og forbundet til resten af systemet. Varmelegeme er operationsdygtigt.	Varmeaktuator/ relæ slukkes	Varmelegeme slukkes.

Tabel 12: Use Case 3, accepttest

Use Case 4: Temperatur regulering- For høj temperatur			
Punkt	Præ-kondition	Action	Forventet resultat
1	Systemets varmeaktuator er operationsdygtigt og forbundet til resten af systemet. Varmelegeme er operationsdygtigt.	Varmeaktuator/ relæ slukkes	Varmelegeme slukkes.

Tabel 13: Use Case 4, accepttest

Use Case 5: Ændring af indstillinger			
Punkt	Præ-kondition	Action	Forventet resultat
1	Systemet er operationsdygtigt. Skærmen er tændt.	Brugeren trykker på fanen ”indstillinger”.	Hub'en viser fanen ”Indstillinger”
2	Netværksforbindelse tilgængelig.	Brugeren trykker på ”Opret forbindelse til internet”, og indskriver data.	Der er oprettet forbindelse til netværk.
3	Brugeren kender til informationer iht. priser fra elleverandør.	Brugeren indtaster informationer iht. elleverandør.	De indtastede informationer gemmes i Raspberry pi og vises på skærmen.
4	Brugeren kender postnummerelet for husstanden, som systemet installeres i.	Brugeren indtaster sit postnummer.	Det indtastede postnummer gemmes i Raspberry pi og vises på skærmen.
5	Indstillerne er korrekt udfyldt	Brugeren trykker godkend på skærmen.	Hub'en viser at indstillerne er godkendt.

Tabel 14: Use Case 5, accepttest

Use Case 6: Fanen data.			
Punkt	Præ-kondition	Action	Forventet resultat
1	Systemet er operationsdygtigt	Systemet åbner fanen data.	Fanen ”Data” vises.
2	Systemet er operationsdygtigt	Brugeren trykker tilbage på touchskærmen.	Systemet går tilbage til startside.

Tabel 15: Use Case 6, accepttest

Use Case 7: Smart kontakt aktivering			
Punkt	Præ-kondition	Action	Forventet resultat
1	Systemet er operationsdygtigt. Smartkontakten er operationsdygtig. Systemet har modtaget data om energipriser	Systemet sender et signal om at kontakten skal tændes.	Den tilsluttede enhed tændes.
2	Systemet får data om elpriser	Systemet slukker/tænder kontakten, udfra energipriserne.	Den tilsluttede enhed slukkes/tændes ikke, hvis over prisloft.

Tabel 16: Use Case 7, accepttest

Use Case 8: Rum Specifikation.			
Punkt	Præ-kondition	Action	Forventet resultat
1	Hub'en viser fanen "rum".	Brugeren Trykker på specifikke rum	Fanen for det specifikke rum åbnes.
2	Brugeren har trykket på indstillinger for det specifikke rum	Data for prisloft og temperatur indtastes for det specifikke rum.	Prisloft og temperatur ændres

Tabel 17: Use Case 8, accepttest

Use Case 9: Enhedsspecifikation.			
Punkt	Præ-kondition	Action	Forventet resultat
1	Hub'en viser fanen "enheder".	Brugeren trykker på den specifikke enhed	Går ind på den specifikke enhed.
2	Hub'en viser fanen for den specifikke enhed.	Brugeren trykker på tilbage-knappen	Går tilbage til startside.
3	Brugeren har trykket på indstillinger for enheden.	Brugeren ændrer prisloftet	Prisloftet for den specifikke enhed er justeret.
4	Brugeren har trykket på indstillinger for enheden.	Brugeren ændrer ikke prisloftet	Prisloftet for den specifikke enhed bliver ikke justeret.
5	Brugeren har trykket på indstillinger for enheden.	Brugeren annullerer indtastning og trykker på tilbage-knappen	Prisloftet bliver ikke ændret, og går tilbage til startside.

Tabel 18: Use Case 9, accepttest

4.2 Ikke-funktionelle krav

Temperaturregulering		
Punkt	Test	Forventet resultat
1	Visuel test	Reguleringssystem er implementeret med en Arduino Mega2560.
2	Visuel test	Der er anvendt et shield fra PR electronics, med påmonteret LM75.
3	HW test	Arduinoen skriver til- og læser fra Hub'en.
4	Måling af temperatur. Måling af signal til varme-kontakt.	Arduinoen sender et højt signal til kontakten, når temperaturen er under den ønskede værdi.
5	Måling af dimensioner	Målene overstiger ikke LxBxH: 10x5x4 cm.

Tabel 19: Accepttest, Temperaturregulering

Temperatursensor		
Punkt	Test	Forventet resultat
1	Visuel test	Implementeret med en LM75 temperatursensor.
2	Temperaturmåling.	Måler temperaturer i intervallet 0°C-30°C.
3	Sensor-målinger sammenlignes med eksterne målinger.	Målingerne har en afvigelse på maks ±2°C

Tabel 20: Accepttest, Temperatursensor

Varmeaktuator		
Punkt	Test	Forventet resultat
1	Visuel test	Implementeret i form af en MOSFET.

Tabel 21: Accepttest, Varmeaktuator

Hub		
Punkt	Test	Forventet resultat
1	Visuel test	Implementeret med en Raspberry Pi 4.
2	Signal test	Hub'en kan kobles til internettet.

Tabel 22: Accepttest, Hub

Smartkontakt		
Punkt	Test	Forventet resultat
1	HW test	Kontakten tænder hvis energiprisen er under det indstillede prisloft.
2	HW test	Kontakten slukker hvis energiprisen er over det indstillede prisloft.
3	Visuel test	Kontaktenens 'on' -tid summeres til den givne tid.

Tabel 23: Accepttest, Smartkontakt

Touchskærm		
Punkt	Test	Forventet resultat
1	Visuel test	Touchskærmen skal implementeres via Raspberry Pi 4.
2	Signal test	Touchskærmen skal kunne kobles til internettet.
3	Visuel test	Implementeret med en Raspberry Pi 7" touchskærm

Tabel 24: Accepttest, Touchskærm

Energimåler		
Punkt	Test	Forventet resultat
1	Måling af strømforbrug	Energimåleren mäter strömforbruget.
2	Visuel test	Implementeret med en lyspuls-måler der vender ind mod elmåleren.
3	Måling af frekvens	Lyspuls-måleren mäter frekvenser i området 10-100Hz

Tabel 25: Accepttest, Energimåler

5 Metode og Proces

af Daniel Naddaf

Projektets analytiske og metodiske del tager sit udgangspunkt i undervisningen i 2. semester kurset *Indledende System Engineering* (SW2ISE-01)[1], herunder specielt anvendelsen af SysML notation i rapportens figurer. I arkitekturfasen er der udarbejdet et block definition diagram (BDD) og et internal block diagram (IBD) for at give et overblik over systemets HW-dele og signalerne herimellem. Dertil er der udarbejdet en domænemodel og et sekvensdiagram, som skal være med til at afklare hvordan systemets forskellige elementer interagerer, hvordan de relaterer sig til hinanden, samt hvordan de gør dette over tid, med udgangspunkt i usecuses.

Ved dette punkt i rapporten er den analytiske del af projektet sådan set overstået. Systemet blev analyseret på baggrund af dets aktører og med udgangspunkt i problemformuleringen, for at opstille relevante usecases og heraf definere de funktionelle- og ikke-funktionelle krav, samt de dertilhørende accepttests.

Den overordnede udviklingsproces har fulgt *Semesterprojektmodellen for 2. semester*[3] tæt, hvor den test- og feedbackdrevne iterative del af processen har været opdelt i systemets 3 delsystemer; Energy Consumption Calculator (ECC), Temperature Regulation og HUB. Tilgangen til softwaredesignet og -implementeringen har derfor været forskellig mellem systemene, men dermed også specielt tilpasset de respektive delsystemers opgaver. Programmerings-tilgangen uddybes for hvert delsystem i afsnit 7, *Systemdesign*. Men fælles for delsystemerne er at programmerne er indelt i klasser og/eller metoder, som baseres på entity-controller-boundary (ECB) arkitektur-mønstret. Dette arkitektur-mønster danner grundlaget for den specifikke beskrivelse af softwaredesignet gennem applikationsmodellen, som består af klasse-, sekvens- og statemachine-diagrammer, og som er opdateret undervejs jf. den iterative del af Semesterprojektmodellen.

Implementeringer og modultest af delsystemerne har foregået separat og med forskellige metoder, som uddybes i afsnit 8 og 9.

5.1 Proces

af Jonas Kirkagaard Skyum

Projektet begyndte med en grundig udviklingsfase, hvor der blev diskuteret det aktuelle problem, defineret klare mål for temperaturreguleringssystemet og delt ambitiøse målsætninger. Formålet var at optimere energiforbruget i en husstand og som bruger kunne prioritere en ønsket temperatur, samt angive et prisloft for elprisen under en opvarmning. Der var høje ambitioner om at skabe det mest optimale og spændende produkt, men måtte erkendes, at det ikke helt kunne lade sig gøre, efter et dalende ambitionsniveau.

Den første del af arbejdsprocessen handlede om at udvikle hele systemet ud fra en grundig kravspecifikation, hvor alle kunne bidrage med input og idéer. Umiddelbart efter kravspecifikationsfasen delte gruppen sig i tre mindre enheder: HUB'en, temperaturreguleringen og elmåleren. I disse grupper arbejdedes der videre med design og implementering af hele systemet. Selvom der arbejdedes i separate grupper, var der stadig gode muligheder for at søge hjælp blandt gruppemedlemmerne. Ved at afholde interne møder med Kristian hver tirsdag, kunne der under hele projektet følges op på tidsplanen og diskutere fremskridtene i de forskellige systemer.

Efter arbejdet i de mindre grupper blev arbejdet med design og implementering samlet, og der blev udført tests på de forskellige dele, inden hele systemet blev samlet. Gennem hele projektforløbet er der blevet arbejdet ud fra semesterprojekt modellen for 2. semester. Hver undergruppe har stået for deres eget arbejde og tilgang til projektet, med Kristian til rådighed for teoretisk eller arbejdsmæssig vejledning efter behov.

Der refereres til Procesbeskrivelsen i bilag W for yderligere informationer omkring processen bag projektet.

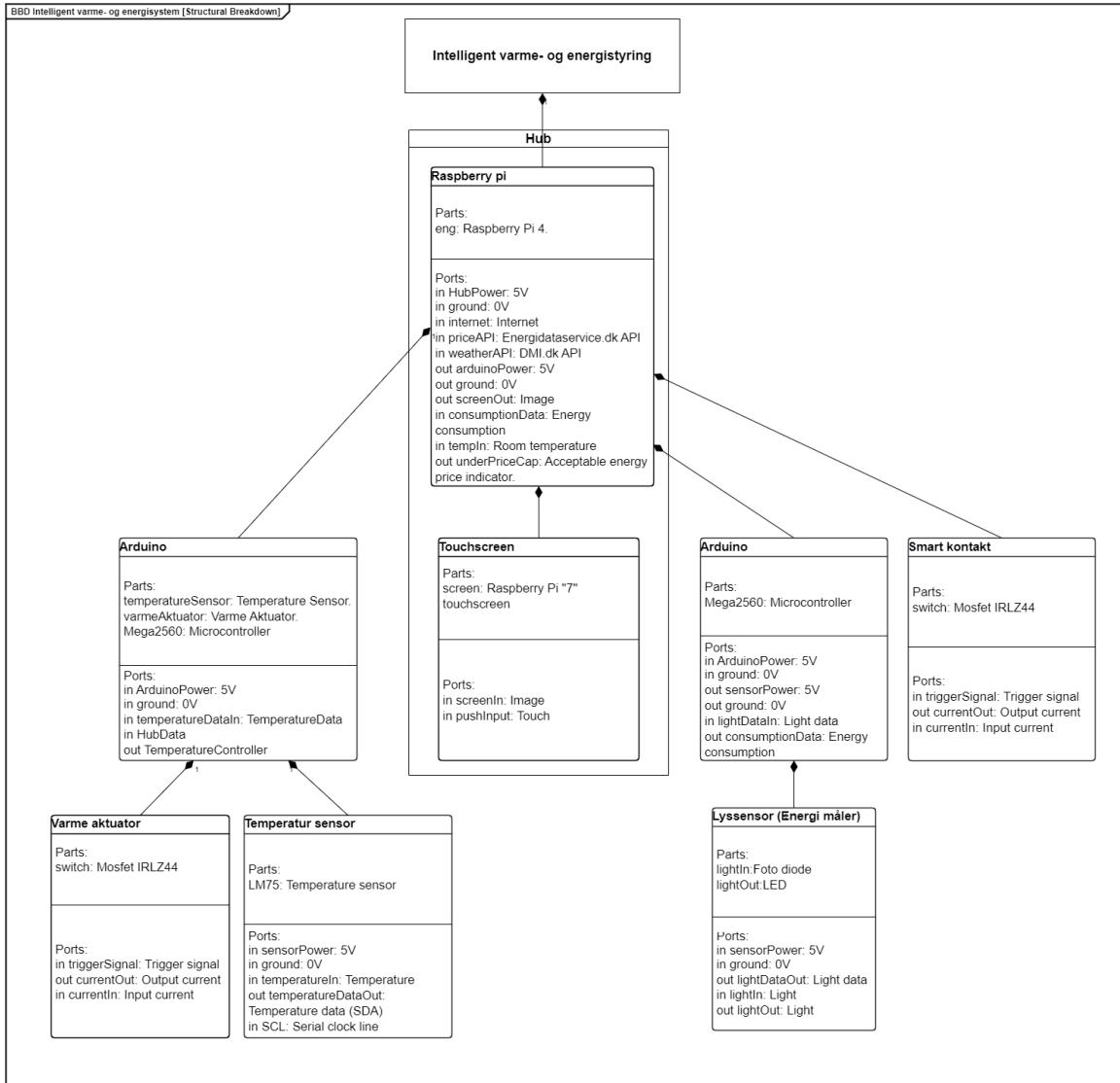
6 Systemarkitektur

6.1 Hardware arkitektur

6.1.1 BDD

af Daniel Naddaf & Jonas Kirkagaard Skyum

På figur 4 ses et Block Definition Diagram for hele systemet. Her er det også specificeret hvad, det indtil nu lidt flydende begreb, ”HUB’em” egentlig er og består af. Som det kan ses på figuren består systemet af to Arduinoer, én som styrer temperatur reguleringen, og én som styrer ECC’en. Arduinoen som styrer temperatur reguleringen, kommunikerer med en LM75 temperatursensor og en varme aktuator. Arduinoen som styrer ECC’en, modtager signaler fra en fotodiode, som er påmonteret forbrugerens el måler, på hvilken den modtager lyspulser. I HUB’en er der en Raspberry Pi 4 og en touchskærm, hvor der på skærmen, vises et interface, som drives af Raspberryen.



Figur 4: BDD. Findes som bilag A.

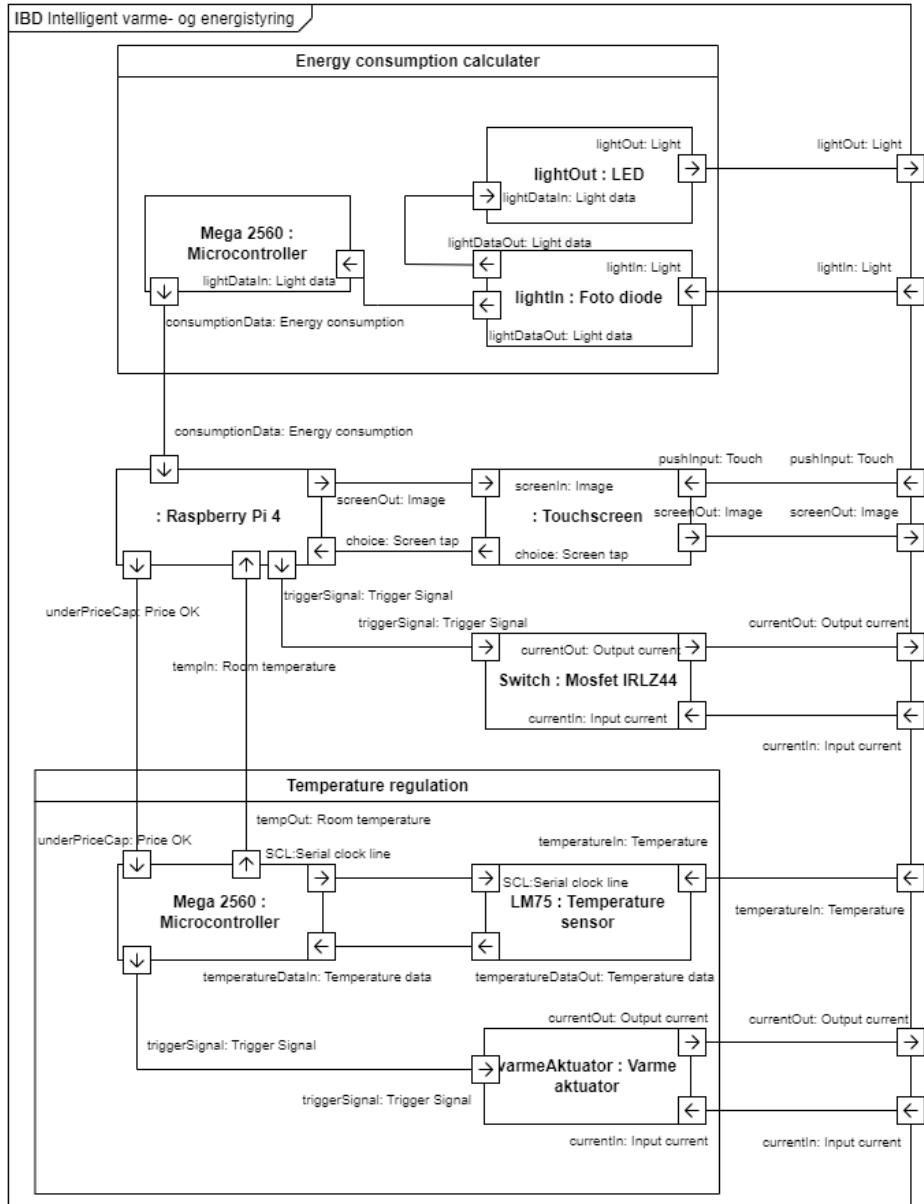
BDD	
Blok	Funktionalitet
Raspberry pi	Sender og modtager data til og fra arduinoerne og touch screen. Henter data fra internettet.
Touchscreen	Viser brugerinterface, som styres af raspberry pi.
Arduino mega 2560	Styrer varmeaktuator. Modtager og procceserer data fra temperatur sensor og lysenesor.
Smartkontakt	Tænder og slukker for en kontakt i hjemmet, baseret på signaler fra Raspberry pi
Temperatursensor	Måler temperaturer og sender data til arduino gennem en I2C forbindelse.
Varmeaktuator	Tænder og slukker for varmelegeme.
Lyssensor	Registrerer lyspulser fra hjemmets energimåler.

Tabel 26: Tabel for BDD

6.1.2 IBD

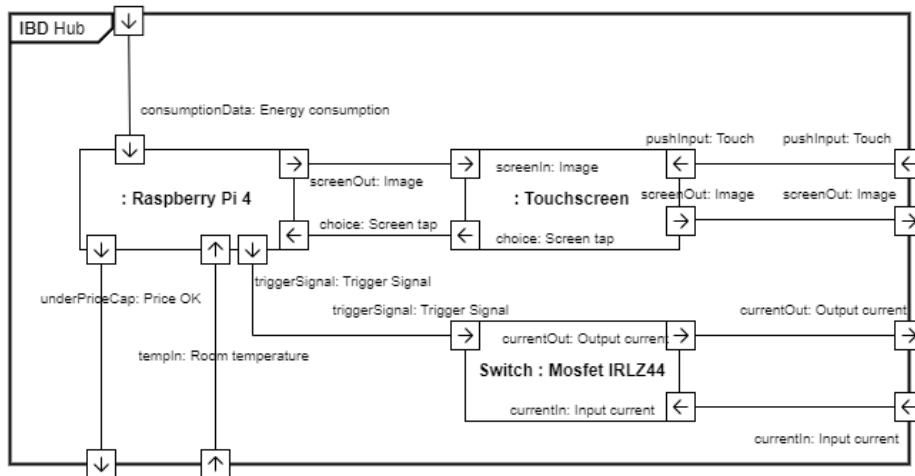
af Daniel Naddaf & Jonas Kirkagaard Skyum

På figur 5 ses et Internal Block Diagram for hele systemet, hvor de enkelte delsystemer også er specificeret. Systemet er delt op i tre delsystemer, som uddybes længere nede. I IBD'en anvendes der udelukkende atomic input- og output flow ports. Hvilket indikerer at kommunikationen for hver forbindelse er envejs.



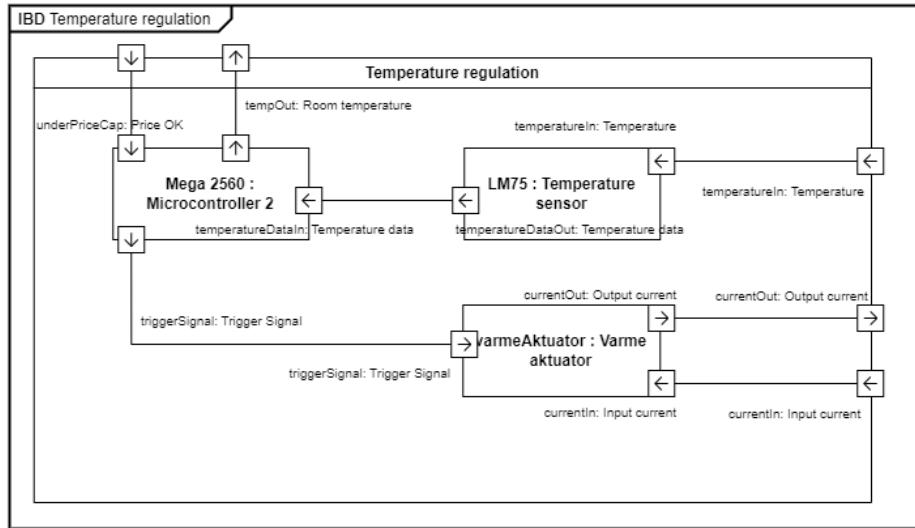
Figur 5: IBD

HUB'en består af tre dele: Raspberry Pi 4, Touchscreen og en Mosfet IRLZ44. Raspberry'en sørger for at sende "Price OK" ud af sit eget system, som ender i Arduinoen som set på figur 6. Den sørger også for at sende "triggerSignal" til Mosfetten om at den skal åbne eller lukke. Udover de to signaler sender den signalet, som skal vises på touchskærmen. Som inputs får den data fra de andre systemer så som "screen tap" og "Room temperatur". Derudover sender den "underPriceCap" til arduinoen, som derefter kan bruge det til temperatur reguleringen. Touchscreen sørger for at vise interfacet til omverdenen, og at modtage inputs, som sendes til raspberry'en. Mosfetten sørger for at sende "Output current til" omverdenen (varmelegemet) via "Input current", og den modtager "Trigger signal" om hvorvidt der må åbnes eller lukkes for strømmen.



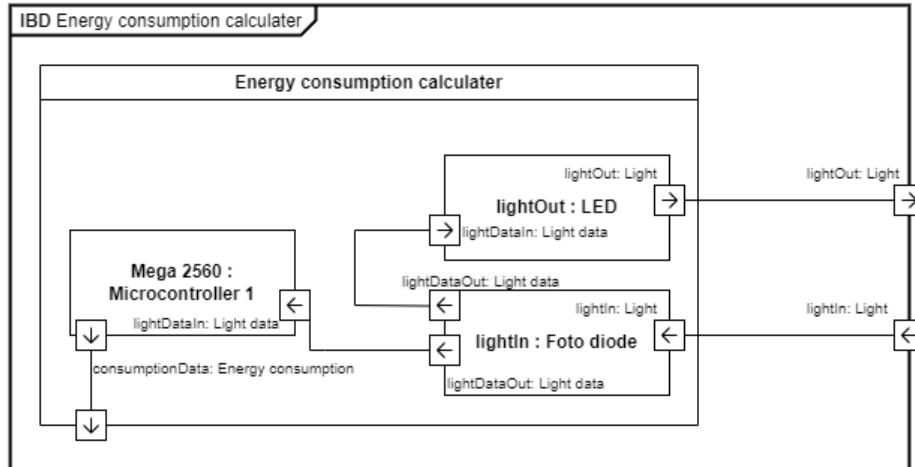
Figur 6: IBD Hub

Modul 2 består af temperatur regulerings algoritmen. I dette system er hjernen en Arduino Mega2560, som har kontakt med en varme aktuator og en temperatursensor LM75. Arduinoen modtager en temperatur fra temperatursensoren, som mäter temperatur fra omgivelserne. Ud fra den temperatur og "Price OK" tager den beslutninger om der må reguleres eller ej.



Figur 7: IBD Temperature regulation

Det tredje og sidste modul består af en Arduino Mega2560, en LED og en fotodiode. Dette moduls formål er at sende en frekvens fra Arduinoen til Raspberry'en. Dette gøres ved at måle lyspulser med fotodioden og derefter sende denne frekvens via driveren. LED'en skal sende "Light" til omverden, så forbrugeren kan se sit eget forbrug.



Figur 8: IBD Energy consumption calculator

6.1.3 Note om reducering af projektets størrelse

Implementeringen af smart kontakten fravælges på grund af tidsmangel. Ved fjernelsen af smart kontakten, følger funktionaliteten af fanen "enheder" også, da fanen er en del af det samme system.

Derudover fjernes funktionaliteten af fanen ”data” fra hub’en, da udviklingen af algoritmen vil kræve for meget tid, til at lave de forskellige funktioner, som vises på hub’en.

Det sidste som fjernes er brugerens initiering af hub’en. Initieringen vil istedet ske på forhånd. Dertil fjernes fanen ”indstillinger” da man ikke længere kan ændre på leverandør og lokation osv.

Denne beslutning er truffet på baggrund af mangel på mænd til at arbejde på implementeringen af forskellige dele. Manglen på mænd betyder, at gruppen ikke har tilstrækkelig tid til at håndtere opgaven

Alt dette betyder også at der kun vil (og kan) arbejdes videre med use cases 2, 3, 4 og 8.

IBD					
Signalnavn	Funktion	Område	Port1(source)	Port2(destination)	Kommentar
underPriceCap	Signalerer at elprisen er under prisloftet.	0-5V	Raspberry Pi 4	Microcontroller 2	N/A
triggerSignal	Signalerer at smartkontakten skal tændes.	0-5V	Raspberry Pi 4	MOSFET IRLZ44	N/A
screenOut	Sender data til touchskærmen, om hvad der skal vises.	0-5V	Raspberry Pi 4	Touchscreen	N/A
consumptionData	Sender data til raspberry pi om energiforbruget.	0-5V	Microcontroller 1	Raspberry Pi 4	N/A
tempOut	Sender temperaturen til Raspberry Pi.	0-5V	Microcontroller 2	Raspberry Pi 4	N/A
triggerSignal	Tænder/slukker varmeaktuator	0-5V	Microcontroller 2	varmeAktuator	N/A
lightDataOut	Sender et højt signal ved detektering af lys	0-5V	lightIn	Microcontroller 1, lightOut	N/A
lightOut	Udsender lys til omgivelserne	N/A	lightOut	Omgivelser	N/A
choice	Sender placeringen af et skærmtryk til Raspberry Pi.	N/A	Touchscreen	Raspberry Pi 4.	N/A
screenOut	Visuelt billede, som vises af skærmen.	N/A	Touchscreen	Omgivelser.	N/A
currentOut	Strømmen, som smartkontakten åbner for.	N/A	MOSFET IRLZ44	Omgivelser	N/A
temperatureDataOut	Rå temperaturdata fra temperatursensoren.	0-5V	Temperature sensor	Microcontroller 2	N/A
currentOut	Strømmen, varmeaktuatoren lukker ind til varmelegetemet	N/A	varmeAktuator	Omgivelser	N/A
lightIn	Det blinkende lys fra elmåleren, som skal registreres af fotodioden.	N/A	Omgivelser	lightIn	N/A
pushInput	Fysisk berøring af touchscreen.	N/A	Omgivelser	Touchscreen	N/A
currentIn	Strømmen, som smartkontakten skal viderelevere.	N/A	Omgivelser	MOSFET IRLZ44	N/A
temperatureIn	Temperaturen i lokallet	0°C-30°C	Omgivelser	Temperature sensor	N/A
currentIn	Strømmen, som skal leveres til varmelegetemet.	N/A	Omgivelser	varmeAktuator	N/A

Tabel 27: Tabel for IBD

6.2 Software arkitektur

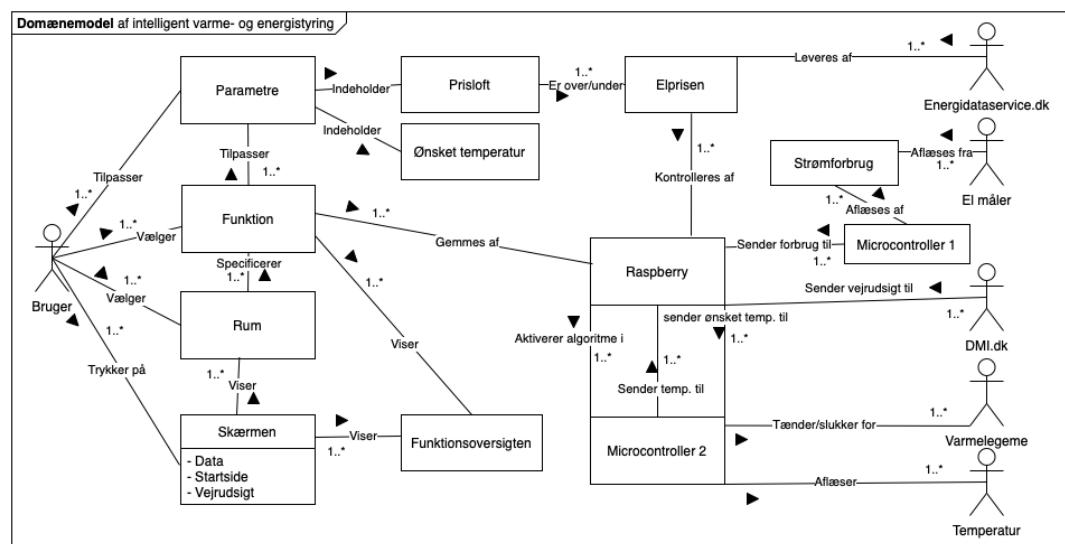
6.2.1 Domænemodel

af Daniel Naddaf & Jonas Kirkagaard Skyum

Denne domænemodel repræsenterer et intelligent varme- og energistyringssystem. Modellen illustrerer, hvordan data flyder gennem de forskellige dele af systemet og interagerer med forskellige komponenter. Brugeren interagerer med en skærm for at justere parametre som ønsket temperatur og prisloft. Disse parametre sendes til en central enhed, Raspberry, som modtager elprisdata fra Energidataservice.dk og vejrudsiger fra openweathermap.org.

Raspberryen styrer to microcontrollere: Microcontroller 1 overvåger frekvensen for strømforbruget og sender data tilbage til Raspberry, mens Microcontroller 2 styrer varmelegemerne baseret på ønsket temperatur. Rumtemperaturen specificeres og overvåges, og funktionerne vises i en funktionoversigt, som brugeren kan tilgå via skærmen.

Elmåleren aflæser strømforbruget og sender data tilbage til Raspberry, hvilket sikrer, at systemet løbende justerer sig for optimal energistyring. Sammen skaber disse komponenter et effektivt og brugervenligt system til varme- og energistyring.



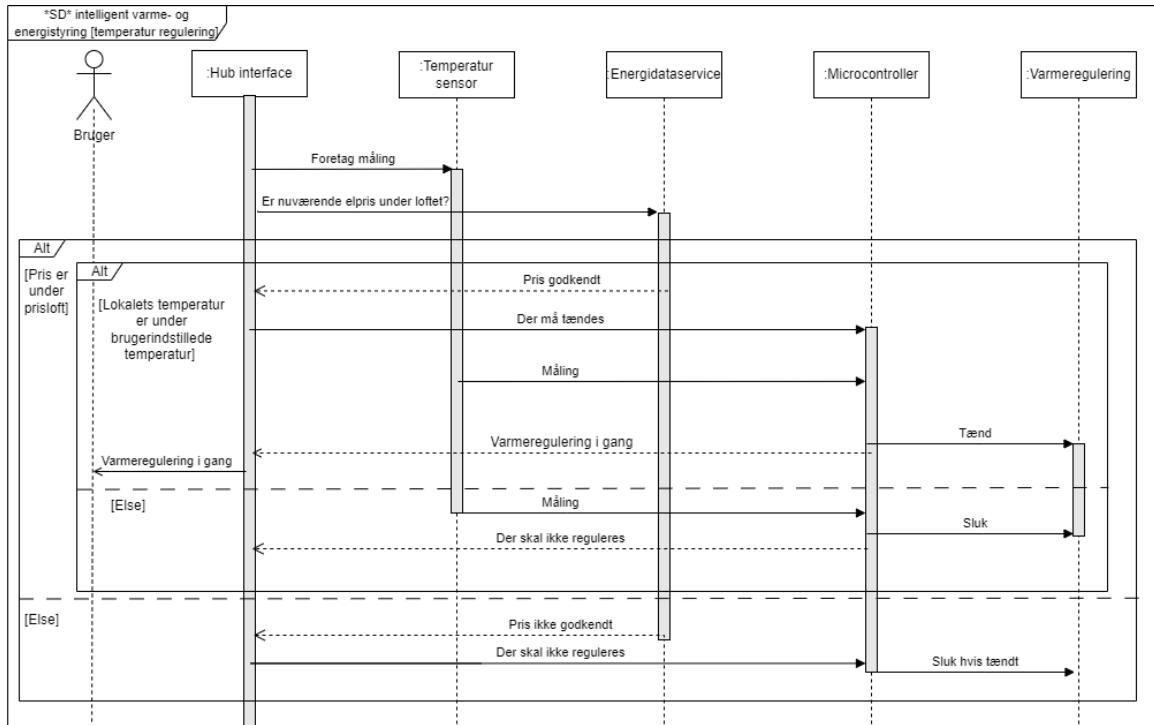
Figur 9: Domænemodel

6.2.2 Sekvensdiagram

af Jakob & Simon Eifer & Khaled Omar

Figur 10 viser processen for, hvordan objekter i systemet kommunikerer med hinanden over tid. Diagrammet viser specifikt, hvordan temperaturreguleringen foregår i dette projekt. Diagrammet er delt op i to ”alt” cases hvor resultatet forgrener ud i to mulige retninger. Første af disse er i forhold til pris, hvor den anden er i forhold til temperatur. Processen er baseret på, at brugeren har prædefineret præferencer for temperaturreguleringen. Først vil Hub-interfacet tjekke to parametre: om temperaturen ligger under den brugerdefinerede temperatur, og om den nuværende elpris er under

det indstillede prisloft. Hvis begge tilfælde gælder, vil varmereguleringen begynde. Når processen er færdig, vil varmereguleringen slutte, og diagrammet er gennemført. Prisloftet er den primære og første kontrol, da dette er den mest afgørende faktor. Hvis denne ikke er mødt, vil diagrammet afslutte uden at tjekke temperaturen, og ingen varmeregulering vil finde sted, da prishesparelse er en væsentlig søjle i dette projekt.

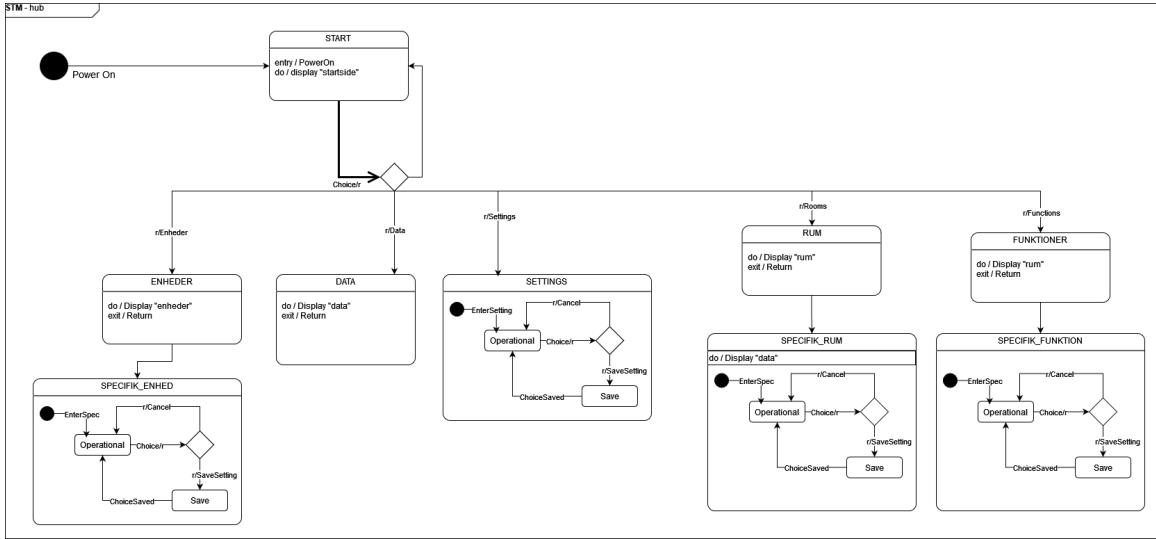


Figur 10: SD diagram

6.2.3 State machine diagram

af Khaled Omar & Filip Alberg & Nichlas

På figur 11 viser hub'ens system, i form af tilstande, og hvordan det skifter mellem forskellige visningstilstande, opsætningstilstande og specifikke indstillinger baseret på brugerens valg. Diagrammet illustrerer, hvordan systemet udfører handlinger i forskellige tilstande entry, do og exit. Systemet kan også gemme indstillinger, afbryde eller vende tilbage til en forrige tilstand, baseret på brugerens valg.



Figur 11: STM Diagram for Hub. Vedhæftet som bilag F

6.2.4 Protokoller

af Daniel Naddaf

Kommunikationen mellem systemets delsystemer vil foregå som seriel kommunikation via UART. Protokollen for al seriel kommunikation ses i figur 12, hvor enhver sekvens af bytes indrammes med start- og stop-bytes, angivet som *STX* og *ETX* i figuren. *CNT*, angiver hvor mange bytes der sendes i den bytestream, den er en del af. Værdien bruges af modtageren til at kontrollere om det rigtige antal af bytes er modtaget. *CNT*-værdien tæller ikke de sidste 2 bytes med. Dernæst sendes et *type*-byte, som skal fortælle modtageren hvilken type signal der sendes, sådan at signalet bliver behandlet korrekt på modtagersiden. På tabel 28 ses værdierne for *Type*-bytes. Herefter sendes de faktiske bytes, som skal bruges i modtagerens program. Sekvensen afsluttes med en checksum, inden stopbyten sendes. Checksummen, *CHSM*, bruges til at kontrollere om det signal, som er modtaget, er det samme som det, der blev sendt. Dette kan gøres med flere forskellige algoritmer, men i dette projekt er der blot anvendt en simpel funktion, som summerer alle bytes op til checksummen. Dette gøres af afsenderen ved afsendelsen og af modtageren ved modtagelsen af signalet. Modtageren sammenligner den modtagne checksum med dens egen, beregnede checksum. Det bør noteres at checksummen har en størrelse på 1 byte, og at der i alle tilfælde vil summeres til et tal som er større end 255, idet STX er 255. Signalet vil stadig producere en unik checksum, idet adderingen vil lave et *wraparound* ved 255, sådan at den reducerede checksum får en værdi, som svarer til $sum \% 255$.

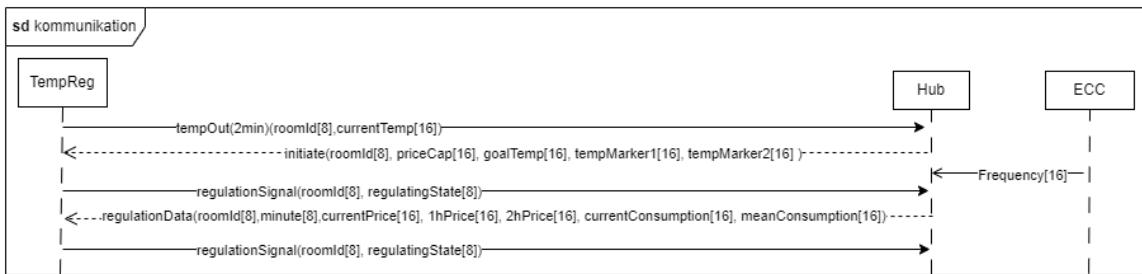
Byte:	0	1	2	3	4	...	$n+3$	$n+4$	$n+5$
Indhold:	STX	CNT	TYPE	B0	B1	...	Bn	CHSM	ETX

Figur 12: Protokol for seriel kommunikation mellem delsystemer.

Type	DEC-værdi	Afsender
regulationSignal	114	Arduino
initiateSignal	115	Raspberry Pi
tempOut	116	Arduino
regulationData	117	Raspberry Pi

Tabel 28: Type-bytes og deres værdier

På figur 13 ses et sekvensdiagram for afsendelsen og modtagelsen af signalerne delsysteme iblandt. Signalerne i diagrammet er angivet med deres signalnavn efterfulgt af en parantes, som angiver signalets indhold. De firkantede paranteser angiver antallet af bits i delindholdet af signalet. Det, som er angivet med 16 bits opdeles i 2 8-bit signaler med MSB og LSB. Tidspunktet for afsendelsen af frekvensen fra ECC er arbitraert, da dette signal sendes periodisk fra ECC'en.



Figur 13: Sekvensdiagram for afsendelse og modtagelse af signaler mellem delsystemerne. Frekvens-signalen er valgt til et arbitraert tidspunkt, da dette sendes kontinuerligt. Temperature Regulation er forkortet til TempReg. Findes som bilag B.

7 Systemdesign

7.1 Energy consumption calculator

af Daniel Naddaf & Jonas Kirkagaard Skyum

Denne del af systemet, som forkortes til ECC, skal beregne frekvensen af lyspulserne fra hjemmets egen energimåler. Til dette udvikles et frekvenstæller kredsløb, som skal kunne opfange disse lyspulser og sende dem til en Arduino, som skal bearbejde signalerne, og til sidst kunne levere en frekvens til HUB'en, som omregner det til et forbrug. Da denne konfiguration vil dække over den blinkende diode på hjemmets elmåler, implementeres der en LED som vender væk fra elmåleren, som skal immitere frekvensen, sådan at man visuelt kan se, at tælleren er monteret korrekt med det samme.

7.1.1 Hardware design

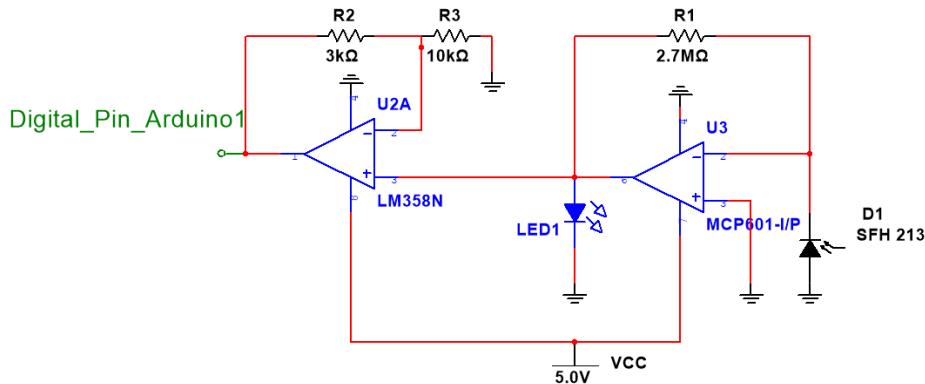
af Jakob & Nichlas

Hardwaredesignet består af et operationsforstærker-kredsløb bestående af to operationsforstærkere, hvor den første forstærker signaler fra en fotodiode og udsender det gennem en LED. Samtidig sender den et signal til den næste operationsforstærker der så sender signalet til en arduino. Signalet, som

leveres til LED'en, er det samme, som skal leveres til microcontrolleren efter det er forstærket. Alt pånær microcontrolleren skal sammensættes så kompakt som muligt, sådan at det let kan monteres som en lille kasse på den eksisterende energimåler. Elmåleren består af en lysdiode, en grøn LED, en Arduino Mega 2560, to operationsforstærkere (MCP601 og LM358N), og tre forskellige modstande. Systemet er designet til at forstærke signalet efter lysdioden, da en enkelt operationsforstærker ikke var nok til at opnå den nødvendige spænding på 3 V, som er tærsklen for, at arduinoen kan registrere et givet signal som et HIGH signal. Datasheet for hardwaren findes under bilag.

Komponentliste

- 1x MCP601 (Operationsforstærker)
- 1x LM358N (Operationsforstærker)
- 1x $2,7\text{M}\Omega$ (Modstand)
- 1x $3\text{k}\Omega$ (Modstand)
- 1x $10\text{k}\Omega$ (Modstand)
- 1x SFH213 (Fotodiode)
- 1x Grøn LED
- 1x Arduino Mega 2560

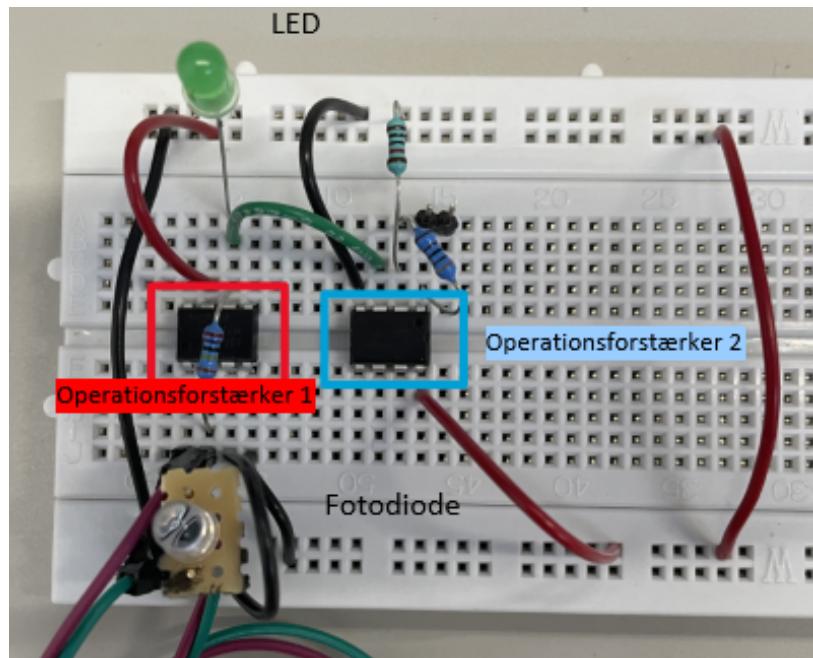


Figur 14: Kredsløbsdiagram for frekvenstælleren.

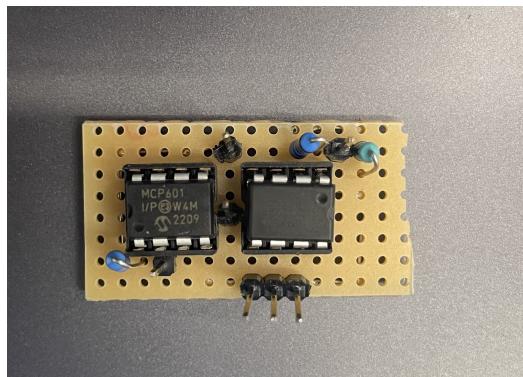
7.1.2 Realisering

af Jakob & Nichlas

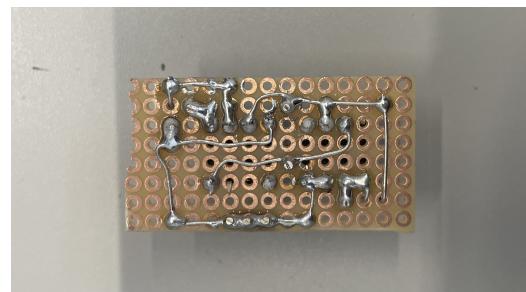
Opstilling af Elmålerkredsløb på fumlebræt ses på figur 15. Efter test af, at kredsløbet fungerer som forventet, blev det designet på vero-board, som ses på figur 16. Det færdig loddet kredsløb på vero-board er det der anvendes som det endelig produkt.



Figur 15: Realisering af kredsløb på fumlebræt.



Figur 16: Realisering af kredsløb på vero-board

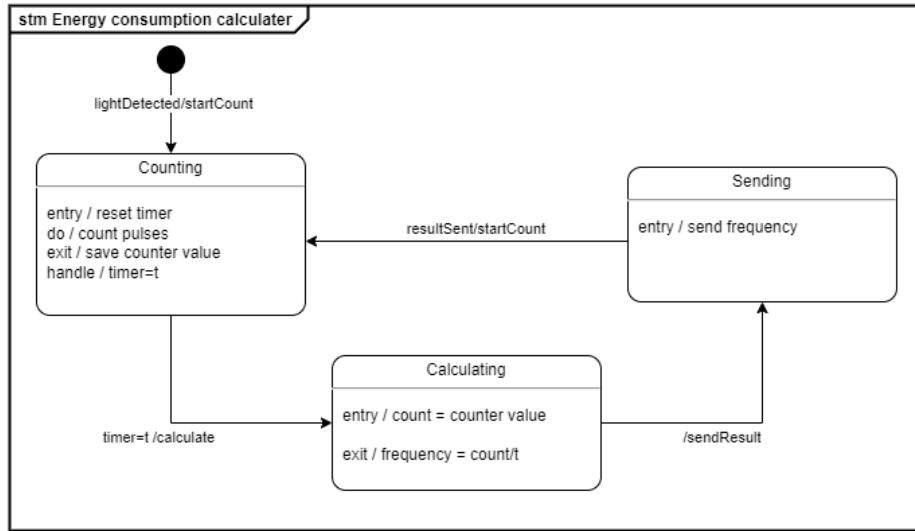


Figur 17: Bagsiden af vero-boardet

7.1.3 Software design

af Daniel Naddaf & Jonas Kirkagaard Skyum

Statemachine-diagrammet på figur 18 viser hvordan processerne i algoritmen forløber. Når systemet startes, begynder Arduinoen at tælle lyspulser, mens den i baggrunden har en timer, som tæller op til tiden t . Så snart timeren når tiden t , gemmes antallet af talte lyspulser. Dette antal deles med tiden t , og derfra fås frekvensen af lyspulserne i løbet af tiden t . Frekvensen sendes herefter



Figur 18: State machine diagram for Energy consumption calculator.

videre til HUB'en og timeren genstartes. Herefter gentages processen igen. Alt dette kan gøres i en enkelt klasse for sig selv.

7.2 Temperatur regulering

af Daniel Naddaf & Jonas Kirkagaard Skyum

Denne del af systemet står for at regulere temperaturen i et lokale baseret på inputs fra brugeren, signaler fra sensorer, håndtering af aktuatorer og indsamling af data fra nettet via HUB'en. Delsystemets primære funktion er at sørge for, at temperaturen i det rum, som det er installeret i, altid har brugerens foretrukne temperatur, så vidt det er muligt. Dog vil reguleringen ikke altid træde i kraft øjeblikkeligt efter at temperaturen falder under den ønskede værdi. Delsystemet skal nemlig vurdere om det kan betale sig at opvarme hjemmet med det samme, eller om der skal ventes. Dette baseres på den aktuelle elpris og energiforbruget, samt elprisen efter 1 og 2 timer. Ydermere kan brugeren indstille 2 tærskelværdier for temperaturen, uddover den ønskede temperatur, som algoritmen vil bruge til at vurdere, hvornår der reguleres på baggrund af de forskellige nævnte faktorer.

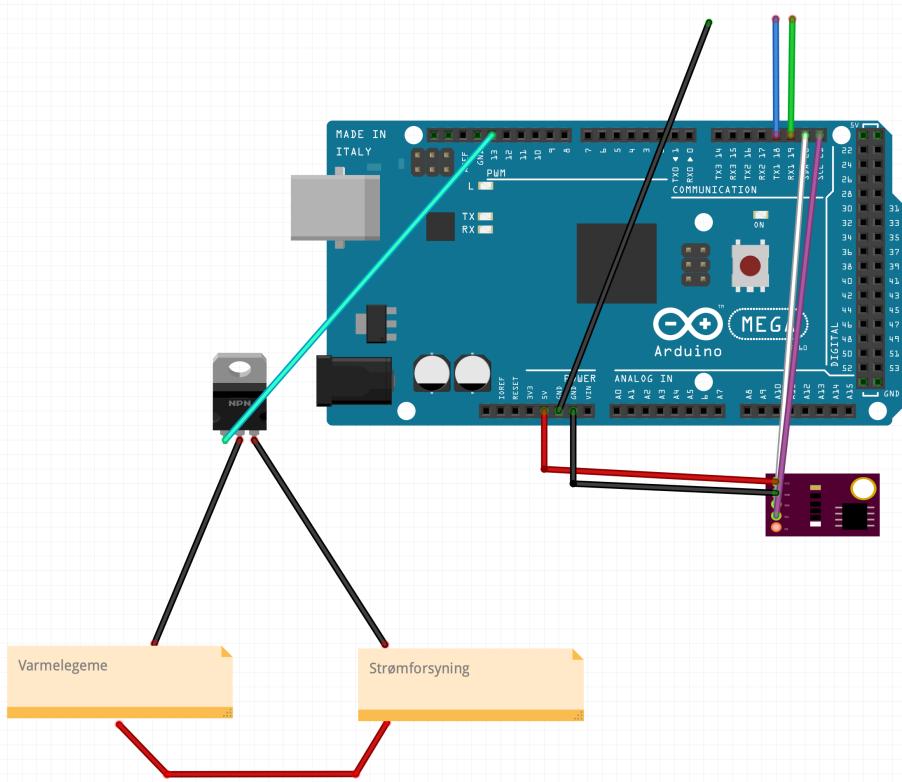
7.2.1 Hardware design

Som det kan ses på figur 19, er der opstillet et design af temperaturreguleringssystemet, hvor komponenterne fremgår af tabel 29.

Komponentliste:	
Komponent	Antal
Arduino Mega2560	1
Varme legeme	1
Strømforsyning	1
Mosfet IRLZ44N	1
LM75	1

Tabel 29: Komponentliste over hardware design

I opstillingen er Arduinoen forbundet til GND på temperatursensoren. MOSFET’ens gate er forbundet til Arduinoens port 13, som styrer åbningen og lukningen af strømtilførslen. MOSFET’ens drain er forbundet til den negative pol på varmelegemet, mens dens source er forbundet til den negative pol på strømforsyningen. De positive poler på strømforsyningen og varmelegemet er også forbundet. Arduinoens Vcc er forbundet til temperatursensoren. En hvid ledning forbinder sensoren til Arduinoens SDA-port, og en lilla ledning forbinder sensoren til Arduinoens SCL-port for at muliggøre I2C-kommunikation. Desuden er der en grøn ledning, der forbinder Arduinoens RX1-port til Raspberry Pi’s TX-port, en blå ledning, der forbinder Arduinoens TX1-port til Raspberry Pi’s RX-port, og en sort ledning, der forbinder Arduinoens GND til Raspberry Pi’s GND. Denne opsætning muliggør kommunikation via UART.



Figur 19: Hardware design for temperatur regulering

7.2.2 Software design

Tilgangen til designet af delsystemets software har været objektorienteret programmering, hvor programmet inddeltes i klasser, som i dette tilfælde relateres til hinanden gennem komposition og aggregation. Softwaredesignet er udarbejdet som en applikationsmodel med ECB-klasser, jf. afsnit 5, *Metode og Proces*, og består af et klassediagram, et statemachine-diagram og et sekvensdiagram. Applikationsmodellen er lavet med udgangspunkt i use case 3 ”Temperatur regulering - For lav temperatur”, som er en af de to use cases, der vedrører temperaturreguleringen. Den anden, use case 4 ”Temperatur regulering - For høj temperatur”, vil anvende nogle af de samme klasser og metoder, men vil ikke indeholde noget nyt. Det kan let indsese, på baggrund af beskrivelsen af use case 4 i afsnit 3.1, at implementeringen af denne er triviel, og det vil vise sig at en sådan implementering er implicit i applikationsmodellen for use case 3.

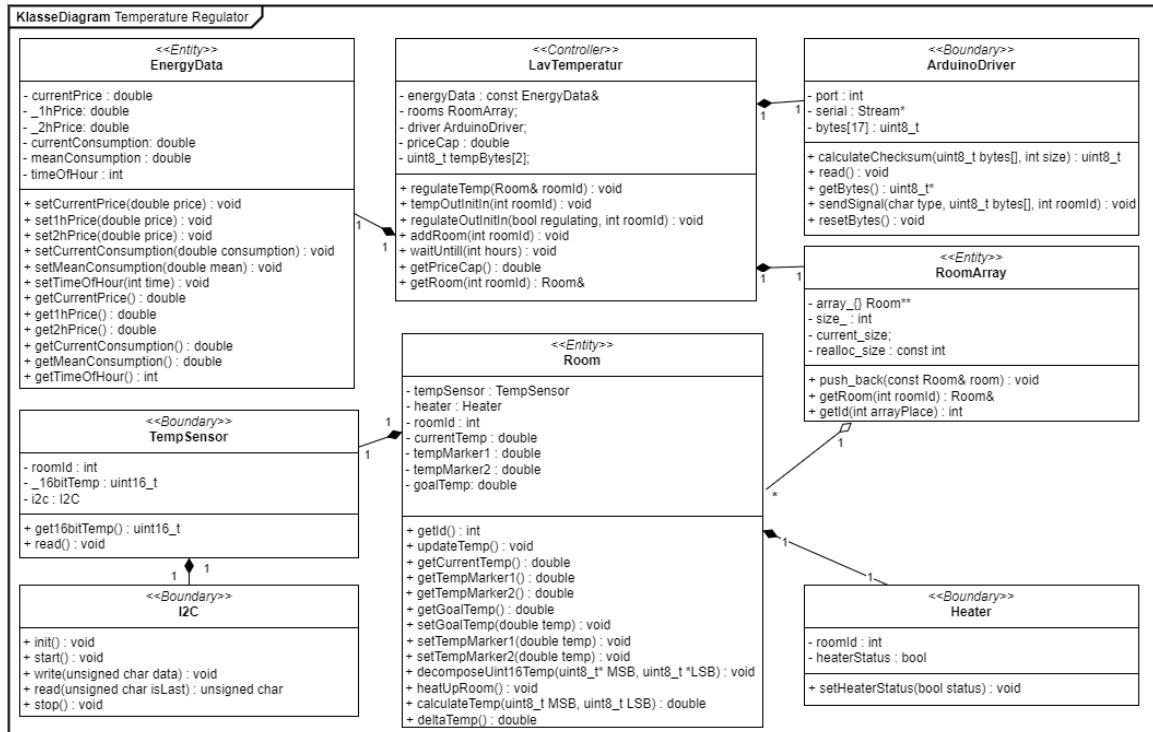
Klassediagrammet på figur 20 indeholder 8 klasser; 1 controller-, 3 entity- (domæne) og 4 boundary-klasser. Entity-klasserne er dataobjekter og indeholder informationer og logik, som bruges af deres relaterede klasser. Boundary-klasserne er grænsefladerne mellem systemet og omverdenen, eller deres omgivende systemer. Disse klasser håndterer inputs og outputs. Controller-klassen er mellemledet mellem boundary og entity klasserne. Den modtager inputtet fra boundary-klasserne og bruger dataen fra entity-klasserne til at behandle disse inputs før den returnerer resultatet tilbage til boundary-klasserne. Det er altså hele *hjernen* i systemet, og det er også her hovedalgoritmen

`LavTemperatur::regulateTemp()` findes.

De fleste klasser har en komposition-relation. Komposition er en ”part-of” relation, hvor den sorte diamant sidder på referencen til hvilken der er en ”part”. Ved denne relationstype vil reference-klasserne eje en eller flere instanser af part-klasserne. Disse instanser af part-klasserne eksisterer kun i deres respektive reference-klasser, idet de konstrueres og destrueres med referenceklasserne.

Aggregation er en ”has-a” relation, hvor den hvide diamant sidder på reference-klassen som ”har-en” anden klasse. Part-klassen eksisterer for sig selv, og bliver blot refereret til af reference-klassen. Det giver i dette tilfælde mening at der kan eksistere et `Room`, selvom det ikke nødvendigvis er en del af samlingen `RoomArray`.

Det er på baggrund af denne viden, at relationerne er valgt som de er, til dette delsystem.



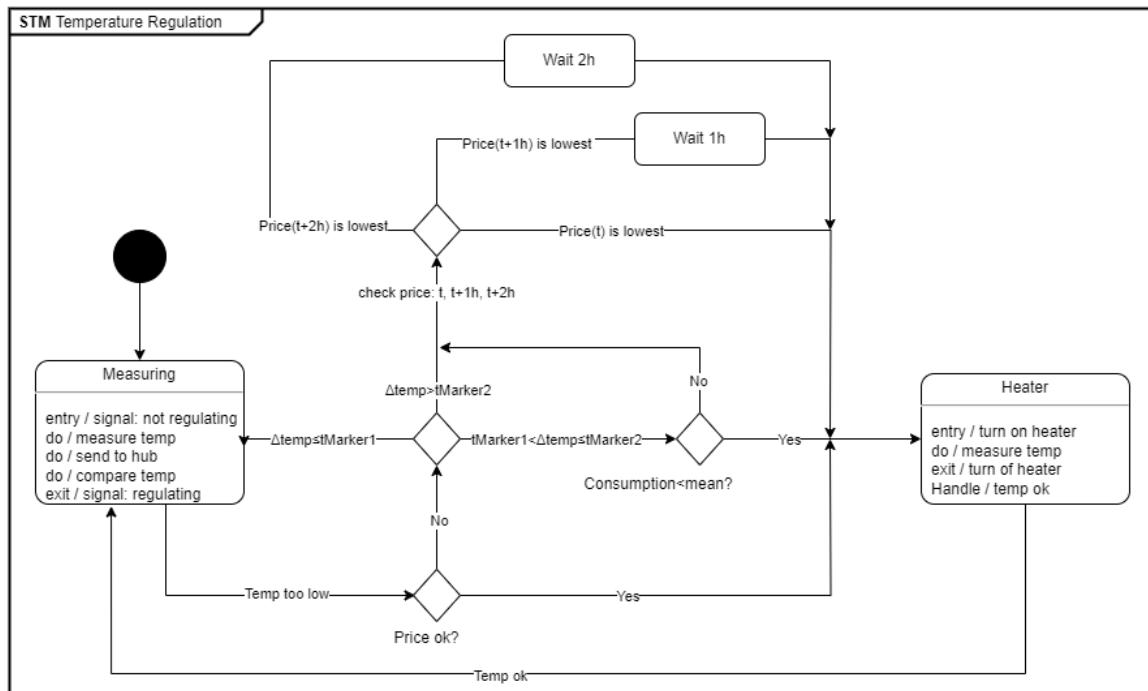
Figur 20: Klassediagram for temperatur reguleringen, use case 3. Vedhæftet som bilag C.

Statemachine-diagrammet på figur 21 beskriver de tilstands-drevne processer i den primære algoritme, som også optræder i klassediagrammet: `LavTemperatur::regulateTemp()`. Det er denne metode, som står for at samle alt logikken for temperaturreguleringen, indsamle data, behandle det og tage beslutninger baseret på elpriser, temperaturen og forbruget. Når metoden kaldes, dvs. når programmet starter, træder det ind i en løkke, hvor det konstant vil hente temperaturen (`currentTemp`) fra temperatursensoren, sende denne til HUB'en og sammenligne denne med den ønskede temperatur (`goalTemp`). Hvis resultatet af sammenligningen `goalTemp - currentTemp` er større end 0, sendes et signal til HUB'en om at der reguleres, og programmet forlader løkkken.

Herefter vil programmet vælge det mest fordelagtige tidspunkt at regulere på, sådan at prisen for at opvarme lokalet bliver så lav som muligt. Dette valg baseres på data som tempMarkers, elpriser

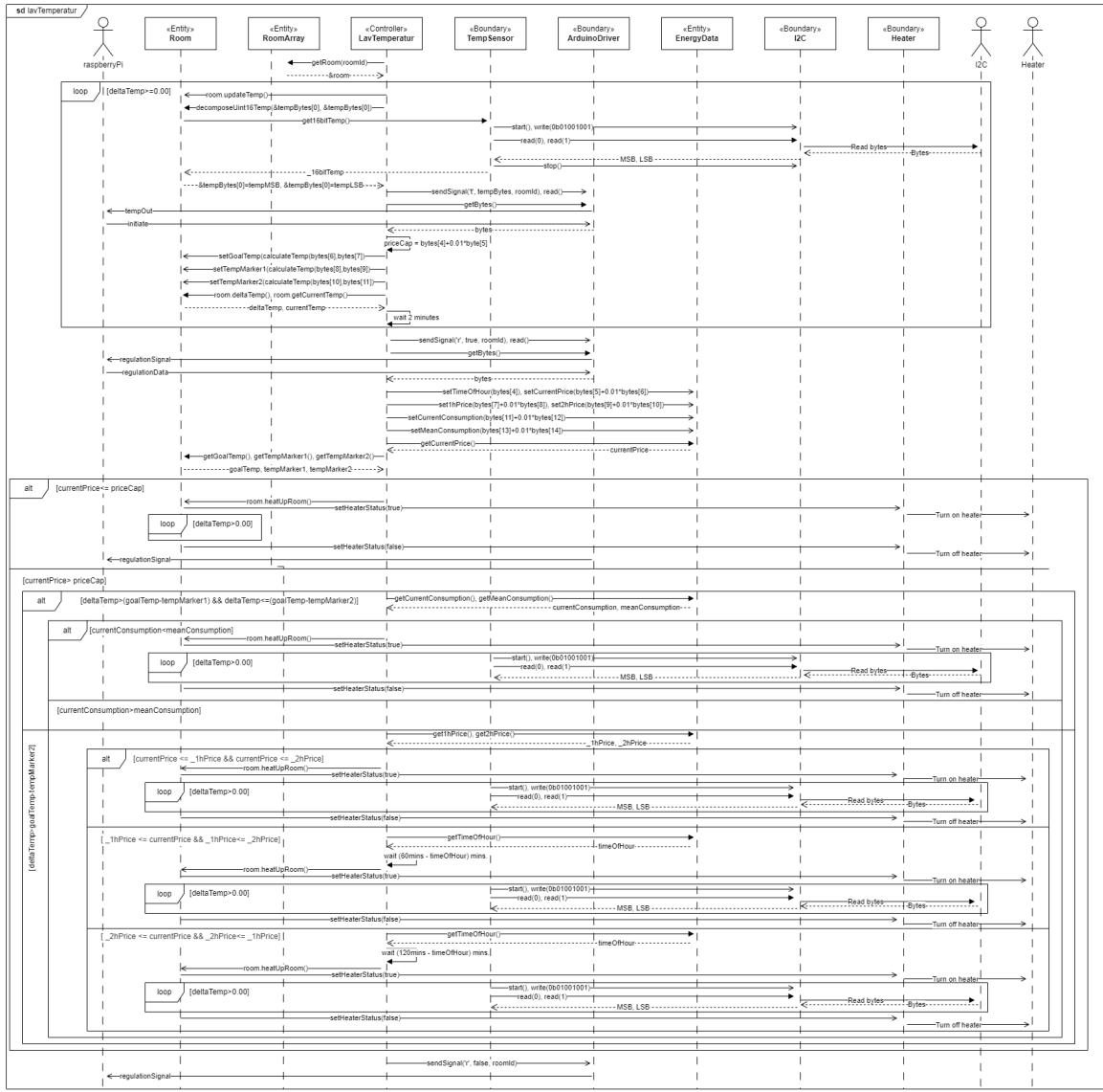
og forbrug, fra entity-klasserne. Hvordan den træffer valget kan ses på forgreningerne i figuren.

Under selve reguleringen vil programmet blive i en løkke; tilstanden *Heater* i stm-diagrammet. Når programmet går ind i løkken, tændes varmelegemet og programmet vil herefter konstant måle temperaturen i rummet og sammenligne den med *goalTemp*. Når temperaturen kommer op på *goalTemp*, vil programmet slukke for varmelegemet, signalere dette til HUB'en og returnere fra løkken. Der kan være flere årsager til at programmet ved et kald af metoden ikke vil registrere den ønskede temperatur; f.eks. hvis temperaturen skifter fra *goalTemp*-0.5 til *goalTemp*+0.5 mellem to læsninger. Det bør derfor noteres at temperaturer over *goalTemp* også vil udløse en returnering fra *Heater* løkken.



Figur 21: Statemachine-diagram for temperatur reguleringen, use case 3. Vedhæftet som bilag D.

Sekvensdiagrammet på figur 22 viser sammenspillet mellem klasserne i løbet af en reguleringssyklus. Diagrammet er meget stort og kan være utydigt i rapportens format, så der henvises til bilag E. Det er her klasse- og statemachine-diagrammerne forenes, idet processerne fra stm-diagrammet specificeres med klasserne og metoderne fra klassediagrammet. Ser man på diagrammet i sin helhed, kommer entity-controller-boundary mønstret hurtigt til kende, idet boundary-klasserne interagerer med delsystemets omgivelser, entity-klasserne indeholder og behandler data og controller-klassen indeholder den logik, som samler det hele.



Figur 22: Sekvensdiagram for temperatur reguleringen, use case 3. Vedhæftet som bilag E.

7.3 Hub

af Khaled Omar, Simon Eifer & Filip Alberg

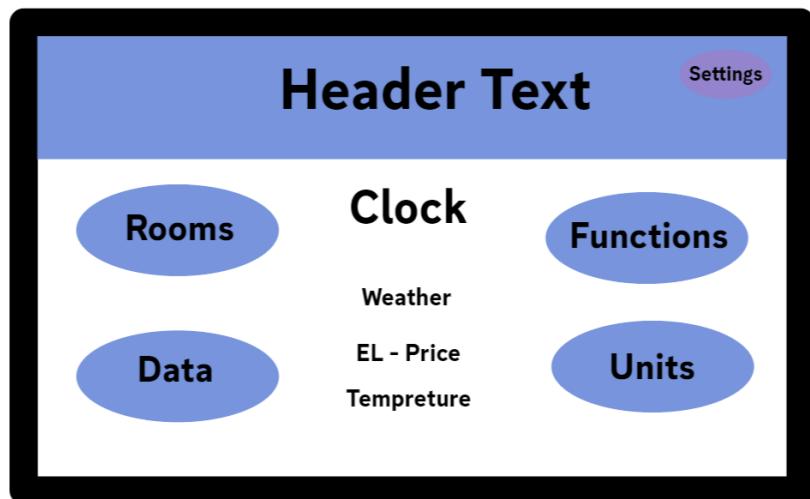
Hub'en er et delsystem hvor brugeren har mulighed for at interagere med hele systemet. Måden brugeren interagerer med systemet på, er ved at brugeren kan ændre på nogle parametre, herunder prisloft og temperatur, der bruges til at regulerer temperaturen. Disse parametre bliver sendt videre til temperatur regulereringssystemet hvorefter der bliver reguleret, ud fra de parametre brugeren har angivet.

7.3.1 Software design

Der udarbejdes mockups af HUB'ens interface, for at få en ide om hvordan fanerne vil se ud.:

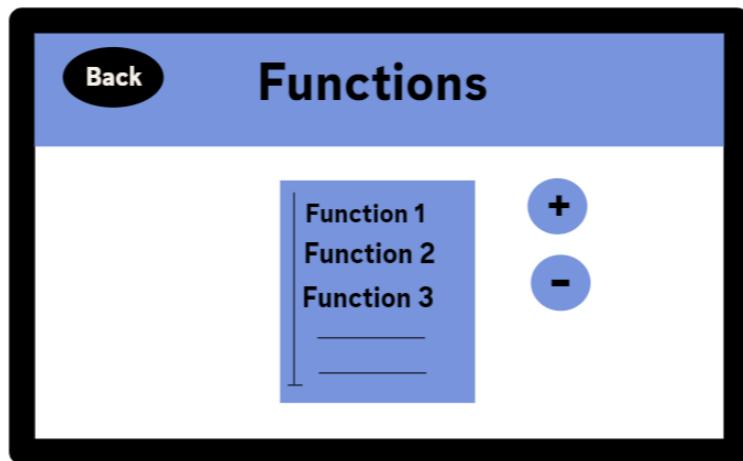
Design af interface:

af Khaled Omar



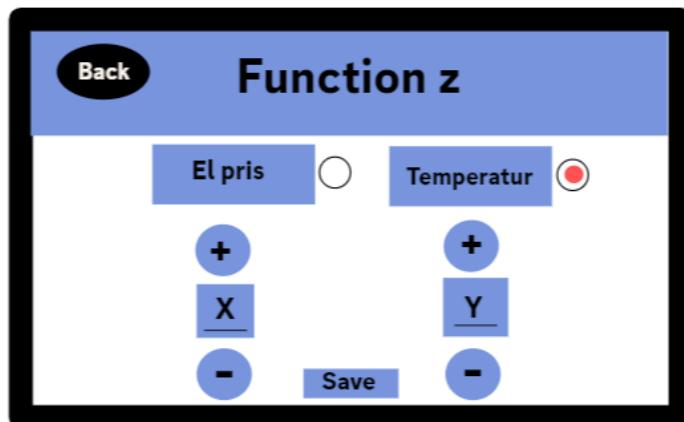
Figur 23: Design for Main Menu, Hub

Main Menu på figur 23 siden præsent præsentere brugeren for alle tilgængelige funktioner og muligheder i systemet. Derudover skal det vise aktuelle oplysninger som klokkeslæt, elektricitetspris, vejrforhold og temperaturen i et specifikt rum.



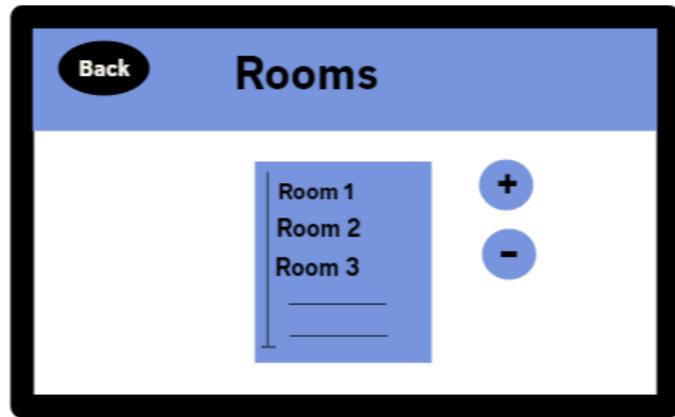
Figur 24: Design for Functions, Hub

Functions på figur 24 giver brugeren et overblik over de eksisterende funktioner samt muligheden for at oprette eller slette gamle eller ny lavet funktioner. Her vises en liste over funktioner, som brugeren kan vælge at redigere for at ændre tilhørende data.



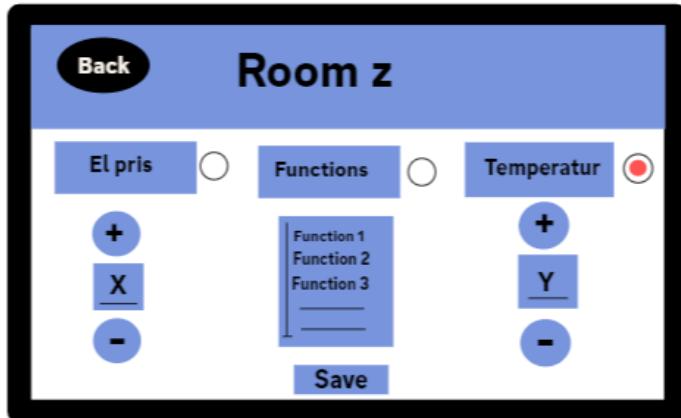
Figur 25: Design for Setting For Function, Hub

Settings for Function på figur 25 tillader brugeren at ændre specifikke paramenter relateret til en valgt funktion. Fra Functions-siden kan brugeren vælge en funktion, som derefter kan redigeres ved at justere ønskede parametre såsom ønsket temperatur og elektricitetsprisloft. Disse ændringer gemmes og sendes til Settings for Room-siden, hvor brugeren kan tildele en funktion til et specifikt rum, med klar paramenter installeret.



Figur 26: Design for Rooms, Hub

Rooms på figur 26 giver brugeren et overblik over de eksisterende rum samt muligheden for at oprette eller slette nye rum. Her vises en liste over rum, som brugeren kan vælge at redigere for at ændre tilhørende data.



Figur 27: Design for Setting For Room, Hub

Settings for Room 27 tillader brugeren at ændre specifikke data relateret til et valgt rum. Fra Rooms-siden kan brugeren vælge et rum, som derefter kan redigeres ved at justere ønskede parametre såsom ønsket temperatur og elpris. På denne side kan brugeren vælge en tidligere oprettet og klar til brug funktion. Dette tillader brugeren at tildele en funktion til det valgte rum uden at ændre i de andre værdier, men i stedet anvende de foruddefinerede data.

Vi har besluttet ikke at gå videre med UC 5, 6 og 9, hvilket inkluderer de tilhørende sider ”Data, Unit,Setting For Unit,” og ”General Settings” på grund af tidsmangel. Selvom vi ikke har tilstrækkelig tid til at gøre disse sider fuldt funktionelle, er de blevet designet og implementeret i frontend-delen.

Disse sider vil derfor være synlige, men de vil ikke have nogen funktionalitet. Desginet af de sider findes i bilag G, I, J, H.

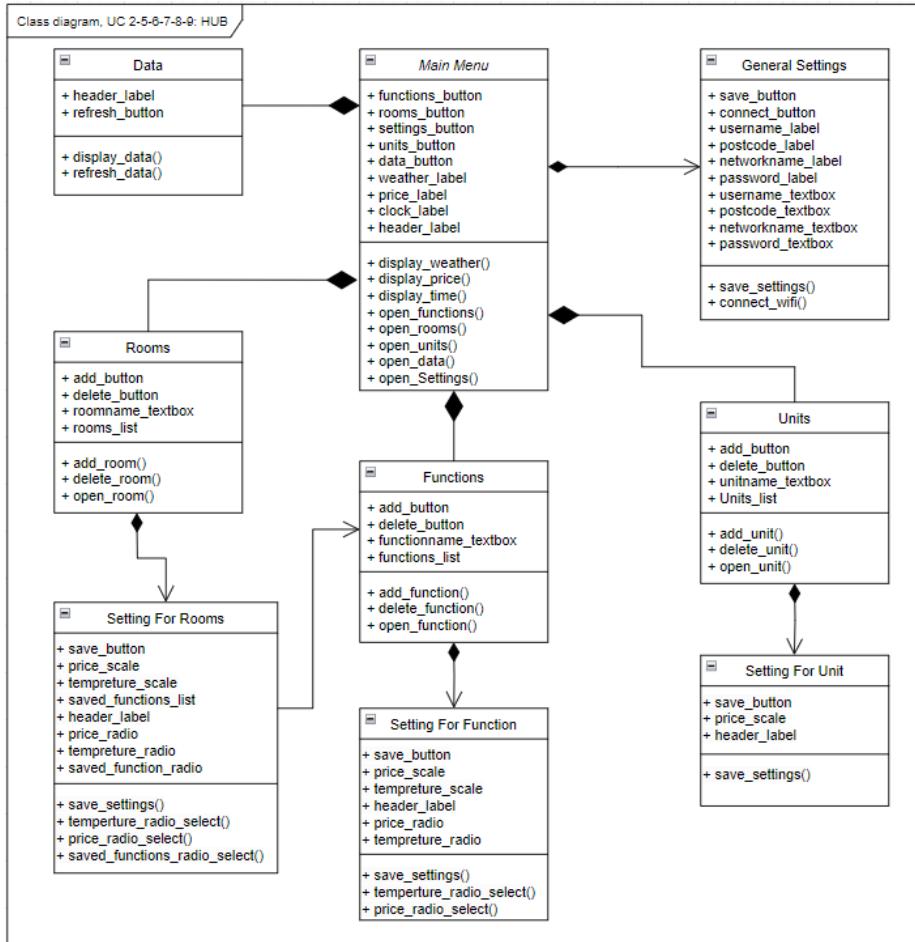
Interface og programmerings sprog:

Vi valgte at arbejde med Python, specifikt er det Tkinter-biblioteket, som er et bibliotek, der er med til at programmer en brugergrænseflade. Desuden er Tkinter kompatibel med platforme som Raspberry Pi 4, hvilket gør det til et ideelt valg til udvikling af interfacet.

GUI-design kræver forskellige elementer for funktionalitet og navigerbarhed. Knapper er essentielle til at navigere mellem sider og udføre handlinger. Tekstfelter, også kendt som Labels i Tkinter, bruges til permanente tekster som overskrifter. En liste giver overblik over rum eller funktioner. På Settings-siden kræves en tekstboks til brugerinput som postnummer eller leverandørnavn. En markør, typisk implementeret med radioknapper, angiver ønskede ændringer eller tilføjelser. Endelig er en skala afgørende for at justere værdier som temperatur og prisloft, implementeret med Tkinter's skalafunktion.

Klasse diagram:

Ud fra use-casene, designet af interfacet, viden om Tkinter og Python, laves et klasse diagram, der kan give et overblik over hvordan, implementering og programmering af interfacet kommer til at være.



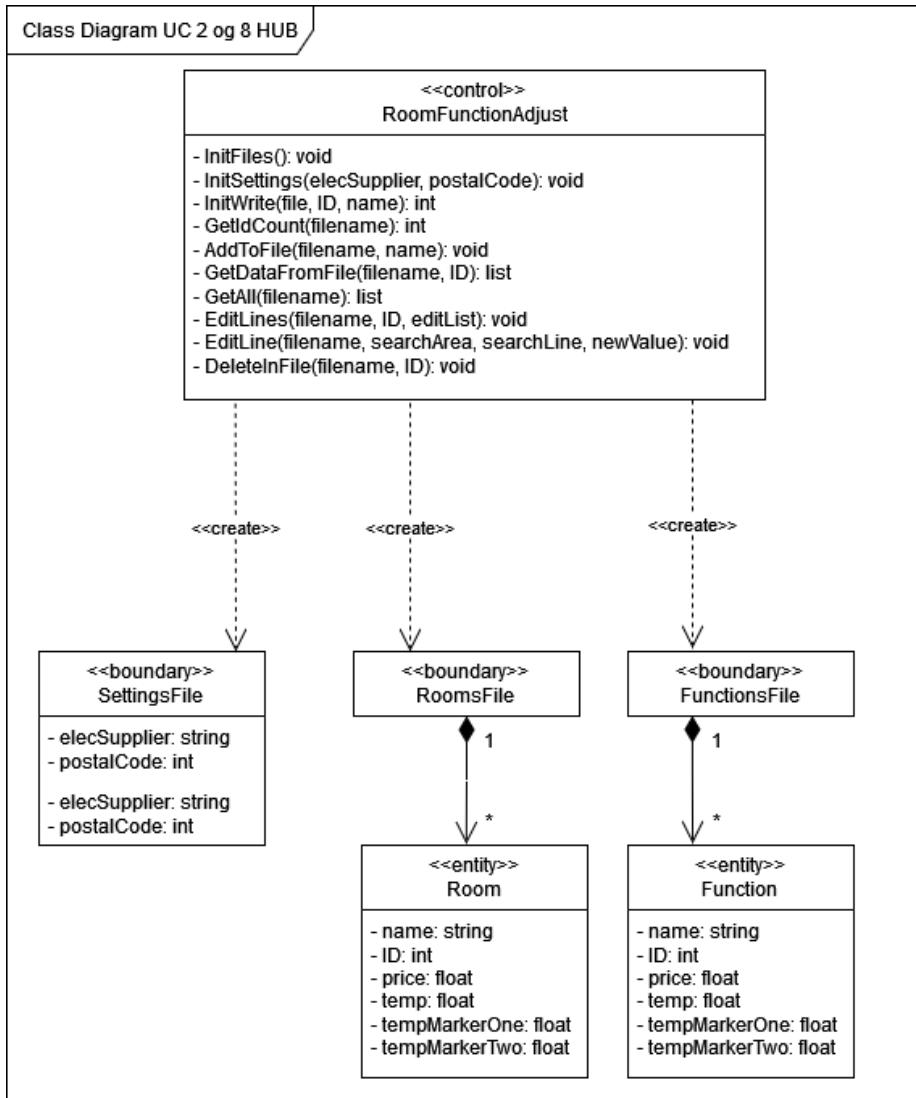
Figur 28: Klasse diagram for interfacet

I figur 28 har siderne *General Setting*, *Units*, *Rooms*, *Functions* og *Data* kompositionerelation med *Main Menu*, fordi siderne ikke kan åbnes hvis ikke *Main Menu* er åben. *Main Menu* har også en assosiationsrelation med *General Setting*, fordi information om leverandøren og postnummer skulle kunne ses fra *Main Menu*. *Rooms* har en kompositions- og assosiationsrelation til *Setting For Rooms*, da dataen for et bestemt rum skal gemmes i det pågældende rum, som man får lavet i listen inde i siden *Rooms*. Det samme gælder for *Functions* og *Units*. Endelig har *Setting For Rooms* en assosiationsrelation til *Functions*, fordi man skal kunne tildele en klar og lavet funktion til et bestemt rum, hvorfra *Setting For Room* har behov for at vide, hvilke funktioner der er til rådighed. Herfra er det næste skridt at få det implementeret.

Backend:

af Filip Alberg

På figur 29 ses et klassediagram over backend delen af interfacet. Det er lavet på baggrund af usecase 2 og 8, hvilke datatyper der er i programmeringssproget Python, samt ECB-klasser, jf. afsnit 5.

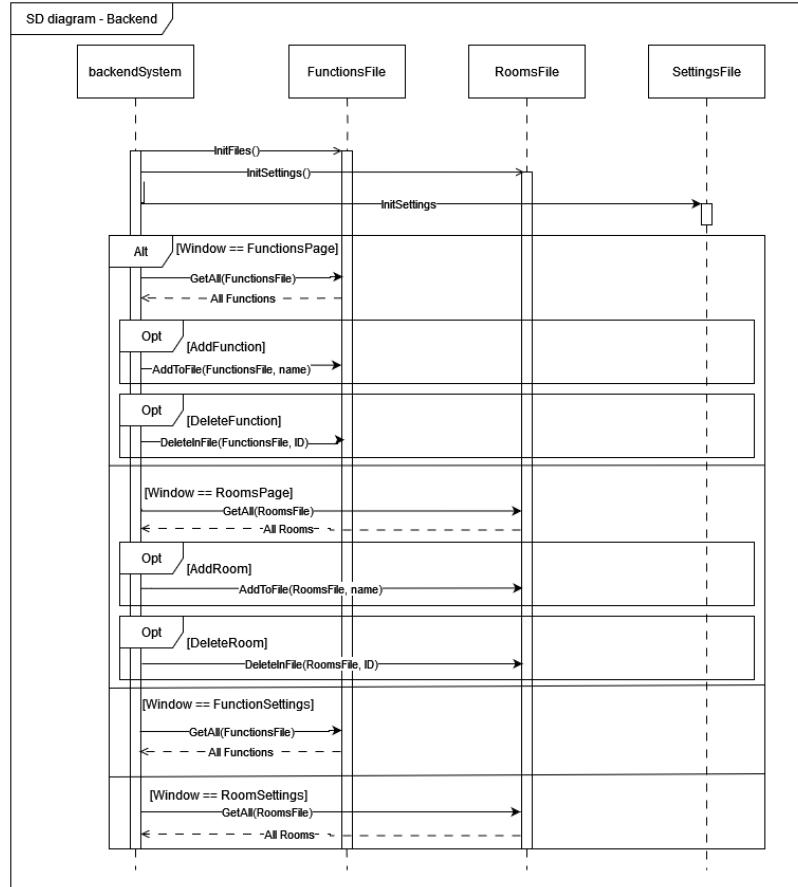


Figur 29: Klassediagram over backend

På klassediagrammet ses control klassen, *RoomFunctionAdjust*, der er selve backend systemet og står for at oprette de nødvendige filer til at lagre data, ændre i filerne, og få noget ud fra filerne i det rigtige format. *Room* og *Function* er hendholdsvis et enkelt rum og funktion i filerne *RoomsFile* og *FunctionFile*. *Room* og *Function* har hver især relationen ”komposition” til *RoomsFile* og *FunctionFile*, da de slettes sammen med filerne, hvis filerne slettes. *RoomFunctionAdjust* har relationen ”dependency” til filerne, fordi *RoomFunctionAdjust* er afhængig af filerne for at kunne gemme parametre, og sende parametrene ud igen.

På figur 30, ses et sekvensdiagram over backend-delen i hub'en. Dette diagram viser den nødvendige kommunikation mellem de relevante klasser fra klassediagrammet, set på figur 29, der skal til for at opnå den funktionalitet beskrevet i usecase 2 og 8. Filerne startes med at initieres/oprettes. Herefter

er det alt efter hvor brugeren er henne på GUI'en der bestemmer hvad der bliver eksekveret. hvis brugeren fx befinder sig på vinduet (Window) FunctionsPage, skal *BackendSystem* eksekvere GetAll der ekstrahere alle elementer fra en given fil. Under de to alts hvor brugeren enten er på FunctionsPage eller RoomsPage, er der to options: Delete og Add som eksekveres når brugeren vælger at tilføje eller slette et element.



Figur 30: SD diagram over backend ud fra UC 2 og 8. vedhæftet som bilag P

7.3.2 Hardware design

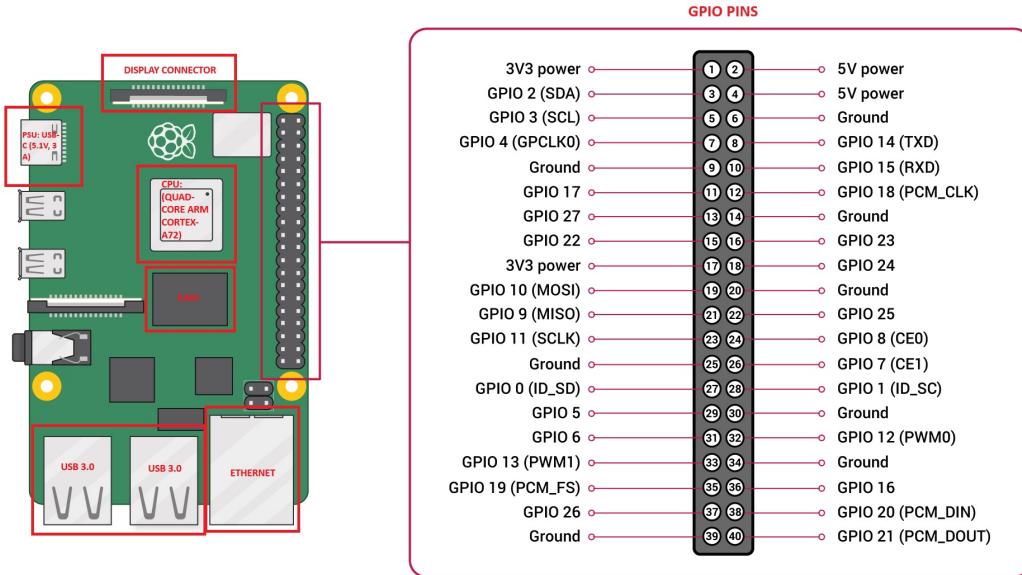
af Simon Eifer

Raspberry Pi 4, Model B:

Raspberry Pi 4 er det mest centrale komponent for hele systemet, da det er den, der agerer bindeleddet mellem bruger og system.

Raspberry Pi 4 er den 4. generation af hovedserien af Raspberry Pi enkelt kortcomputere, hvorpå der er en 40-pin GPIO-header, hvilket sikrer kompatibilitet med de andre delsystemer i det samlede intelligente energistyringssystem, da GPIO-headeren eksemplvist har pins til UART, SPI og I2C

kommunikation.



Figur 31: Diagram over Raspberry Pi 4 hardwarekomponenter
[11]

På figur 31[11] kan man danne sig et overblik over de mest relevante dele af Raspberry Pi 4'erens hardware design. Display connectoren bliver brugt til at forbinde raspberrien med en Raspberry Touchscreen, som er skærmens hvor interfacet kan interagere med brugeren.

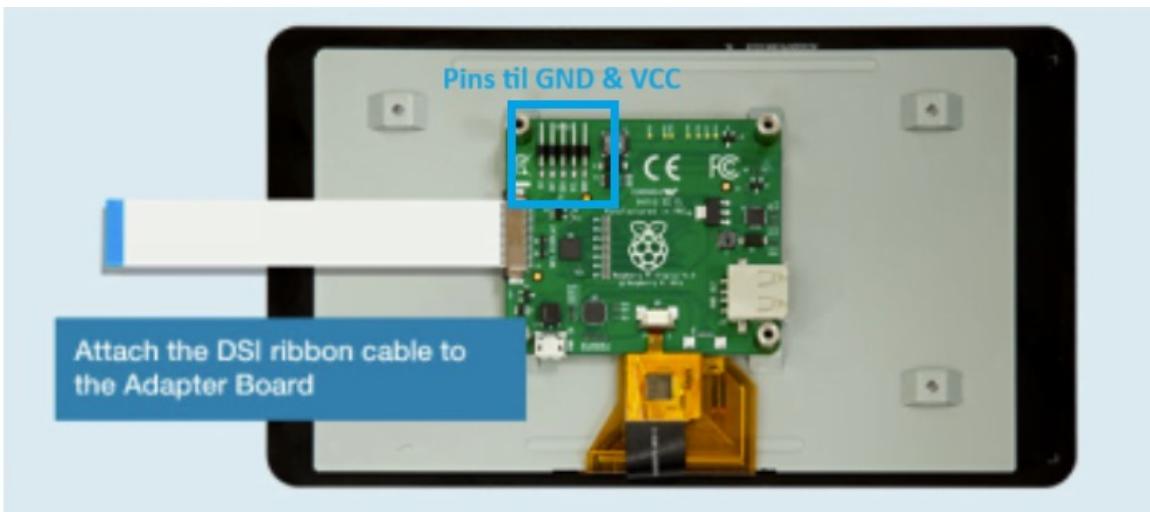
Ud af de mange GPIO-pins bliver pin 4(5V VCC) og pin 6(GND) brugt til at give strøm til skærmen. Pin 8 (UART TX) og pin 10 (UART RX) bliver brugt til at kommunikere med arduinoen der står for temperaturstyringen.

På undersiden af Rapsberry Pi 4 sidder en micro-SD kortlæser, som der er sat et 64 GB micro-SD kort i. SD-kortet bruges til at køre Raspberriens styresystem på (Rapsberry Pi OS), og derudover bruges SD-kortet også som Raspberriens hukommelse. Da selve styresystemet kun fylder ca. 1 GB, har Raspberrien hele ca. 63 GB tilovers til hukommelse. Til dette system, kunne man sagtens have brugt et mindre SD-kort på eksempelvist 16 GB, men da der kun var et kort på 64 GB til rådighed, er det det, der er blevet brugt.

Raspberry Pi Touchscreen, 7. inch:

Raspberry Pi 7"touchscreen er et hardwarekomponent designet til at komplementere Raspberry Pi, som passer systemet godt, i og med at det giver brugeren mulighed for at interagere med systemet, gennem GUI'en, som kører på Raspberry Pi 4 enheden.

Skærmen forbindes til Raspberry Pi ved hjælp af et DSI-fladkabel, der leverer video.



Figur 32: Raspberry Pi Touchskærm bagplade
[10]

Af figur 32[10] fremgår det, hvordan skærmen forbinderes til Raspberry Pi. Det flade kabel er det førnævnte DSI-fladkabel, som forbindes til Raspberriens "Display Connector", som kan ses i figur 31. Derudover bruges to pins på skærmens bagplade til 5V VCC og GND, som den får fra hhv. pin 4 og pin 6 på Raspberry Pi's GPIO pins.

8 Implementering

8.1 Temperatur regulering

af Daniel Naddaf

Koden skrives i det objektorienterede programmeringssprog C++ i Visual Studio Code (version 1.89.1), da denne code editor kan anvendes med utallige udvidelser, herunder specielt PlatformIO[4], som gør det muligt at skrive og sende C++ kode til Mega2560 Arduinoen. Da Arduinoen kun er forbundet til én computer er dette en effektiv måde for flere personer at skrive kode til den samtidigt. Begge udvidelser har præsteret godt og har været de rette valg til de anvendelsesområder, som de var tiltænkt. For at kunne følge med i programmet, og for lettere at kunne debugge ved fejl-kørsler, er koden implementeret med fejlmeldelser og statusbeskeder, som sendes på Arduinoens Serial-port. Det er denne port, som kan sende og modtage beskeder via UART på Arduinoens USB-B port. Dertil anvendes Arduinoens Serial1-port til kommunikationen med HUB'ens Raspberry Pi. Al seriell kommunikation anvender Stream-klassen, og herunder Serial-klassen, fra Arduino.h-headeren[5]. Disse klasser sørger for al logikken bag UART-kommunikation, da disse er skræddersyede af Arduino til bl.a. Mega2560 Arduinoen, som anvendes i dette delsystem. Alle koden til dette delsystem findes i bilag V.

Arduinoen blev forbundet til LM75 temperatursensoren gennem en I2C forbindelse, som angivet i figur 19 i designfasen. Implementeringen af I2C kommunikationen er uddybet i afsnit 8.1.1. I programmet vil enhver aktivering af varmelegemet sende et højt signal til Port 13, som bør gå til MOSFET'ens *gate* ben, sådan at et højt eller lavt signal vil hhv. åbne eller lukke for strømtilførslen

fra *source* til *drain* på MOSFET'en. Da der kun arbejdes med en prototype til test- og demonstrationsformål, blev der monteret en LED i stedet for en MOSFET, for visuelt at indikere hvornår varmelegemet er tændt.

8.1.1 I2C/TWI kommunikation

af Jonas Kirkegaard Skyum

I projektet er der valgt at anvende TWI (Two-Wire Interface) / I2C (Inter-Integrated Circuit) kommunikation på grund af dens mange fordele. TWI/I2C gør det muligt at forbinde flere enheder med kun to ledninger (SDA og SCL), hvilket reducerer kompleksiteten hardwaredesignet. Den understøtter flere master- og slave-enheder på samme bus, hvilket giver fleksibiliteten til nemt at udvide systemet med flere sensorer og moduler til en potentiel fremtid. Desuden er TWI/I2C en standardiseret protokol, hvilket sikrer kompatibilitet med et bredt udvalg af komponenter såsom temperatursensoren LM75, der er anvendt her. Protokollen er også pålidelig med indbyggede funktioner, og gør det derfor simpelt at implementere.[9]

8.1.2 Reflektering over implementering

af Daniel Naddaf

Eftersom temperaturen hentes fra sensoren over en I2C forbindelse, er systemet begrænset af I2C forbindelsens korte rækkevidde på omkring 2,5-3,5 meter[12]. Dette er en klar begrænsning af systemet og placeringen af Arduinoen, som skal kunne hente temperaturer i flere rum. Der er flere metoder, som kan slås sammen, ikke nødvendigvis to-og-to, men logikken i enkelte metoder vil kunne opløses og uddelegeres til andre metoder. Der er dog rettet og optimeret i programmet undervejs, så den nuværende version er stadig tilfredstillende, især fra et funktionelt perspektiv, da programmet gør, som det skal. Specielt har designet af hovedalgoritmen været tilfredsstillende, da denne tager højde for mange forskellige faktorer, og sørger for også at inkludere dataen fra de andre delsystemer i overvejelserne. Eksekveringen af denne har næsten præsteret som den skulle, ud over et problem med timingen, som beskrives yderligere i diskussionen.

8.2 HUB

Implementeringsdelen fokuserer på udviklingen af en brugervenlig hub. Der er lagt vægt på designfasen og implementering af GUI-elementer som knapper og lister for at skabe en responsiv brugergrænseflade. Backend-delen, der håndterer dataene fra frontend-interfacet, diskuteres for at give et helhedsbillede af udviklingsprocessen. Implementeringen på Raspberry Pi 4 og den tilhørende skærm præsenteres også. Målet er at give et kort indblik i udviklingen af både frontend, backend og implementering på RPi.

8.2.1 Frontend:

af Khaled Omar

Denne rapportsektion fokuserer på udviklingen af frontend-interfacet til softwareapplikation. Frontend-interfacet er afgørende, da det er brugerens primære interaktionspunkt med systemet.

Valg af programmeringsprog:

Valget af Tkinter som værktøj til udviklingen af vores frontend-interface blev motiveret af flere faktorer. For det første er Tkinter kendt for at være et nemt at lære og bruge grafisk brugerfladebibliotek

til Python. Dets enkle syntaks og intuitive natur, som også ses i python gjorde det til et ideelt valg for vores udviklingsteam, hvilket muliggjorde en hurtig og effektiv oprettelse af GUI-elementer. Et yderligere incitament til valget af Tkinter var dets kompatibilitet med Raspberry Pi 4-enheder og tilhørende touchskærme. Da vores softwareapplikation var designet til at køre på Raspberry Pi 4, valgte vi Tkinter som GUI-bibliotek på grund af dets evne til at fungere sømløst med denne hardwarekonfiguration.

Opsætning:

For at programmere med Tkinter skal biblioteket installeres. Biblioteket *requests*, til API'er for el-pris og vejrudsigt, installeres også. Biblioteket *time*, som er integreret i Python, bruges til at opsætte et ur og en timer, der opdaterer API'erne hver time.

Farver og teksttype:

De defineres som variabler i koden og anvendes forskellige steder i applikationen.

```
# Colors
bg_color = "#A6E3E9"
header_color = "#00ADB5"
Button_color = "#CBF1F5"
text_color = "black"
```

Figur 33: Farver for interface

```
38 |     # Fonts
39 |     ft10 = tkFont.Font(family='Times', size=10, weight='bold')
40 |     ft20 = tkFont.Font(family='Times', size=20, weight='bold')
41 |     ft38 = tkFont.Font(family='Times', size=38, weight='bold')
42 |     ft13 = tkFont.Font(family='Times', size=13, weight='bold')
```

Figur 34: Fonts for interface

For at registrere en farve angives enten hex-koden eller en standardfarve som ”black”. For tekst specificeres skrifftype, størrelse og vægt.

Klasser:

Der startes med at, oprette klasserne ud fra figur 28, og bygger siderne op til hvert side.

```

> class InterFase: ...
> class FunctionsPage: ...
> class FunctionsSettings: ...
> class RoomsPage: ...
> class RoomsSettings: ...
> class UnitsPage: ...
> class UnitsSettings: ...
> class DataPage: ...
> class SettingsPage: ...

```

Figur 35: Klasser for Interface

Størrelse og placering af interface:

```

252     # Set the size of the window to 800x480 pixels
253     width = 800
254     height = 480
255     # Get the screen width and height
256     screenwidth = self.root.winfo_screenwidth()
257     screenheight = self.root.winfo_screenheight()
258     # Calculate the position of the window to center it on the screen
259     alignstr = '%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2, (screenheight - height) / 2)
260     # Apply the calculated size and position to set the geometry of the window
261     self.root.geometry(alignstr)
262     # Make the window non-resizable in width and height
263     self.root.resizable(width=False, height=False)

```

Figur 36: Vindue konfiguration for interface

Koden er udviklet til at konfigurer parametre for et applikationsvindue, herunder titel, dimensioner, positionering og størrelsejustering. Først indstilles vinduets bredde og højde baseret på skærmens specifikationer, konfigureret til en Raspberry Pi 4. Derefter beregnes vinduets placering for at centrere det på skærmen. Til sidst deaktiveres ændring af vinduets dimensioner, hvilket sikrer en fast størrelse og position.

Klokke:

```

50     # Clock Label
51     Clock_Label = tk.Label(root, font=('Times', 40, "bold"), fg="#000000", bg=bg_color, justify="center")
52     Clock_Label.place(x=260, y=225, width=280, height=100)
53     # This function is used to display time on the label
54     def display_time():
55         string = strftime('%H:%M:%S')
56         Clock_Label.config(text=string)
57         Clock_Label.after(1000, display_time)
58     # Styling the label widget so that clock will look more attractive
59     display_time()

```

Figur 37: Klokke for interface

Koden er opdelt i to sektioner. Den første sektion, ”Clock Label,” viser tiden på hovedmenuen og definerer labelens farve, placering, skriftype og størrelse. Den anden sektion håndterer uret ved at bruge Pythons time-bibliotek. Funktionen strftime fra dette bibliotek viser tiden baseret på syste-

mets lokale tid.

Knapper:

```
97     # Functions
98     Functions_Button = tk.Button(root, text="Functions", font=ft20,
99     |           fg=text_color, bg=Button_color, justify="center", activebackground=Button_color)
100    Functions_Button.place(x=100, y=130, width=150, height=100)
101    Functions_Button["command"] = self.open_functions # Bind button command to the method in InterFase
102
103   def open_functions(self):
104       # Open a new window for Functions
105       functions_window = tk.Toplevel(self.root)
106       # Create the FunctionsPage instance within the new window
107       functions_page = FunctionsPage(functions_window)
```

Figur 38: Knapper for Interface

Koden for knapper er opdelt i to sektioner. Den første sektion opretter knappen med de ønskede farver, skriftype, størrelse og placering. Den anden sektion definerer knapfunktionen. I dette eksempel åbner knappen siden ”Functions” fra ”Main Menu”. En anden eksempel på knap er ”Gem”, som bruges til backend-delen.

Lister:

```
# Create a listbox widget within the root window
self.listbox = tk.Listbox(root, bg=Button_color, fg=text_color)
# Pack the listbox with some padding
self.listbox.pack(pady=10)

# Insert some initial items into the listbox
initial_items = ["Night Function", "Morning Function"]
for item in initial_items:
    self.listbox.insert(tk.END, item)
```

Figur 39: Lister for Interface

Kode for liste opdeles i to sektioner: oprettelse og definition af indhold. Siderne ”Functions”, ”Rooms” og ”Units” følger samme mønster, hvor brugere kan tilføje eller fjerne elementer og åbne specifikke elementer. Backend er ikke udviklet, så standard elementer som morgen- og aftensfunktioner tilføjes til ”Functions”, for at illustrere funktionaliteten. Listen giver brugere en intuitiv måde at administrere og kontrollere elementer.

```

208     # Create an entry widget for user input
209     self.entry = tk.Entry(root, width=30, bg=Button_color, fg=text_color,)
210     # Pack the entry widget with some padding
211     self.entry.pack(pady=5)
212     # Create an "Add" button to add items to the listbox
213     self.add_button = tk.Button(root, text="Add", command=self.add_item,
214     |     |     fg=text_color, activebackground=header_color, bg=Button_color)
215     # Pack the "Add" button to the left with some padding
216     self.add_button.pack(side=tk.LEFT, padx=5)
217     # Create a "Delete" button to delete selected items from the listbox
218     self.delete_button = tk.Button(root, text="Delete", command=self.delete_item,
219     |     |     fg=text_color, activebackground=header_color, bg=Button_color)
220     # Pack the "Delete" button to the left with some padding
221     self.delete_button.pack(side=tk.LEFT, padx=5)

```

Figur 40: Indstillinger til listen for interface

Brug af en tekstboks (Entry) med ”Add” og ”Delete” knapper forbedrer brugeroplevelsen ved at tillade brugere at tilføje og redigere funktioner efter behov.

Radio og tilhørende skala'er:

Radio knapper er med til at bestemme hvilke af pris eller temperatur man ønsker at ændre i, der ændres også indstillinger for placering, farve, størrelse osv.

```

329     # Variable to control the radio buttons
330     self.radio = tk.IntVar()
331     # Radio buttons
332     Price_Radio = tk.Radiobutton(root, text="Price ₦", font=ft, fg=text_color, bg=bg_color,
333     activebackground=bg_color, variable=self.radio, value=1, command=self.Price_Radio_command)
334     Price_Radio.place(x=100, y=160, width=150, height=50)
335     Temp_Radio = tk.Radiobutton(root, text="Temperature 🌡", font=ft, fg=text_color, bg=bg_color,
336     activebackground=bg_color, variable=self.radio, value=2, command=self.Temp_Radio_command)
337     Temp_Radio.place(x=550, y=160, width=150, height=50)
338     # Temp radio on as default
339     self.radio.set(2)

```

Figur 41: Radio knapper for interface

Radio-knapper bruges til at vælge mellem pris eller temperatur, og brugeren kan ændre indstillinger som placering, farve og størrelse. Hver knap har skalaer, der justerer prisen eller temperaturen. Der er tre skalaer for temperatur, hvoraf to blev tilføjet senere og ikke er med i klasse-diagrammet, da algoritmen for temperaturregulering ikke var fastlagt. Implementeringen indebærer tildeling af værdier til knapperne via en radio-variabel, hvilket skaber to grupper af knapper. En standardradio sættes for at undgå, at ingen knap er valgt, hvilket ellers ville tillade ændringer i både pris og temperatur uden specifikation.

```

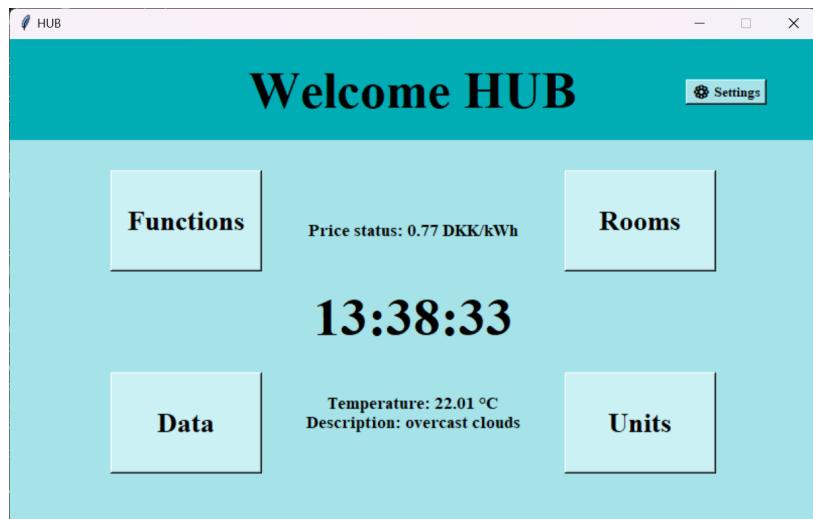
308     # Scales
309     self.price_scale = tk.Scale(root, from_=0, to=5, orient="vertical", bg=bg_color, activebackground=bg_color,
310     highlightbackground=header_color, troughcolor=header_color, width=20, sliderlength=30, resolution=0.01)
311     self.price_scale.set(1)
312     self.price_scale.place(x=150, y=220, height=200)
313
314     self.temp_scale = tk.Scale(root, from_=-30, to=60, orient="vertical", bg=bg_color, activebackground=bg_color,
315     highlightbackground=header_color, troughcolor=header_color, width=20, sliderlength=30, resolution=0.1)
316     self.temp_scale.set(22)
317     self.temp_scale.place(x=520, y=240, height=180)
318
319     self.Acceptable_scale = tk.Scale(root, from_=-30, to=60, orient="vertical", bg=bg_color, activebackground=bg_color,
320     highlightbackground=header_color, troughcolor=header_color, width=20, sliderlength=30, resolution=0.1)
321     self.Acceptable_scale.set(18)
322     self.Acceptable_scale.place(x=580, y=260, height=160)
323
324     self.Minimum_scale = tk.Scale(root, from_=-30, to=60, orient="vertical", bg=bg_color, activebackground=bg_color,
325     highlightbackground=header_color, troughcolor=header_color, width=20, sliderlength=30, resolution=0.1)
326     self.Minimum_scale.set(14)
327     self.Minimum_scale.place(x=640, y=290, height=130)

```

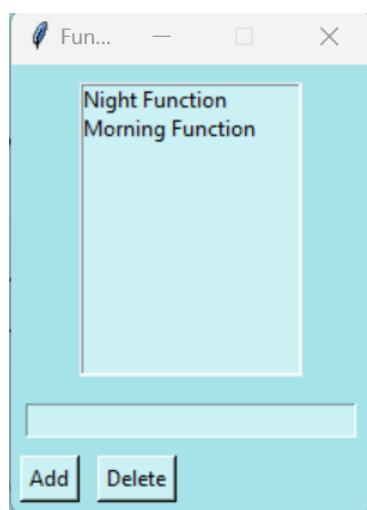
Figur 42: Skala for interface

Radio-knapper bruges til at vælge mellem pris eller temperatur, hvor brugeren kan ændre indstillinger som placering, farve og størrelse. Implementeringen tildeler værdier til radio-knapperne via en radio-variabel, hvilket skaber to grupper af knapper. En standardradio undgår, at ingen knap er valgt, hvilket ellers ville tillade ændringer uden specifikation. I ”Setting for Function” er der to radio-knapper, mens der er tre i ”Setting For Room”. Den tredje knap i ”Setting For Room” registrerer brugerens gemte funktioner.

Resultat: Alt ovenstående er koder og elementer, der genbruges flere gange, men på forskellige måder og forskellige steder. Dette har været med til at implementere og fuldføre interfacet i overensstemmelse med det ønskede design. Implementering af siderne General Settings, Units, Setting For Units og Data er lavet ift. frontend-delen, for at illustrerer hvordan det vil have set en fuld fungerende system, og de findes i bilag K, L,M og N. Resten af koden findes i bilag Q



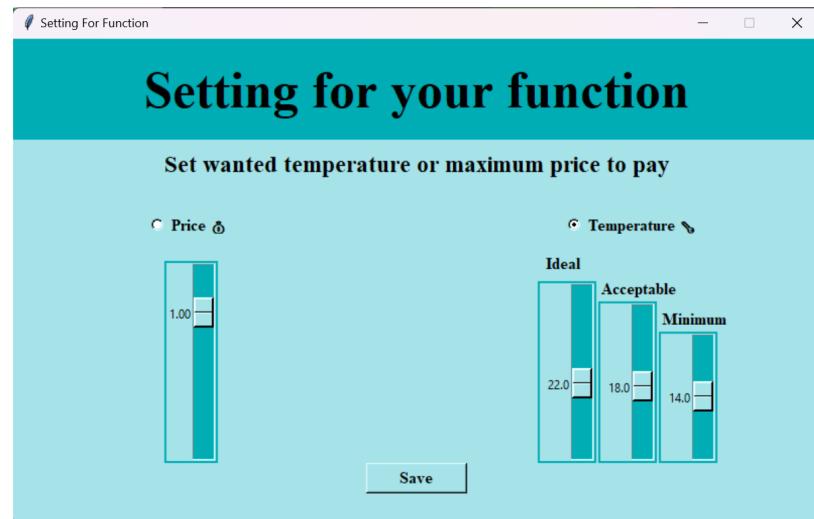
Figur 43: Implementering for Main Menu, Hub



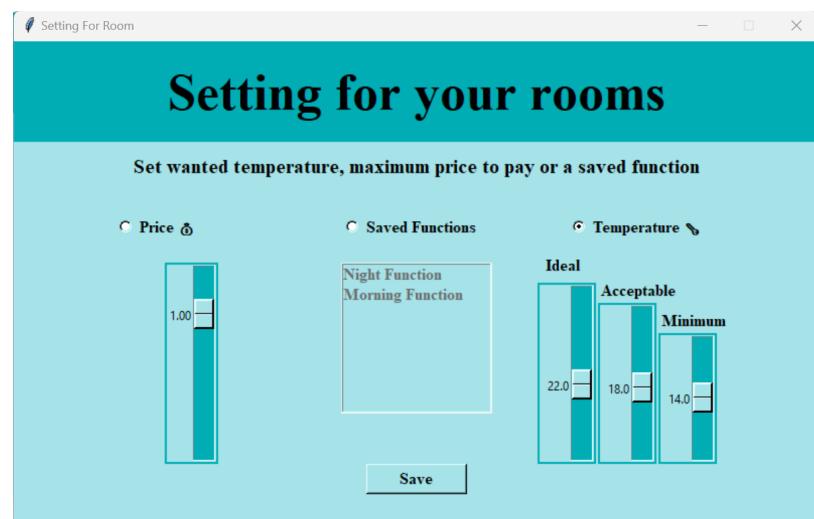
Figur 44: Implementering for Functions, Hub



Figur 45: Implementering for Rooms, Hub



Figur 46: Implementering for Setting For Function, Hub



Figur 47: Implementering for Setting For Room, Hub

8.2.2 API

af Khaled Omar & Simon Eifer

Integration af API'er til at hente energipriser og vejrdata var en vigtig del af udviklingen. Der implementeres to API'er som en del af vores GUI: en til at hente energipriser og en til at hente vejrdata. I dette tilfælde hentes data fra "Energidataservice.dk" og "Openweathermap.org", som løbende opdaterer databaserne med nye energipriser og vejrdata.

Energidataservice API'en bliver brugt som en central del af hele systemet. Den bruges, som tidligere nævnt, når brugeren indstiller et prisloft. Dette afgør, om varmereguleringen skal starte (givet grønt lys) eller stoppe (givet rødt lys), baseret på om den nuværende energipris er højere eller lavere end det prisloft, som brugeren har valgt at indstille.

I figur 48 er der brugt python-biblioteket "requests" til at hente API'en. Requests gør det nemt at sende HTTP/1.1 forespørgsler. Der er ikke behov for manuelt at tilføje query strings til dine URL'er eller at form-kodificere dine PUT og POST data — men der bruges blot json-metoden. Med andre ord, kan det bruges til at hente API'er på en nem måde.

```
def fetch_and_display_price(self):
    # Definer API-endepunktet for Elspotprices
    url = "https://api.energidataservice.dk/dataset/Elspotprices"

    # Parametre for forespørgslen
    params = {
        'filter': '{"PriceArea": "DK1"}',
        'limit': 1  # Henter kun den seneste post
    }

    # Udfør HTTP GET-forespørgslen
    response = requests.get(url, params=params)

    # Tjek om forespørgslen var vellykket
    if response.status_code == 200:
        data = response.json()
        if 'records' in data and len(data['records']) > 0:
            latest_price_data = data['records'][0]  # Den seneste post for DK1
            # Konverter prisen til kWh og afrund til to decimaler
            price_per_kwh = round(float(latest_price_data['SpotPriceDKK']) / 1000, 2)
            price_status = f" Price status: {price_per_kwh} DKK/kWh "
        else:
            price_status = "Ingen data fundet for DK1."
    else:
        price_status = "Forespørgslen mislykkedes"

    # Update the label text with the price status
    self.ELPrice_Label.config(text=price_status)
```

Figur 48: Energidataservice.dk API, kode

Koden i figur 48, indeholder parametre til at præcisere hvilke data der skal hentes, f.eks. et filter der

begrænser prisdata til energipris-området DK1 i Danmark. I Danmark er der to energipris-områder. DK1 er Jylland og DK2 er øerne (Fyn, Sjælland, Lolland-Falster, osv.) I dette tilfælde er det mest relevant kun at hente prisdata for DK1.

Det andet API i hovedmenuen fokuserer udelukkende på vejrudsigten og opdateres også hver time, ligesom elpris-API'en. Disse oplysninger bidrager til at skræddersy informationen efter behov og sikre aktuelle data til brugeren.

```
def fetch_and_display_weather(self):
    # Get weather data from API
    weather_data = requests.get("https://api.openweathermap.org/data/2.5/weather",
                                params={'lat': 56.162937, 'lon': 10.203921, 'appid': '4d7dad712e7ed35a0f7bd672abbea9a3', 'units': 'metric'}).json()

    # Extract relevant weather information
    temperature = weather_data['main']['temp']
    description = weather_data['weather'][0]['description']

    # Display weather information
    weather_info = f"Temperature: {temperature} °C\nDescription: {description}"
    self.Weather_Label.config(text=weather_info)

    # Schedule the function to run again after 3600 seconds (1 hour)
    self.root.after(3600 * 1000, self.fetch_and_display_weather)
```

Figur 49: API for vejrudsigt

For at få adgang til denne API skal du have en konto hos ”Openweathermap.org” og indhente en API-nøgle, kaldet ”appid”. Du skal også angive en specifik lokation ved hjælp af breddegrad, ”lat”, og længdegrader, ”lon”, hvor vi i dette tilfælde bruger Aarhus’ koordinater. For at sikre, at temperaturvisningen er i Celsius, skal du indstille parameteren ”units” til ”metric”. Der skal derefter implementeres visningselementer i hovedmenuen for vejrinformationer. Endelig oprettes en timer for at sikre, at vejroplysningerne opdateres automatisk hver time.

8.2.3 Backend

af Filip Alberg

Udover en frontend del, skal der også være en backend del der håndterer dataen der kommer fra frontend-delen. Der skal være mulighed for at gemme data, der kommer fra frontend-delen, så dataen altid kan hentes, både af frontend, men også af andre dele af systemet. Dette sørger backend-delen for, samt håndtering af *Business Logic* af hub'en og CRUD (Create, Read, Update and Delete). Til implementering af backend og lagring er der ikke brugt eksterne libraries, men ren python. Python er et godt sprog at bruge, fordi det giver udvikleren øget produktivitet fordi det er et ret simpelt sprog i forhold til andre sprog på markedet, og dette skærer godt ned på produktionstiden. Python er også et nemt sprog at lære, så hvis der er nogle nye funktioner der skal bruges, er det ikke tidskrævende. Derudover, siden der allerede er erfaring i sproget, var det et oplagt valg, så det ikke var nødvendigt at bruge tid på at lære noget nyt.

Der er ikke valgt eksterne libraries, da den nødvendige funktionalitet allerede findes indbygget i python, og ville ellers give unødig fylde i koden og gøre den langsommere, hvis et library, specifikt til dette, blev brugt. Som en forlængelse af dette er der brugt lagring i form af filer. Dette er fordi, det er relativt nemt at sætte op i python, og fordi det ikke kræver så meget ekstra tid at finde ud

af, hvilket betyder, at det blev færdigt til tiden. Filer er også gode, fordi filer ikke rigtig kan lukke ned, og ikke være tilgængelige, fordi de ligger lokalt på systemet. Dette gør at systemet hele tiden har adgang til brugerens indtastede data om rum og funktioner.

En mulighed at bruge i stedet for filer til lagring, er at bruge en database, men ulempen ved dette er at databasen kan gå offline, og brugeren derfor ikke har adgang til deres data. Fordelen ved en database er dog at den er meget mere struktureret og øget sikkerhed. I dette tilfælde er det dog bedre at bruge filer, da det ikke er store mængder data der skal gemmes pr. bruger. Filer er også meget lette at oprette, skrive og læse til i python, og eftersom der allerede er viden indenfor python, kræver det kun en smule research.

Måden filer laves i python, er ved brug af den indbyggede funktion *open*. *Open* er en funktion der gør det muligt at åbne filer og kommunikerer med dem, enten i form af at skrive til dem, eller læse fra dem. Det er også muligt at oprette filer ved brug af *open*. Kort sagt returnerer *open* et fil objekt, baseret på filen som *open* bliver givet, hvor dette fil objekt har nogle metoder der gør det muligt at manipulerer filen.[13]

Backendsystemet blev integreret på GUI'en ved at det er en fil der importeres i GUI'en, og GUI'en her kalder de funktioner der er defineret i backendsystemet. Dette vil sige at hvor fx der tilføjes et rum i GUI'en kaldes funktionen fra backendsystemet der gemmer rummet til en fil.

8.2.4 Raspberry Pi 4 og Raspberry 7"Touchscreen

af Simon Eifer

For at implementere Raspberry Pi 4 som en interaktiv enhed med en Raspberry Pi 7"touchscreen blev Raspberry Pi OS installeret.

Efter installationen af Raspberry Pi OS blev Python's pakkehåndteringssystem PIP. For at sikre, at alle nødvendige Python-biblioteker var til stede, blev blandt andet "tkinter" og "requests" installeret

Python er et godt valg af sprog mht. Raspberry Pi, da man i Python kan drage nytte af mange biblioteker, eksempelvist tkinter, til at lave GUI, og pyserial, til at lave integration med andre microkontrollere, så som Mega2560 arduinoen. Derudover er der i Raspberry Pi OS indbygget støtte af netop Python, som gør det nemt at hurtigt implementere python-kode på enheden.

Med de nødvendige softwarekomponenter på plads blev GUI-applikationen kørt på enheden. Koden til GUI'en blev skrevet i Python ved hjælp af tkinter, for at skabe en brugervenlig grænseflade. Raspberry Pi 7"touchscreen blev derefter tilsluttet Raspberry Pi 4 via DSI-porten, korrekt monteret og testet for at sikre, at touch-funktionen fungerede.

8.3 ECC / Elmåler

af Jakob & Nichlas

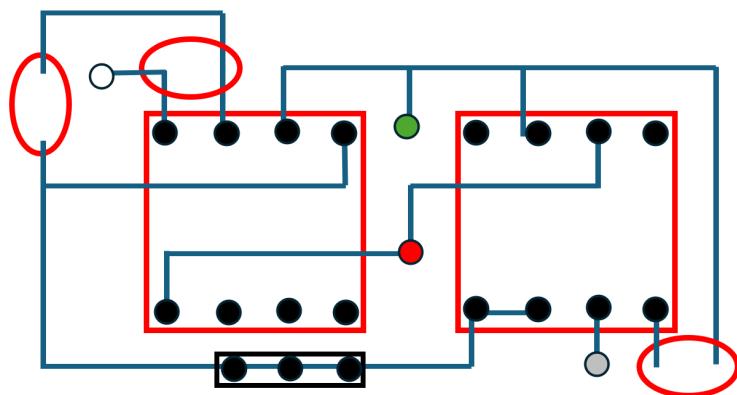
8.3.1 Software implementation

Arduino IDE blev valgt som udviklingsplatform, da den er specielt designet til Arduino og derfor er meget kompatibel med microcontrolleren. Serial monitoren gør det muligt at tracke data i realtid og debugge programmet effektivt.

Selvom Arduino IDE ikke understøtter Live Share eller andre let tilgængelige realtids samarbejdsfunktioner, blev koden skrevet med medlemmer til stede, hvilket gjorde samarbejde, debugging og test lettere. I Arduino IDE skriver i C++ som primært kodesprog, hvilket passer godt da dette er gruppens mest erfarne sprog at kode i. Som nævnt i afsnit 7.1.3 vil koden for elmåleren bestå af en enkelt klasse, men skrives i C++, som er kompatibelt med Arduino IDE.

8.3.2 Hardware implementation

Hardwareimplementeringen er valgt at starte på et fumlebræt, da dette er en sikker og let måde hvorpå at arbejde med kredsløb. Når så opsætningen var korrekt og hardwaren virkede blev det derefter overført til et veroboard og loddet. Systemet blev testet med Analog Discovery 2 for at sikre korrekt spænding og LED-funktion. Dette design gjorde det muligt for Arduinoen at tælle lyspulserne og omregne dem til frekvens for præcis energiorvægning.



Figur 50: Top-Down design af kredsløb. Med forklaring i bilag, ECC

9 Test

9.1 Modultest

9.1.1 Temperature regulation

af Daniel Naddaf og Jonas Kirkegaard Skyum

LM75 modultesten har til formål at verificere funktionaliteten af LM75 temperatursensoren, som kommunikerer via I2C-protokollen. Testen skal sikre, at sensoren korrekt kan foretage temperaturmålinger og sende disse data til en mikrocontroller. I følge kravspecifikationerne skulle temperatursensoren kunne måle temperaturer i intervallet 0°C - 30°C . For at teste det varmes og køles temperatur sensoren op ved at holde henholdsvis en finger og en ispind på LM75'eren. Det blev observeret på en seriell monitor på PlatformIO[4], at der blev udskrevet en temperatur fra temperatursensoren, og at temperatursensoren kunne måle i intervallet 0°C - 30°C . denne temperatur blev sammenlignet med et termometer, og det kunne konkluderes, at temperaturen havde en afvigelse på $\pm 3^{\circ}\text{C}$.

Kommunikationsprotokollen for dette delsystem blev testet ved at sende forskellige signaler til Mega2560 microcontrolleren fra en PSOC 5LP[7]. PSOC-programmet blev skrevet i C via PSOC Creator 4.4. PSOC-programmet sendte to signaler, der hver immitterede *initiate*-signalet og *regulationData*-signalet fra HUB'en. Signalerne blev indrammet som beskrevet i arkitektur-afsnittet for protokollen, afsnit 6.2.4, og på arduino-siden blev signalerne afkodet og verificeret med count- og checksum-bytes. Disse to signaler blev sendt skiftevis med en ventetid på 10 sekunder mellem hvert signal. Hele arduinoDriver klassen i Arduinoen blev testet grundigt igennem, ved at sende forskellige signaler fra PSOC'en, hvor der blev ændret på én ting af gangen, og derefter følge med i monitor-outputtet fra en computer. Der blev sendt forkerte *type*-bytes, forkerte checksum- og count-værdier, for få og for mange bytes og forkerte *roomId*-værdier. Der var en del uønskede og ikke-planlagte fejl i de første kørslер, men disse blev rettet, og gav et indblik i andre uforudsete fejl som kunne opstå - disse blev også rettet. Den endelige driver opfører sig præcis, som den skal.

Hovedalgoritmen og programmets andre metoder blev testet sammen idet hovedalgoritmen anvender alle klasserne i programmet, enten direkte eller indirekte. Alle metoder bliver altså brugt, ved at køre de mulige situationer i hovedalgoritmen. Ingen er der anvendt en PSOC 5LP til at immitere signalerne, som sendes fra HUB'en. Signalerne blev tilpasset sådan, at de forskellige scenarier, som hovedalgoritmen er inddelt i, blev testet. Ved at sende et prisloft, som er højere end den sendte aktuelle pris, blev det testet hvorvidt opvarmningen gik i gang. Selve opvarmningen blev indikeret af en LED, som lyser rødt, samt en besked, som sendes til en terminal på computeren, gennem UART. Ved at sende et lavere prisloft end den sendte aktuelle pris, blev det testet om programmet fortsatte korrekt til næste forgrenning af muligheder. Forgreningerne fremgår af STM diagrammet i figur 21, og disse blev alle testet ved at tilpasse signaler til de forskellige scenarier, og sende dem til Arduinoen. I alle tilfælde gjorde programmet det, som det skulle. Det blev også testet om programmet ventede den korrekte mængde tid ved den sidste forgrenning. Denne sidste test gik ikke som forventet. Selvom programmet ventede, når det skulle vente, var tiden ikke korrekt og meget utilregnelig. På grund af tidsmangel blev denne fejl ikke udbedret.

9.1.2 HUB

Af Filip Alberg & Khaled Omar & Simon Eifer

Backendsystemet blev testet ved at give det nogle forskellige testværdier for at vise at backend delen kan tage imod disse specifikke værdier. Disse testværdier, bliver givet til backendsystemet i form af lister der simulerer data i et element (rum/funktion) som brugeren gerne vil have ændret til. Der er også enkeltværdier som testværdier, herunder navne på elementer, der bruges når elementer oprettes.

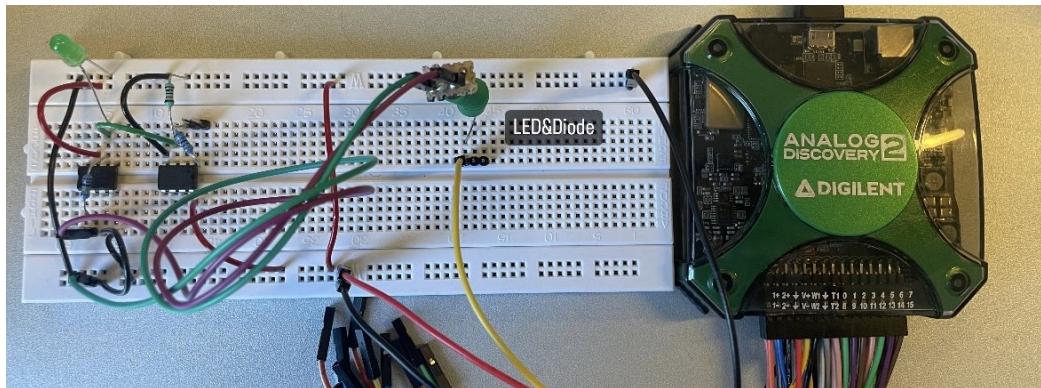
En anden test af backendsystemet, er hvor funktionerne i systemet bliver kaldt i bestemte rækkefølger, for at simulere en bruger der fx tilføjer og derefter ændrer i et element. Dette er til for at se om der er bugs, der gør om brugeren fx ikke kan tilgå et element efter at have slettet et element. Under denne test vil funktionerne til initiering af filerne altid være de første der eksekveres, herefter er det forskelligt hvilken rækkefølge funktionern fra backendsystemet eksekveres i.

Frontend systemet blev testet ved at køre programmet på Raspberry Pi og tjekke efter om alle implementerede funktionaliteter fungerede korrekt. Eksempelvist at API'erne virkede efter hensigt og at de forskellige "knapper" i GUI'en, så som "functions" kunne åbnes og blev vist korrekt.

9.1.3 ECC / Elmåler

af Jakob & Nichlas

Modultesten for systemet er designet til at sikre, at lysimpulserne bliver talt korrekt, og at data bliver sendt. I følge kravspecifikationen, skal ECC'en kunne måle frekvenser fra intervallet 10 - 100 Hz. Dette testes ved brug af en Analog Discovery 2, som er forbundet til et fumlebræt med en LED, der kan lyse. Lysdioden placeres således, at den vender ind mod LEDen med to rør for at undgå unødvendigt lys. I programmet WaveForms åbnes Wavegen, hvor det er muligt at justere LEDens blinkhastighed i hertz per sekund. Dette bliver observeret på en serial monitor på Arduino IDE, hvor værdierne bliver udskrevet. Udfra observationerne kan der konkluderes, at Elmålerkredsløbet er i stand til at måle frekvenser i intervallet 10-100 Hz. Koden for at teste dette findes i under bilag.



Figur 51: Opstilling af modultest



Figur 52: Sender firkant signal på 1 Hz per sekund: test

```
50
51
52
53
54
55
56
57
58
59
60
Frekvens (Hz) : 1.00
```

Figur 53: Observeret resultat på serial monitor

9.2 Integrationstest

af Daniel Naddaf

Integrationstesten foregik ved at tilslutte Arduinoen til Raspberry Pi gennem UART og ground. Energy Consumption Calculatoren (ECC'en) kunne ikke integreres i systemet, da Raspberry'en kun kunne have én seriel port, som blev anvendt til Arduinoen. Der blev påbegyndt noget kode til SPI-kommunikation på ECC'en, men tidsmangel gjorde, at dette ikke nåede at blive færdigjort, testet eller integreret. Testen forløb ved at sende forskellige signaler mellem Arduinoen (temperatur regulerings controlleren) og Raspberry'en og kontrollere om informationerne blev verificeret af begge parter, samt om de blev håndteret korrekt på baggrund af deres signaltyper. Informationerne blev modtaget og verificeret korrekt af begge delsystemer.

9.3 Accepttest

af hele gruppen

Use Case 2: Tilpasning af funktioner

Punkt	Præ-kondition	Action	Forventet resultat	Opnået resultat	Kommentar
1	Systemet er operationsdygtigt.	Brugeren trykker på funktionen, som ønskes ændret.	Fanen for den valgte funktion åbnes	Fanen åbner	Der oprettes en funktion for at kunne opnå den.
2	Brugeren kan tilgået funktionen som skal ændres.	Brugeren trykker ”Tilpas funktion”.	Fanen ”tilpas funktion” åbnes for den valgte funktion	Indstillinger til en specifik funktion åbnes	Fanen kaldes ”Setting For Function” i stedet.
3	Parametre, der kan ændres, vises på skærmen.	Brugeren tilpasser parametrene efter eget behov.	Parametrene bliver ændret og vises på skærmen.	Paramentrene ændres og vises	GODKENDT
4	Brugeren kan gemme sine indstillinger	Brugeren trykker gem.	Systemet gemmer indstillinger.	Instillingerne gemmes	GODKENDT
5	De nye indstillinger er gemt.	Systemet sender brugeren til fanen med oversigten over funktionen.	Skærmen viser oversigten over funktionen.	Skærmen viser funktionerne	Lukkes manuelt

Tabel 30: Use Case 2, accepttest

Use Case 3: Temperatur regulering- For lav temperatur					
Punkt	Præ-kondition	Action	Forventet resultat	Opnået resultat	Kommentar
1	Systemet har adgang til elpriser.	Tjekker elpris.	Elpris under prisloft.		
2	Systemets varmeaktuator er operationsdygtigt og forbundet til resten af systemet. Varmelegeme er operationsdygtigt.	Varmeaktuator/ relæ aktiveres	Varmelegeme tændes.	Der måles et højt signal på arduinosens output port	GODKENDT
3	Temperatursensor er operationsdygtigt og forbundet til resten af systemet.	Temperatursensor mäter temperatur.	Temperaturen har den ønskede værdi.	Temperatursensoren mäter temperaturen i rummet og sender denne til Arduinoen.	GODKENDT
4	Systemets varmeaktuator er operationsdygtigt og forbundet til resten af systemet. Varmelegeme er operationsdygtigt.	Varmeaktuator/ relæ slukkes	Varmelegeme slukkes.	Der måles et lavt signal på arduinosens output port	GODKENDT

Tabel 31: Use Case 3, accepttest

Use Case 4: Temperatur regulering- For høj temperatur					
Punkt	Præ-kondition	Action	Forventet resultat	Opnået resultat	Kommentar
1	Systemets varmeaktuator er operationsdygtigt og forbundet til resten af systemet. Varmelegeme er operationsdygtigt.	Varmeaktuator/ relæ slukkes	Varmelegeme slukkes.	Der måles et lavt signal på arduinosens output port	GODKENDT

Tabel 32: Use Case 4, accepttest

Use Case 8: Rum Specification.					
Punkt	Præ-kondition	Action	Forventet resultat	Opnået resultat	Kommentar
1	Hub'en viser fanen "rum".	Brugeren Trykker på specifikke rum	Fanen for det specifikke rum åbnes.	Fanen "Setting For Room" åbnes	Der oprettes en rum for at kunne opnå den.
2	Brugeren kan tilgå indstillinger for det specifikke rum.	Data for prisloft og temperatur indtastes for det specifikke rum.	Prisloft og temperatur ændres	Paramenterne ændres	Der trykkes på gem efter.

Tabel 33: Use Case 8, accepttest

Temperaturregulering				
Punkt	Test	Forventet resultat	Opnået resultat	Kommentar
1	Visuel test	Reguleringssystem er implementeret med en Arduino Mega2560.	Reguleringssystemet er implementeret med en Arduino Mega 2560	GODKENDT
2	Visuel test	Der er anvendt et shield fra PR electronics, med påmonteret LM75.	Der er ikke anvendt et shield til arduinoen	Se note 1 - DELVIST GODKENDT
3	HW test	Arduinoen skriver til- og læser fra Hub'en.	Skriver til og læser fra Hub'en	Godkendt
4	Måling af temperatur. Måling af signal til varmekontakt.	Arduinoen sender et højt signal til kontakten, når temperaturen er under den ønskede værdi.	Der måles et højt signal på arduinoens output port	GODKENDT
5	Måling af dimensioner	Målene overstiger ikke LxBxH: 10x5x4 cm.	Længde 10, Bredde 5,3, Højde 1,5	IKKE GODKENDT

Tabel 34: Accepttest, Temperaturregulering

Temperatursensor				
Punkt	Test	Forventet resultat	Opnået resultat	Kommentar
1	Visuel test	Implementeret med en LM75 temperatursensor.	Der er implementeret en LM75 som temperatursensor	GODKENDT
2	Temperaturmåling.	Måler temperaturer i intervallet 0°C-30°C.	LM75'eren måler i intervaller mellem 0°C-30°C.	GODKENDT
3	Sensor-målinger sammenlignes med eksterne målinger.	Målingerne har en afvigelse på maks $\pm 2^{\circ}\text{C}$	Målingerne har en afvigelse på $\pm 3^{\circ}\text{C}$	IKKE GODKENDT

Tabel 35: Accepttest, Temperatursensor

Varmeaktuator				
Punkt	Test	Forventet resultat	Opnået resultat	Kommentar
1	Visuel test	Implementeret i form af en MOSFET.	Implementeret i form af en MOSFET - demonstreres med en LED	GODKENDT

Tabel 36: Accepttest, Varmeaktuator

Hub				
Punkt	Test	Forventet resultat	Opnået resultat	Kommentar
1	Visuel test	Implementeret med en Raspberry Pi 4.	Implementeret	GODKENDT
2	Signal test	Hub'en kan kobles til internettet.	Koblet til internet	GODKENDT

Tabel 37: Accepttest, Hub

Touchskærm				
Punkt	Test	Forventet resultat	Opnået resultat	Kommentar
1	Visuel test	Touchskærmen skal implementeres via Raspberry Pi 4.	Implementeret med RPi4	GODKENDT
3	Visuel test	Implementeret med en Raspberry Pi 7" touchskærm	Implementeret med Rpi 7" touchskærm	GODKENDT

Tabel 38: Accepttest, Touchskærm

Energimåler				
Punkt	Test	Forventet resultat	Opnået resultat	Kommentar
1	Måling af strømforbrug	Energimåleren mäter strømforbruget.	Se note 2	IKKE GODKENDT
2	Visuel test	Implementeret med en lyspuls-måler der vender ind mod elmåleren.	Implementeret lyspuls-måler og vender ind mod elmåleren	GODKENDT
3	Måling af frekvens	Lyspuls-måleren mäter frekvenser i området 10-100Hz	Lyspuls-måleren kan mæle frekvenser i området 10 - 100Hz	GODKENDT

Tabel 39: Accepttest, Energimåler

9.3.1 Noter til accepttest

Note 1: Et af de ikke funktionelle krav til temperaturregulering var at der skulle anvendes et arduino shield fra PR electronics, hvorpå der er påmonteret en LM75 temperatursensor. Men ud fra flere overvejelser under arbejdet med projektet, blev dette ændret til en ekstern LM75 temperatursensor. Det blev vurderet at det ville give mere mening at bruge en ekstern, da den ville kunne placeres bedre, og da den ikke vil kunne påvirkes af varmen fra microcontrolleren.

Note 2: Det er ikke meningen, at elmåleren skal kunne måle strømforbruget, men at Raspberry Pi skulle regne strømforbruget.

10 Resultater

af hele gruppen

10.1 Temperature Regulation

- Algoritme vælger reguleringstidspunkt på baggrund af temperatur, tempmarkers, forbrug ift gennemsnitligt forbrug, elprisen i et tidsinterval.
- Timing af igangsættelse af reguleringen er ikke korrekt, når programmet skal vente x-antal minutter, indtil den næste time eller to timer frem i tiden.
- Sensor kan have store målfejl.
- Kommunikation med HUB virker som den skal, og programmet kan skelne mellem de forskellige signaler. Det er muligt at følge med i hele processen via terminal på PC.

10.2 HUB

- GUI'en åbner de rigtige sider når der trykkes på knapperne.
- Elementer (rum og funktioner) kan redigeres, tilføjes og slettes gennem GUI, og gemmes derefter i filer.

10.3 Energy Consumption Calculator

- Hardwaren er implementeret korrekt og fungerer som det skal, hvilket muliggøre optælling af lysimpulser
- Softwaren på arduinoen fungerer via USB-port fra computeren
- Kommunikation med HUB'en blev ikke testet, grundet af tidsmangel

10.4 Hele systemet

- Systemet kan ikke testes i sin helhed, da HUB'en kun kan forbindes til ét andet delsystem.
- HUB'en kommunikation med Temperature Regulation virker som det skal.

11 Diskussion af resultater

11.1 Temperature Regulation

af Daniel Naddaf & Jonas Kirkegaard Skyum

Projektets algoritme virker som den skal ved at vælge reguleringstidspunkter baseret på flere parametre såsom temperatur, temperaturmarkører, forbrug i forhold til gennemsnitligt forbrug og elprisen i et givet tidsinterval.

Dog er der identificeret et problem med timing af reguleringen. Når programmet skal vente et bestemt antal minutter, indtil den næste time eller to timer frem i tiden, igangsættes reguleringen ikke på det korrekte tidspunkt. Dette indikerer en nødvendighed for at forbedre algoritmen, for at opnå en mere præcis regulering.

En anden udfordring er sensorens målefejl, som kan have betydelige konsekvenser for systemets præcision og pålidelighed. Store målefejl i sensoren kan føre til ineffektiv energistyring og utilstrækkelig regulering for brugeren. Derfor er det essentielt at undersøge og muligvis udskifte eller kalibrere sensoren for at minimere disse fejl. På den positive side fungerer kommunikationen med HUB'en som forventet, og programmet er i stand til at skelne mellem forskellige signaler, indpakke signalerne som sendes, samt afkode signalerne som modtages. Dette er en essentiel del af systemet, da det muliggør en problemfri udveksling af data mellem de forskellige enheder, hvilket er afgørende for systemets overordnede funktionalitet.

Samlet set virker projektet med visse nøglekomponenter, såsom kommunikationsprotokollen og overvågningsmulighederne. Dog er der behov for yderligere arbejde med hensyn til præcisering af tidsstyring og sensorernes nøjagtighed for at forbedre systemets effektivitet og pålidelighed yderligere.

11.2 Hub

De opnåede resultater er generelt tilfredsstillende og opfylder projektets målsætninger. Tabellen for accepttestene for ”Use Case 2: Tilpasning af funktioner” viser, at systemet fungerer som forventet i alle testpunkter, hvor brugerne kan tilpasse funktioner, ændre parametre og gemme indstillinger effektivt. Dette matcher de forventede resultater og sikrer, at systemet er brugervenligt og responsivt, hvilket er essentielt for at give brugeren mulighed for at optimere energiforbruget. Ligeledes viser testresultaterne for ”Use Case 8: Rum Specifikation,” at brugeren kan tilpasse indstillinger for specifikke rum uden problemer, hvilket understøtter vores mål om præcis styring og overvågning af energiforbruget i forskellige rum, afgørende for energieffektivitet og brugerkomfort.

Hub'en, implementeret med en Raspberry Pi 4, er blevet testet både visuelt og med signaltest, og resultaterne viser, at den fungerer korrekt og kan tilsluttes internettet uden problemer. Tilsvarende er touchskærmen testet og godkendt til både hardware- og softwareintegration med Raspberry Pi 4, hvilket understøtter målet om at skabe en intuitiv og funktionel brugerflade.

Samlet set er de ønskede resultater opnået, og som matcher projektets problemformulering. Systemet kan optimere energiforbruget ved at kontrollere varmelegemer baseret på brugerens præferencer og aktuelle energipriser. De omfattende testresultater viser, at både hardware- og softwarekomponenter fungerer som forventet. De omfattende testresultater viser, at både hardware- og softwarekomponenter fungerer som forventet, og ved at opnå disse resultater demonstreres tekniske færdigheder samt evnen til at skabe praktiske og brugervenlige løsninger.

11.3 Energy Consumption Calculator

af Jakob & Nichlas

Under opbygningen af hardwaren opstod der et problem med signalstyrken fra operationsforstærkeren. Operationsforstærkeren der blev brugt var ikke i stand til, at levere nok udgangsspænding for at Arduinoen kunne registrere et HIGH signal. Det gjorde at Arduinoen ikke modtog de forventede signaler. For at løse problemet blev der tilføjet en ekstra operationsforstærker, som øgede udgangsspændingen, så Arduinoen kunne modtage de nødvendige signaler.

Softwaren stødte dog på en del problemer under udviklingen da arbejde med måling af lysimpulser har vist sig udfordrende at holde konstant. Dette blev overkommet ved brug af interrupts, der tillod programmet kun at registrere Rising signaler, samt at teste softwaren i et kontrolleret mørkt miljø, der fjernede den usikkerhed som ellers ville kunne forstyrre programmet. Givet de korrekte parametre og miljø vil programmet kører som forventet, beregne den korrekte frekvens for lysimpulser og til sidst pakke informationen i en protokol som HUBBEN kan afkode.

Selvom resultaterne for elmåleren er korrekte, og 2 ud af 3 fastsatte krav er opfyldt, blev kommunikationen mellem Arduino og hubben ikke testet på grund af tidsmangel. Dette er kritisk for produktets funktionalitet. Det er dog positivt, at softwaren på Arduino fungerer via USB-porten fra computeren. Derfor, selvom testen ikke blev udført, fungerer de grundlæggende funktioner i softwaren stadig.

11.4 Hele systemet

Kommunikationen på tværs af delsystemerne kan fungere, idet der er udarbejdet drivere til både Arduinoer og Raspberry Pi, som følger samme protokol. Driverne fungerer fint sammen, og blev testet mellem HUB'en og temperaturreguleringen. Dog blev det af ukendte årsager ikke undersøgt hvor mange serielle porte der er på Raspberry Pi 4, og det kom derfor meget sent frem, at der kun er én seriel port, og at det derfor umiddelbart ikke var muligt at forbinde alle delsystemer på én gang, uden en ændring i systemets arkitektur og design, hvilket der ikke var tid til. Udarbejdelsen af drivere og protokoller blev udskudt til meget sent i projektet, men på trods af dette, lykkedes det at pakke, sende, modtage og verificere signaler mellem Arduino og Raspberry Pi, efter den planlagte protokol.

12 Konklusion

af hele gruppen

Det er lykkedes at udarbejde et samlet system, bestående af tre delsystemer, hvoraf 2 af dem er implementeret med Arduino Mega2560 microcontrollerer og det sidste med en Raspberry Pi 4. Kommunikationen mellem HUB'en og den ene Arduino (Arduino1) blev ikke etableret på grund af tidsmangel og manglende serielle porte på Raspberry'en. Denne skulle ellers måle lyspuls-frekvenser fra en elmåler og sende disse til HUB'en, som skulle omregne frekvensen til et forbrug. Den anden Arduino (Arduino2) står for temperatur-målinger og reguleringen heraf, ved at tænde og slukke for et varmelegeme. Valget om at regulere baseres på data som sendes til denne Arduino fra HUB'en, gennem en 2-vejs seriel kommunikations kanal. HUB'en kan indhente aktuelle energipriser gennem en API, og disse sendes videre til Arduino2. Tanken var at disse elpriser skulle kunne skaleres ift brugerens el-abonnement, men dette blev nedprioriteret på grund at tidsmangel. Oprindeligt skulle brugeren tage stilling til, hvordan systemet agerer ved samtidige registreringer af høj pris og lav

temperatur. I stedet er der tilføjet 2 parametre, tempMarker1 og -2, som er med til at regulere temperaturen på de mest fordelagtige tidspunkter, mht elprisen. Brugeren kan indtaste deres ønskede temperatur for lokalet, et prisloft samt tempMarkers. Planen var også at implementere en smartkontakt, men mangel på tid gjorde, at denne del måtte tilskidesættes.

13 Fremtidigt arbejde

13.1 UART Kommunikation

I integrationen af vores system stødte vi på det primære problem: manglen af UART-porte på vores Raspberry Pi. Dette er en udfordring, da både signalet fra ECC'en og temperaturreguleringen anvender UART, men Raspberry Pi'en har kun én UART-port. Derfor er det nødvendigt at kunne bro mellem de to signaler, så de kan kommunikere med Raspberry Pi'en. En løsning hertil kunne være enten en serial bridge-kommunikationsmetode eller en daisy chain-kommunikationsmetode.

"Daisy chaining" refererer til metoden, hvor flere forskellige moduler er forbundet i en kæde af serielle forbindelser, der tillader kommunikation gennem kæden og dermed skaber forbindelse mellem alle elementer. På den anden side fungerer en serial bridge ved at have en "bro" mellem vores Raspberry Pi og andre moduler. I dette tilfælde ville en af de to Arduino-enheder agere som bro, med den anden Arduino forbundet til den. Da vi kun arbejder med to Arduino-enheder, vil disse kommunikationsmetoder sandsynligvis fungere på samme måde, da en kæde af to enheder og en bro af to enheder opstilles på samme måde.

14 Kildeliste

Litteratur

- [1] Ukendt forfatter, *SW2ISE-01 Indledende System Engineering*, kursuskatalog.au.dk, sidst besøgt d. 23-05-2024, url: <https://kursuskatalog.au.dk/da/course/123934/SW2ISE-01-Indledende-System-Engineering>.
- [2] Ukendt forfatter, *SysML FAQ: What is the relation between SysML and UML?*, sysml.org, sidst besøgt d. 23-05-2024, url: <https://sysml.org/sysml-faq/what-is-relation-between-sysml-and-uml.html>.
- [3] Kim Bjerge (KBE) (2015) *Vejledning til udviklingsprocessen for projekt 2*, Aarhus Universitet.
- [4] Ukendt forfatter, *What is PlatformIO?*, platformio.org, sidst besøgt 23-05-2024, url: <https://docs.platformio.org/en/latest/what-is-platformio.html>.
- [5] Ukendt forfatter, *Stream*, arduino.cc, sidst besøgt 25-05-2024, url: <https://www.arduino.cc/reference/en/language/functions/communication/stream/>.
- [6] Aarhus Universitet, *HÅNDBOG FOR CDIO VED ASE*, ase.medarbejdere.au.dk ,sidst besøgt 28-05-2024, url: https://ase.medarbejdere.au.dk/fileadmin/www.ase.au.dk/Filer/Medarbejdere/CDIO-haandbog-2012_01.pdf.
- [7] Ukendt forfatter, *32-bit PSoC™ 5 LP Arm® Cortex®-M3*, infineon.com, sidst besøgt 27-05-2024, url: <https://www.infineon.com/cms/en/product/microcontroller/32-bit-psoc-arm-cortex-microcontroller/32-bit-psoc-5-lp-arm-cortex-m3/>

- [8] Ukendt forfatter, *Projekt 2*, monday.com, sidst besøgt 28-05-2024, url: <https://daniel-naddaf.monday.com/boards/1397314738>
- [9] Ukendt forfatter, *SW1MSYS-01 Microcontroller systemer*, Brightspace.au.dk, sidst besøgt 19-05-2024, url: <https://brightspace.au.dk/d2l/le/lessons/107054/units/1438287>
- [10] Ukendt forfatter, *Official Raspberry Pi 7 Touchscreen Display*, raspberrypi.dk, sidst besøgt 28-05-2024, url: <https://raspberrypi.dk/en/product/official-raspberry-pi-7-touchscreen-display/>
- [11] Ukendt forfatter, *Raspberry Pi hardware, GPIO and the 40-pin header*, raspberrypi.com, sidst besøgt 28-05-2024, url: <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html>
- [12] Ukendt forfatter, *I2C FAQ*, esacademy.com, sidst besøgt 29-05-2024, url: <https://www.esacademy.com/en/library/technical-articles-and-documents/miscellaneous/i2c-bus/frequently-asked-questions/i2c-faq.html>
- [13] Python Software Foundation, *Built-In Functions*, docs.python.org, sidst besøgt 29-05-2024, url: <https://docs.python.org/3/library/functions.html>
- [14] Ukendt forfatter, *Forum, Serial Bridge?*, forum.arduino.cc, sidst besøgt 30 - 05 - 2024 url: <https://forum.arduino.cc/t/serial-bridge/14617/3>
- [15] Ukendt forfatter, *Arduino pin levels HIGH and LOW*, arduino.cc, sidst besøgt 30 - 05 - 2024 url: <https://www.arduino.cc/reference/en/language/variables/constants/highlow/>
- [16] Ukendt forfatter, *Daisy-chain, What is a Daisy-chain?*, analog.com, sidst besøgt 30 - 05 - 2024 url: https://www.analog.com/en/resources/glossary/daisy_chain.html

15 Bilagsoversigt

A BDD

Dette bilag indeholder et block definition diagram for hele systemet.

B Sekvensdiagram: Seriel kommunikation

Dette bilag indeholder sekvensdiagrammet til beskrivelse af systemets serielle kommunikation iblandt dets interne delsystemer.

C Klassediagram: lavTemperatur

Dette bilag indeholder klassediagrammet for temperaturreguleringen. Dette er en del af applikationsmodellen for delsystemet *Temperature regulation*.

D Statemachine-diagram: lavTemperatur

Dette bilag indeholder statemachine-diagrammet for temperaturreguleringen. Dette er en del af applikationsmodellen for delsystemet *Temperature regulation*.

E Sekvensdiagram: lavTemperatur

Dette bilag indeholder det store sekvensdiagram for temperaturreguleringen i softwaredesign afsnittet (7.2). Dette er en del af applikationsmodellen for delsystemet *Temperature regulation*.

F Statemachine-diagram: hub

Dette bilag indeholder statemachine-diagrammet, til beskrivelse af de forskellige stadier, for hub'en.

G Design-Hub: General Setting

Dette bilag indeholder designet af siden ”General Setting” for interface. Dette er en del af softwaredesign afsnit for Hub (7.3.1)

H Design-Hub: Data

Dette bilag indeholder designet af siden ”Data” for interface. Dette er en del af softwaredesign afsnit for Hub (7.3.1)

I Design-Hub: Units

Dette bilag indeholder designet af siden ”Units” for interface. Dette er en del af softwaredesign afsnit for Hub (7.3.1)

J Design-Hub: Setting For Unit

Dette bilag indeholder designet af siden ”Setting For Unit”for interface. Dette er en del af software-design afsnit for Hub (7.3.1)

K Implementering-Hub: General Setting

Dette bilag indeholder implementering af siden ”General Setting”for interface. Dette er en del af frontend implementering afsnit for Hub (8.2.1)

L Implementering-Hub: Data

Dette bilag indeholder implementering af siden ”Data”for interface. Dette er en del af frontend implementering afsnit for Hub (8.2.1)

M Implementering-Hub: Units

Dette bilag indeholder implementering af siden ”Units”for interface. Dette er en del af frontend implementering afsnit for Hub (8.2.1)

N Implementering-Hub: Setting For Unit

Dette bilag indeholder implementering af siden ”Setting For Unit”for interface. Dette er en del af frontend implementering afsnit for Hub (8.2.1)

O Implementering-ECC: Top-Down design

Dette bilag indeholder et udvidet top down design for implementering for elmåleren. Dette er en del af implementering afsnit for ECC. (8.3.2)

P Sekvensdiagram: backend

Dette bilag indeholder sekvensdiagrammet, der beskriver kommunikationen mellem blokkene i backend.

Q Interface Kode: Frontend

Denne bilag indeholder hele frontend koden for Hub. Dette er en del af frontend implementerings afsnit (8.2.1)

R ECC: Datasheets

Dette bilag indeholder datasheets af hardwaren af ECC. Dette er en del fra hardware design afsnit for ECC (7.1.1)

S ECC: kode

Dette bilag indeholder alt koden for Elmåleren. Dette er bruges til modultest af kodden som ses i afsnit (9.1.3)

T Kredsløbsdiagram

Dette bilag indeholder begge kredsløbsdiagrammer for ECC hardwaren. (Til test og produkt) (7.1.1)

U Source code: backend

Dette bilag indeholder alt source code til backenddelen til HUB delsystemet, der er udviklet under implementering i afsnit 8.2.3

V Source code: Temperature Regulation

Dette bilag indeholder alle kodelinjer til delsystemet Temperature Regulation.

W Proces

Dette bilag indeholder Proces-delen til projektet, heri findes også procesbeskrivelsen som omtales i afsnit 5.1.