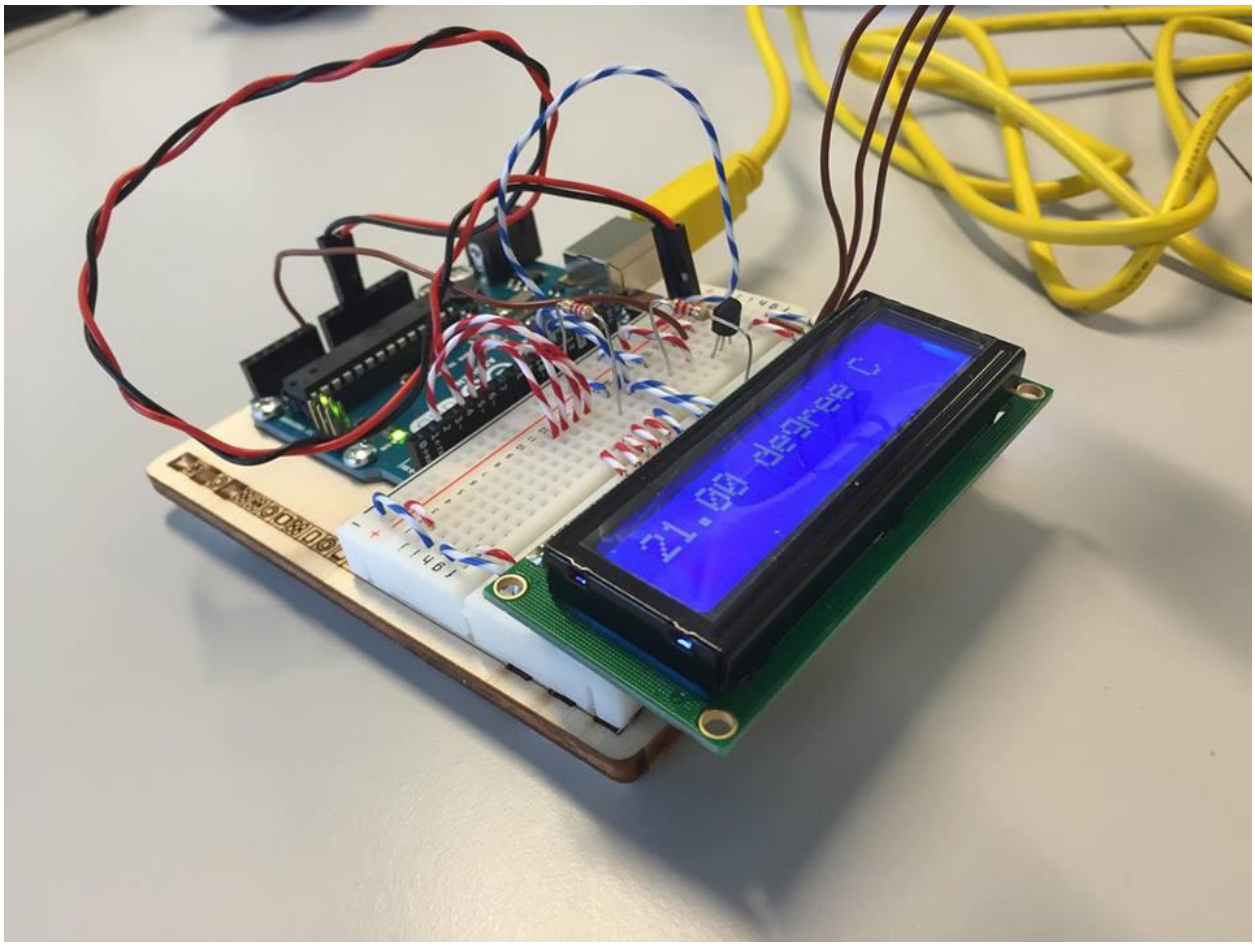
Høgskolen i Sørøst-Norge
Fakultet for teknologiske fag
Bachelor i ingeniørfag

RAPPORT FRA 2. SEMESTERS PROSJEKT VÅREN 2016

IA2112 Objektorientert programmering - videregående

Gruppe 3

Alarmsystem med PC og Arduino UNO



Høgskolen i Sørøst-Norge
Fakultet for teknologiske fag
Bachelor i ingeniørfag

RAPPORT FRA 2. SEMESTERS PROSJEKT VÅREN 2016

Emne: *IA2112 Objektorientert programmering - videregående*

Tittel: *Alarmsystem med PC og Arduino Uno*

Rapporten utgjør en del av vurderingsgrunnlaget i emnet.

Prosjektgruppe: *Gruppe 3* **Tilgjengelighet:** *Åpen*

Gruppedeltakere:

Øyvind Skaten
Sander Heldal
Shing Wai Chan
Simen Theie Havenstrøm

Veileder: *Morten Borg*
Sensor: *Morten Borg og Nils Olav Skeie*

Godkjent for arkivering:_____

Sammendrag:

Et alarmsystem for hus- eller hytteinstallasjon skal utvikles. Dette systemet skal håndtere alarmer for bevegelse og temperatur, samt feil på PC som manglende lading, lav batteriprosent, og feil på kommunikasjon eller sensorer. Alarmer skal varsles til brukere av systemet ved E-post. Det er på bakgrunn av systemspesifikasjon fra kunde (oppgavetekst) at gruppens egne spesifikasjoner og løsninger er basert.

Alarmsystemet bruker Arduino Uno med sensorer for bevegelse og temperatur for å måle de fysiske størrelsene. All annen logikk og funksjonaliteter i programmet løses i PC-applikasjonen. PC-applikasjon programmeres i C# i utviklingsmiljøet Visual Studio. Brukergrensnittet blir laget som en Windows form app.

Programkoden som leveres ved dette prosjektet oppfyller alle de funksjonelle kravene, så vel som kravene til integrering av ulike programmeringsteknikker; slik som events, interface-implementering og klassestruktur med arv. Brukergrensesnittet er enkelt og lett oversiktlig, og møter kravene satt i spesifikasjonene.

FORORD

Denne rapporten er utarbeidet av fire studenter på Høgskolen i Sørøst-Norge, avd. Porsgrunn. Dette i sammenheng med prosjektoppgave i emne «Objektorientert programmering – videregående» i 2. semesteret i studiet «Informatikk og automatisering».

Rapporten omfatter analyse av programdesign, beskrivelse av databasestruktur, og andre tekniske beskrivelser av utstyret og koding som er benyttet i prosjektbesvarelsen. For å få optimalt utbytte av denne rapporten er det derfor nødvendig å ha kjennskap til programmeringsspråket C#, grunnleggende kunnskap om databaser, Arduino, samt prosjektering og utvikling av dataprogrammer.

Dataprogrammer tatt i bruk:

- SQL Sever 2014 Management Studio
- CA Erwin Data Modeler r9.6
- Visual Studio 2015
- Arduino
- Fritzing

Sted, dato:

Øyvind Skaten	Sander Heldal
Shing Wai Chan	Simen Theie Havenstrøm

INNHOLDSFORTEGNELSE

1 INNLEDNING

Utgangspunktet for prosjektet er å utvikle et alarmsystem, som kan benyttes eksempelvis på en hytte eller fritidsbolig, for å håndtere en rekke ulike alarmer. Når alarm inntreffer skal abonnenter bli varslet på e-post eller SMS. Alarmer for temperatur (høy/lav), bevegelse, manglende lading på PC, lav batterispenning på PC, og feil på kommunikasjon med Arduino Uno (Arduino) skal håndteres av alarmsystemet. Programmet skal også vise live-temperatur på PC og LCD-display på Arduino.

Alarmsystemets brukergrensesnitt lages i Visual Studio i C# som Windows Forms Application. For å måle fysiske størrelser som temperatur og bevegelse brukes Arduino Uno med tilhørende sensorer. Noe logikk kan programmeres i Arduino, men gruppen har valgt å programmere all logikk i hovedprogrammet, for ryddighetens skyld.

Database blir modellert og generert script for vha. Erwin. Videre brukes SQL Management studio til å administrere databasen og lage spørringer og prosedyrer. Kobling mellom hovedprogrammet og database programmeres i Visual Studio.

Spesifikasjoner gruppen har definert for programmet er som følger:

- Varsling av alarmer sendes på E-post. Epost skal inneholde hva slags alarm om har inntruffet og en liten beskrivelse.
- Alarmhistorikk lagres i database. Denne skal kunne lagres i listeformat i et PDF-dokument, og sendes ved e-post til brukere.
- Administrasjon av abonnenter får sin egen form i programmet. Her skal informasjon om abonnentene og hvilke alarmer en spesifikk abonnent får varling om kunne endres.
- Opprettelse av ny abonnent får sin egen form i programmet, hvor man legger inn personalia og type alarmer det ønskes å bli varslet om.
- Grafisk temperaturvisning, utvidet alarmhistorikk og vindu for innstillinger får hver sin form.
- Programmets hovedform viser oppsummering av programmets viktigste funksjoner, dvs. live temperatur, de siste alarmene som er varslet, status på bevegelsesdetektering, samt sende alarmliste på e-post til abonnentene.
- Arv og interface skal implementeres i kodingen. Det skal og benyttes events og threading.

Hovedmålet for prosjektet er å lage et program som oppfyller spesifikasjonene, samt har en fornuftig og brukervennlig løsning for brukergrensesnittet.

Leserveiledning:

- Kapittel 2 omhandler analyse og design
- Kapittel 3 omhandler testing
- Kapittel 4 er brukerveiledning til programmet
- Kapittel 5 er oppsummeringen av prosjektet

2 ANALYSE OG DESIGN

Kapittelet tar for seg prosjektering og oppbyggingen av løsningen som er utviklet. Underkapitlene omhandler use case diagram, klassediagram, database, og Arduino.

2.1 Use case

Her henvises det til Vedlegg G for use case diagram. Videre beskrives hva som er planlagt i forhold til programmets funksjoner. Det er opprettet to use case diagrammer for henholdsvis PC og Arduino.

Konfigurere Parametere:
Parameterne leseintervall og loggeintervall settes av bruker, og lagres til fil. En klasse, som kalles ProgramParameters, opprettes for å håndtere kommunikasjon mellom programmet og parameterne lagret i fil. Det er mulig å justere parameterne når programmet kjører.

Registrere og redigere abonnent:
Bruker av programmet skal kunne legge til og redigere informasjon om abonnenter i databasen. En sentral funksjonalitet i sammenheng med dette er muligheten for at forskjellige abonnenter skal kunne abonnere på ulike alarmer. Abonnenter skilles med primærnøkkelen E-post, det vil si at én e-post kun kan legges inn én gang i databasen.

Temperaturmåling:
Temperaturmåling gjøres med sensor koblet til Arduino. Videre sendes rådataene for behandling i PC. Målingene blir brukt på flere måter:

- For å lagre en enkelt måling i databasen
- Hentes fra database for å lage en temperaturgraf som vises på skjerm
- Kontrolleres i forhold til alarmgrensene om status er ok, eller om alarm skal utløses.

Alarm og alarmhistorikk:
Full historikk for alle typer alarmer skal kunne vises, i tillegg til dette vil de siste alarmene som er inntruffet være synlig i programmets hovedform. Når alarm oppstår vil den automatisk bli lagret i databasen. Logikken for alarmene blir programmert i egne klasser. Her skal og Arduino involveres, fordi lydsignal skal gis når alarm inntreffer. Ellers er det e-postvarsling til abonnentene som blir utført ved alarm.

PDF:
Alarmhistorikken skal kunne konverteres til en PDF-fil hvor alarmene vises i listeform. En egen klasse håndterer dette. Den skal igjen kunne sendes som vedlegg til e-post om bruker skulle ønske det.

Lese data:
Kommunikasjon mellom Arduino og PC behandles her. Dataene som sendes er temperatur- og bevegelsessensorenes verdier. Disse må videre tolkes og sorteres i PC-programmet. Det blir og programmert et målefilter på temperaturmåling for å unngå støy og unøyaktigheter.

Videre beskrives use case for Arduino.

Måle sensorer:
Input-aktørene her er temperatur og bevegelses verdier. Disse verdiene blir sendt til PC og LCD-display.

Alarm:
Når alarm inntreffer sendes det signal til Arduino om å aktivere lydsignal, eller «buzzer», i 3 sekunder. Det skal og vises en melding på LCD-display når alarm er inntruffet.

2.2 Klassediagram

Klassediagrammet viser hvordan oppgavene i programmet er fordelt og løses på. Klassene har hvert sitt ansvarsområde og videre beskrives dette enkelt. For mer detaljert forklaring på hvordan koden fungerer se Vedlegg C med kommentarer.

ArduinoCom og Arduino: ArduinoCom oppretter en kommunikasjonsport til mikroprosessoren. Når porten opprettet tar klassen Arduino seg av behandling av dataene som leveres via porten.

SQLCom og SQL: Samme prinsippet som over. SQLCom oppretter kommunikasjon til databasen, og SQL er klassen som leverer alle kommandoene/spørringene til databasen.

Alarmklassene: Arver felles kode fra en abstrakt baseklasse kalt «Alarm». De arvede klassene tar seg av logikken for hver sin alarmtype, måtte det være temperaturalarm, bevegelsesalarm osv.

PdfForm: Oppretter et PDF-dokument, formaterer og legger inn tekst av alarmhistorikken.

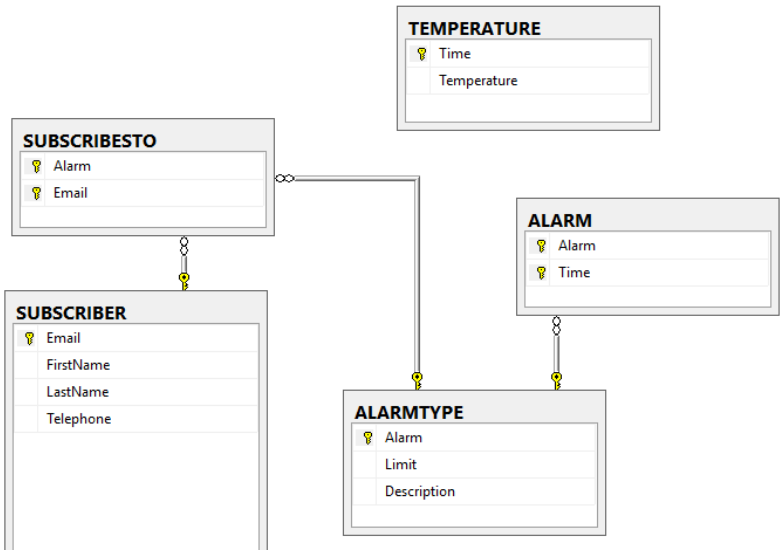
ProgramParameters: Lagrer og leser innstillingene for lese- og lagringsintervall til configfil.

Mail: Tar seg av alle oppgaver i forbindelse med oppretting og sending av mail til brukere.

Subscribers: klassen som lager dataobjekter av abonnenter i programmet, og brukes i samarbeid med klassen SQL for å opprette og oppdatere abonnenter i databasen.

2.3 Database

Databasestrukturen og spørringene som brukes i denne løsningen er laget i utviklingsmiljøet SQL Server Management Studio. Figur 2-1 er et diagram over tabellene databasen inneholder.



Figur 2-1 Databasediagram

I tabellen «Subscriber» blir abonnentene lagret. Her skilles de ulike abonnentene ved en unik e-post adresse. Videre legges det inn informasjon om fornavn og etternavn, samt telefonnummer i databasen. Telefonnummer er lagt inn som kolonne i tabellen med tanke på videreutvikling av programmet, da ved SMS-varsling.

Forskjellige alarmtyper legges inn i tabellen «Alarmtype», da med navn på alarm, grenseverdien for den respektive alarmen, og en kort beskrivelse. Det er navnet på alarmen som er primærnøkkelen i denne tabellen. Da det er et krav i spesifikasjonene at en abonnent skal kunne abonnere på flere typer alarmer er det laget en tabell som heter «SubscribesTo». Her er det et mange-til-mange forhold mellom tabellene «Alarmtype» og «Subscriber». Det vil si at en

abonnent skal kunne kobles mot (les: abonnere på) flere typer alarmer, og en spesifikk alarmtype skal kunne kobles mot flere abonnenter.

Tabellen «Alarm» inneholder alarmhistorikken som skal brukes i programmet. Her er det to primærnøkler som identifiserer en spesifikk alarm; kombinasjonen av hva slags alarm som har gått av (alarmtypen) og tidspunktet alarmen er utløst på.

Til slutt er det en enkeltstående tabell for temperaturloggen. Her skilles en unik temperaturmåling av måleverdien for temperatur, og tidspunktet hvor målingen er tatt.

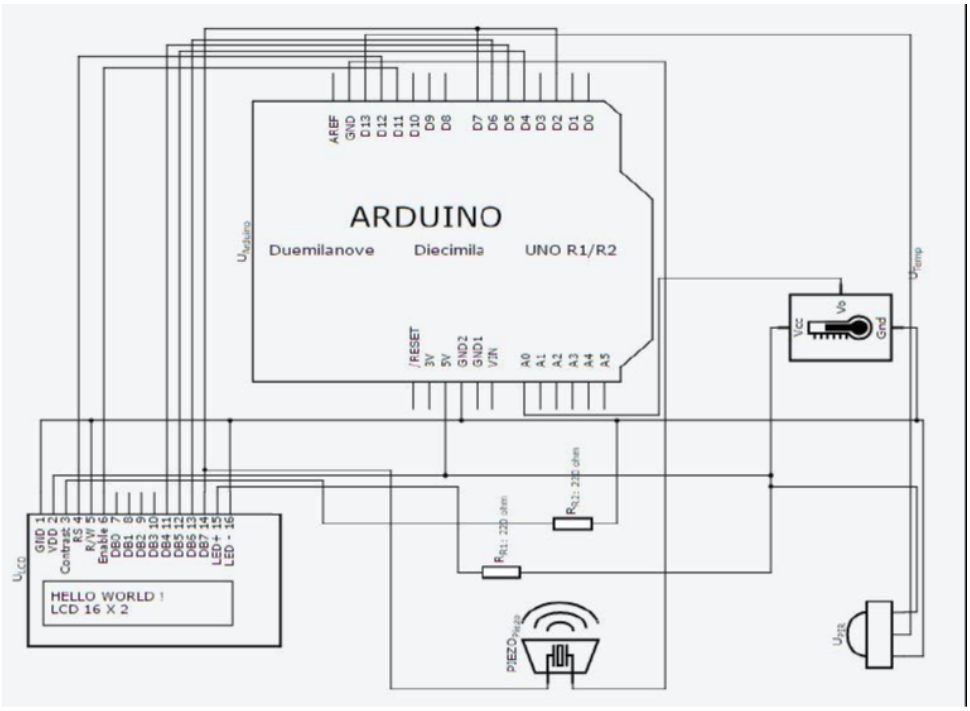
Når det gjelder interaksjoner mellom hovedprogrammet og databasen er det opprettet «Stored Procedures» for å forenkle og gjøre programkoden ryddig.

2.4 Arduino

Arduino Uno er en programmerbar mikroprosessor som er benyttet i alarmsystemet. Den programmeres i medfølgende utviklingsmiljø med samme navn. I henhold til use case for Arduino skal sensormålinger behandles, samt alarm skal gi lyd på buzzer og melding i LCD-display. Til dette kreves det følgende komponenter:

- Arduino Uno mikrokontrollerkort basert på AVR Atmega 328.
- Temperatur sensor (TMP36) er koblet i analog inngang pin A0. Måler analog verdi i volt, skaleres så til Celsius og sendes med i datastringen til PC.
- Pir sensor (HC-SR501) er koblet i digital inngang med pin 13. Måler enten tilstand «HIGH» eller «LOW» og sendes som «ON» eller «OFF» i datastringen til PC.
- LCD-display, printer ut temperaturverdi eller alarm. Kobles iht. skjema.
- Buzzer er koblet til digital utgang pin 7. Gir lydsignal når utgangen er høy.
- 2x220Ω motstand, for henholdsvis kontrast og bakgrunnsbelysning på LCD-display.

Figur 2-2 viser koblingsskjemaet for Arduino-komponentene.



Figur 2-2 Elektrisk oppkobling av Arduino

3 TESTING

Testing av programmet er utført på to forskjellige måter; predefinerte tester for ferdig program og testing/debugging underveis under programmeringen. Dette er en gunstig måte å drive softwaretesting på da man får plukket opp småfeil og bugs «as-you-og», samtidig som man får et helhetlig inntrykk av programmet når man skal teste større prosedyrer til slutt.

I dette kapittelet vil de predefinerte testene programmet må bestå før levering bli beskrevet.

Test 1 – Automatisk tilkobling/opprettelse av database ved oppstart:
Ved oppstart skal programmet automatisk prøve å tilkoble databasen. Hvis ikke databasen eksisterer på computeren skal den opprettes ved å kjøre et script.

Test 2 – Legge til og redigere abonnenter:
Test utføres ved fremgangsmåte beskrevet i kapittel 4.2. Tabellene i databasen ble oppdatert riktig og formen viser korrekte verdier.

Test 3 – Alarmtest:
Test går ut på å simulere at en alarm er inntruffet. I henhold til use case skal e-postabonnenter bli varslet, buzzer skal gi lydsignal, alarmen skal lagres i databasen, visning i LCD-display og i alarmlisten.

Test 4 – Eksportere til PDF:
Alarmhistorikkens eksport til PDF-funksjon og legg ved e-post testes.

Test 5 – Send e-post:
E-post skal sendes til abonnenter i databasen, eller andre e-postadresser. Testmail sendes og kontrolleres.

Test 6 – Endre innstillinger:
Mulighet for å endre programmets parametere gjøres fra innstillingsvindu. Korrekte verdier for lese- og lagringsintervall lagres og leses til/fra configfil. Alarmgrensene skal oppdateres i riktig rad i databasetabellen.

Test 7 – Vis temperaturgraf:
Temperaturgraf skal vise korrekte avleste verdier fra Arduino. Datovisning for graf testes.

Test 8 – Lese og vise data fra Arduino:
Verdiene avlest av Arduino for temperatur og PIR behandles riktig i programmet, og presenteres korrekt på skjerm og LCD-display.

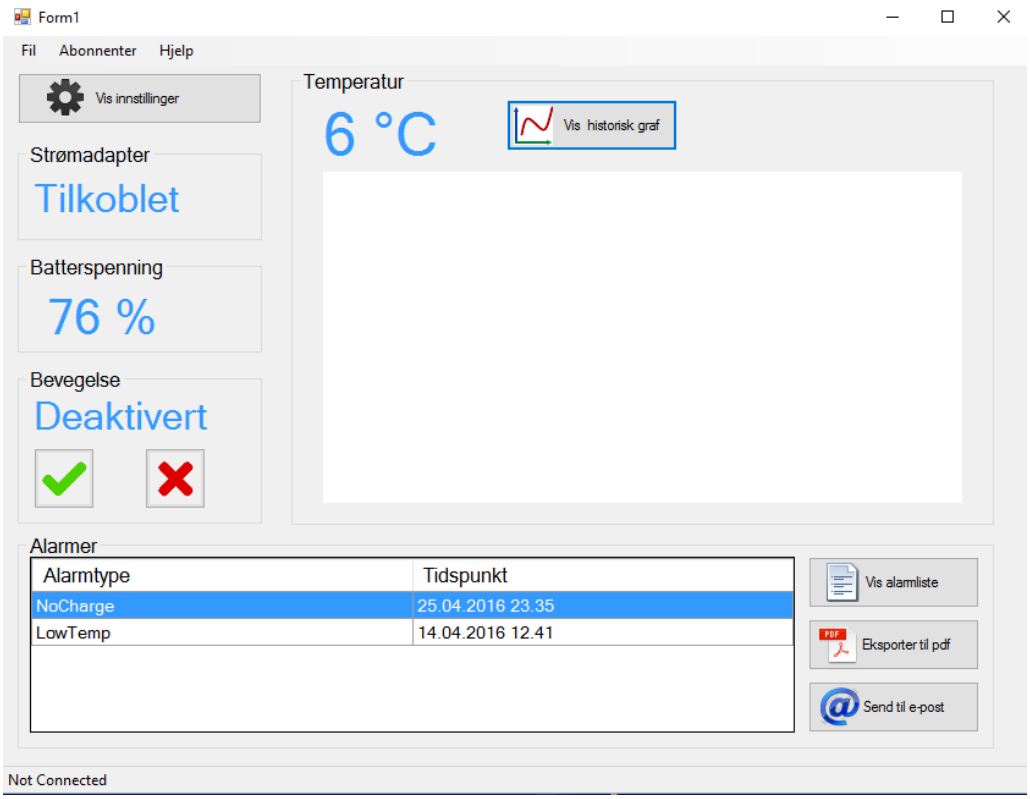
4 BRUKERVEILEDNING

I dette kapittelet er det laget en kort brukerveiledning for applikasjonen gruppen har utviklet. Det innebærer beskrivelse av brukergrensesnitt og hvordan man skal gå frem for å bruke funksjonene. Denne veiledningen er rettet mot allmenne brukere uten spesielle forkunnskaper om data.

4.1 Oppstart og forside

Når man starter programmet vil det automatisk prøve å opprette forbindelse til databasen via databasehåndteringssystemet, som i dette tilfellet er SQL Server. Hvis den ikke finner servernavnet ber den bruker om å skrive det inn (navnet på computeren programmet kjøres fra). Deretter sjekkes det om databasen «Alarmsystem» eksisterer i SQL Server. Hvis ikke databasen eksisterer blir den opprettet automatisk, da uten data som abonnenter, alarmhistorikk eller loggede verdier.

Etter at forbindelsen til databasen er opprettet møter man programmets forside. Figur 4-1 viser hvordan programmets forside er designet.



Figur 4-1 Forside

Forsiden er en slags oppsummering av programmets mest sentrale funksjoner. Livevisning av temperatur, i tillegg til graf er lokalisert øverst til høyre i vinduet. Tilgang til programmets innstillinger når man ved å trykke på «Vis innstillinger» til venstre for temperaturseksjonen.

Ellers er det oversikt over strømstatus, batterispenning, og aktivering/deaktivering av bevegelsessensoren på Arduino. Til slutt er det en liten liste over de fem siste alarmene som har inntruffet, så vel som mulighet til å vise komplett alarmhistorie i egen visning. Den komplette alarmlisten kan eksporteres som PDF til og sendes til e-postadresse om ønskelig. For å sende alarmlisten på e-post trykk på knappen «Send til e-post». Her får man mulighet til å velge en e-post som ligger i databasen, eller skrive inn e-postadressen manuelt.

4.2 Legge til og administrere abonnenter

I menylinjen er det en tab med navn «Abonnenter». Trykker man på denne får man to muligheter, «legg til ny» og «Administrer abonnenter». Klikk på legg til ny for å åpne en ny form hvor man skriver inn personalia om den nye abonnenten og hvilke alarmer det ønskes varsling om. Trykk deretter på knappen «Legg til» for å bekrefte at denne abonnenten skal legges til i databasen. Husk at e-postadresser må være unike i databasen, så feilmelding vil oppstå om det forsøkes å legge til en allerede eksisterende e-postadresse.

For å endre personalia eller oppdatere hvilke alarmer det ønskes varsling om, klikk på «Administrer abonnenter». Det vil åpnes en form hvor det vises en liste over abonnentene i databasen. Trykk på abonnenten det ønskes å oppdatere og tekstboksene med personalia og «checkboxene» for alarmer fylles automatisk ut. En kan nå bruke de samme tekstboksene til å gjøre endringer. For å endre hvilke alarmer den respektive abonnenten er koblet mot aktiveres eller deaktiveres «checkboxene». Når ønskede endringer er utført brukes knappen «Oppdater» for å gjøre endringene permanente i databasen.

4.3 Temperatur- og alarmhistorikk

Trykk på knappen «Vis historisk graf» på programmets forside for å åpne en ny form med temperaturhistorikken. På denne formen er det mulighet for å følge temperaturgrafen under nåtid eller velge og vise graf over temperaturmålingene innenfor et spesifikt datointervall, angitt av brukeren. For å velge visning huker man av på den respektive radioknappen for den ønskelige visningen.

Alarmhistorikken vises om man trykker på «Vis alarmliste» på applikasjonens forside. På samme måte som i temperaturhistorikken har man mulighet for å angi datointervall for ønsket visning. Når man åpner alarmlisten vil «Vis alle alarmer» være huket av som standard, og følgelig vises alle alarmene i databasen i listeform. Det kan også sendes e-post fra denne formen, samt eksportere listen til PDF.

4.4 Innstillinger

Som nevnt får man tilgang til programmets innstillinger ved å klikke på «Vis innstillinger» i forsiden. Her er det justeringsmuligheter for parameterne høy og lav alarmgrense for temperatur, lese- og lagringsintervall, og lav alarmgrense for batterispenningen.

Forskjellen på lese- og lagringsintervall er at leseintervallet forteller hvor ofte programmet skal motta data fra Arduino, og lagringsintervallet er hvor ofte temperaturdataene skal lagres i databasen (logges).

Høy og lav alarmgrenser for temperatur angis i grader Celsius, og grensen for batterispenning angis i prosent.

5 OPPSUMERING

Løsningen som leveres har oppnådd spesifikasjonskravene definert i innledningen og use case. En del av målet var å designe et brukergrensesnitt som var lett forståelig og enkelt å ta i bruk. Det mener gruppen at løsningen ble, faktisk så brukervennlig at det nesten er selvforklarende hvordan man skal gå frem for å bruke programmet. De forskjellige formene fordelte funksjonalitetene på en naturlig måte, og forsiden gjør at man får raskt oversikt over status på programmet.

Kodingen i C# er et punkt hvor det nesten alltid kan gjøres forbedringer. Det er vanskelig å si noe om akkurat hvor størst potensialet for forenkling kan gjøres (hadde det vært kjent hadde det naturligvis vært utbedret før innlevering), men det kan antas at samhandling mellom klassene er det punktet som er vanskeligst å løse.

Klassene som er opprettet er naturlig inndelt, eksempelvis tar klassen SQLCom seg av interaksjoner mellom programmet og databasen, klassen ArduinoCom tar seg av all kommunikasjon med Arduino osv. Dette gjør koden oversiktlig og enkel å vedlikeholde, så vel som videreutvikle. Med hensyn til nettopp videreutvikling er det laget et «event» for når alarmer utløses, og e-postklassen abonnerer på dette eventet. Dersom det ønskes å implementere SMS-varsling i alarmsystemet, vil det enkelt kunne abonneres på samme eventet for å aktivere en metode for å sende SMS. Fordeler med dette er at det ikke vil kreves å gjøre endringer i Alarm-klassen, som igjen gjør at denne klassen ikke trenger å rekompileres, og at Alarm klassen ikke «kjenner til» hverken E-post eller den fremtidige SMS-klassen. Dette er prinsippet for «løst koblede» applikasjoner, hvor klassene er så lite avhengige av hverandre som praktisk mulig, som ofte er den ønskelige måten å programmere på [1].

Arv og interface er implementert i programmeringen. Klassen «Alarm» er en abstrakt baseklasse for de arvede alarmklassene (se 2.2 Klassediagram). At en klasse er abstrakt vil si at det ikke kan opprettes et objekt av den. Siden det ikke gir mening å lage et objekt av kun «Alarm» er dette en god måte å hindre at denne type feil oppstår. (Programmet vil ikke kompilere om man prøver å lage et objekt av en abstrakt klasse.)
Interface er implementert i klassen «ProgramParameters», hvor det .NET utviklede interfacet INotifyPropertyChanged er brukt. Dette interfacet krever at et event av typen PropertyChangedEventHandler deklareres i klassen. Funksjonen av dette er at hver gang programparameterne lese- og lagringsintervall, som er egenskaper i klassen, endres vil et event aktiveres. Grunnen til at det ble valgt å implementere dette eventet i nettopp denne klassen er at timerintervallene skal få beskjed om at programparameterne er endret, uten å ha direkte kontakt med klassen «ProgramParamters». [2]

Filhåndtering er inkludert i flere klasser. I «ProgramParameters» skrives og leses parameterne fra tekstfil, og i «PdfForm» opprettes det et PDF-dokument av alarmlisten. Feilhåndtering er brukt i klassene hvor det er størst sannsynlighet for program-crash. Spesielt i klassene som har interaksjon med andre applikasjoner, som «Arduino» og «SQL», er try-catch blokker hyppig brukt.

Det er også valgt å programmere i tråder (threading) i noen elementer av programmet. Dette for å øke ytelse og mulighet for at programmet kan utføre flere oppgaver samtidig. Det er særlig kommunikasjon med Arduino som det har vært gunstig å bruke tråder.

Når det gjelder punkter til forbedring/avvik er dette faktorer som bør utbedres i programmet:

- Det burde vært restriksjoner mot at en alarmtype kan oppstå oftere enn én gang i minuttet
- «ComFault»-alarmen håndterer ikke for hurtige temperaturmålinger, dvs. sensorfeil.

REFERANSER

[1] Margaret Rouse, 2011, *Loose coupling*, URL: http://searchnetworking.techtarget.com/definition/loose-coupling

[2] MSDN, "INotifyPropertyChanged Interface", 2016, URL: https://msdn.microsoft.com/en-us/library/system.componentmodel.inotifypropertychanged(v=vs.110).aspx

VEDLEGG

Vedlegg A WBS

Vedlegg B Gantt

Vedlegg C C#-kode

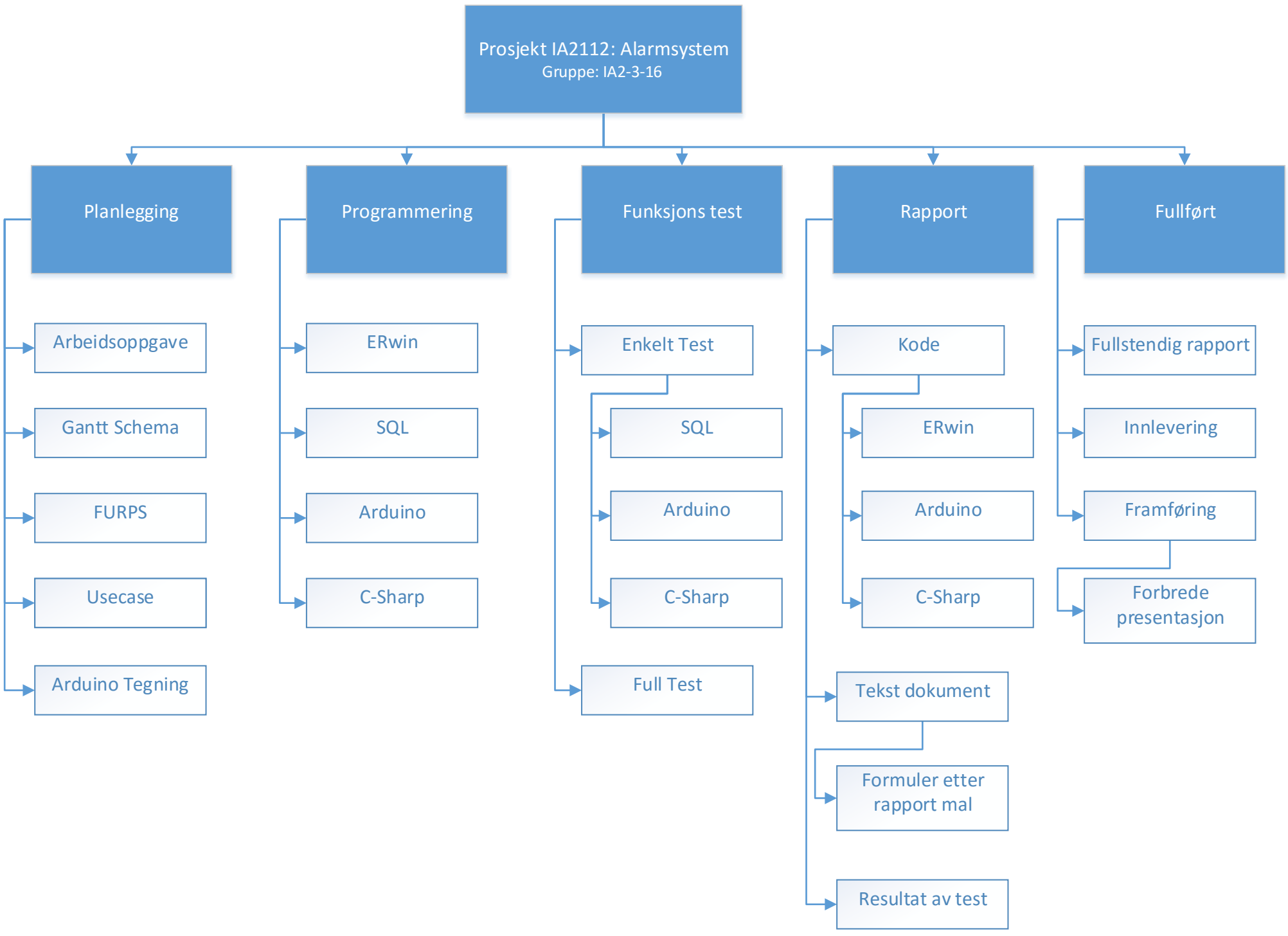
Vedlegg D Arduino-kode

Vedlegg E SQL-script

Vedlegg F Klassediagram

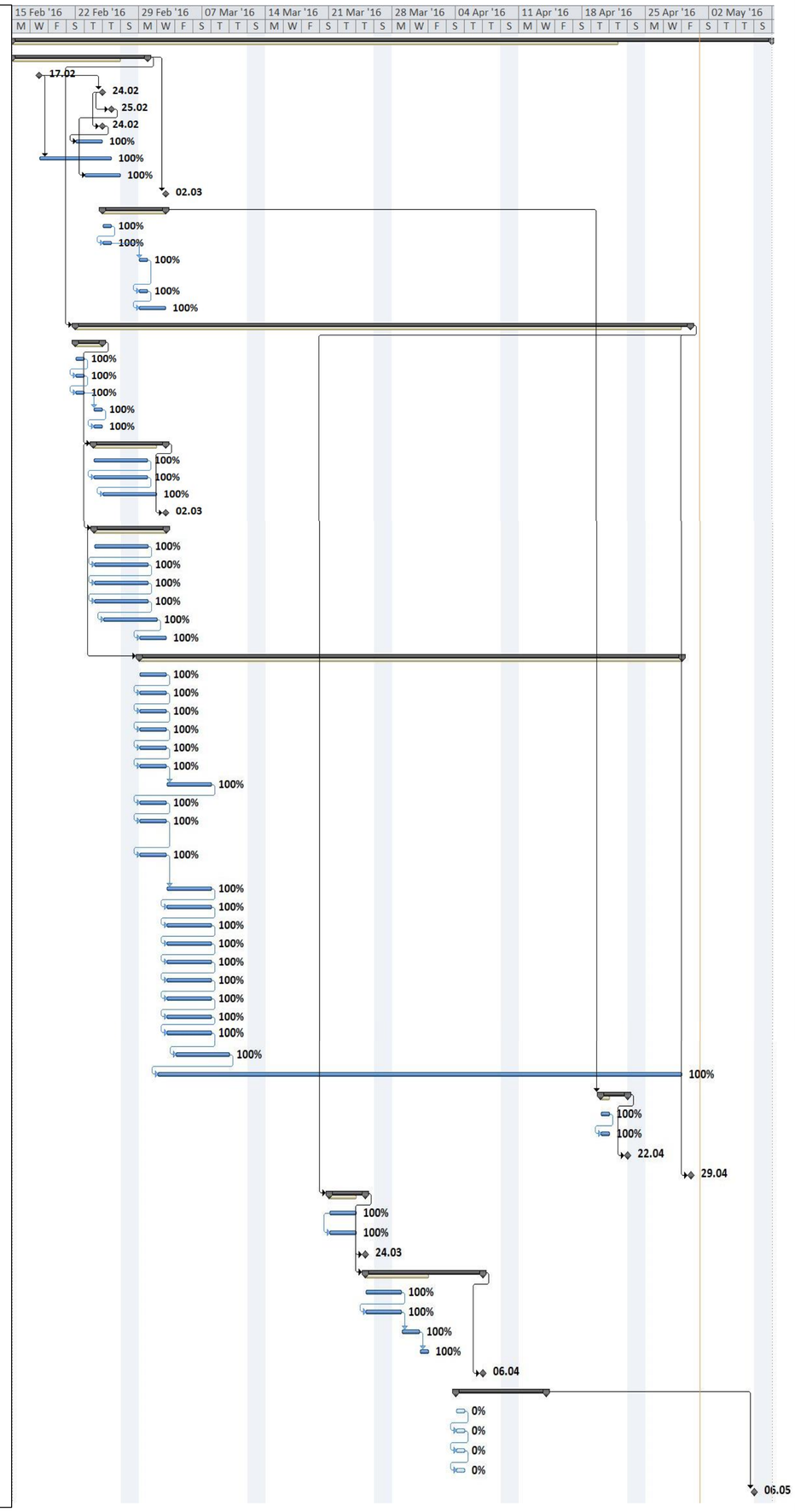
Vedlegg G Use case

VEDLEGG A – WBS – IA2112-2016



VEDLEGG B – GANTT - IA2112 -2016

	Task Mod	Task Name	Duration	Start	Finish	Resource Names
1		**Prosjektoppgave IA2122**	**61 days**	**Mon 15.02.16**	**Sun 08.05.16**	
2		**Planlegging**	**11 days**	**Mon 15.02.16**	**Mon 29.02.16**	**Sander Heldal;Shing Wai Chan;Simen Theie Havenstrøm;Øyvind Skaten**
3		Samarbeidsavtale	0 days?	Mon 15.02.16	Wed 17.02.16	Sander Heldal;Shing Wai Chan;Simen Theie Havenstrøm;Øyvind Skaten
4		WBS	0 days?	Mon 22.02.16	Wed 24.02.16	Shing Wai Chan
5		FURPS	0 days?	Mon 22.02.16	Thu 25.02.16	Øyvind Skaten;Simen Theie Havenstrøm
6		Definer Oppgave	0 days?	Wed 17.02.16	Wed 24.02.16	Sander Heldal;Shing Wai Chan;Simen Theie Havenstrøm;Øyvind Skaten
7		Inndeling av Arbeid	3 days?	Mon 22.02.16	Wed 24.02.16	Sander Heldal;Shing Wai Chan;Simen Theie Havenstrøm;Øyvind Skaten
8		Gantt Schema	6 days?	Thu 18.02.16	Thu 25.02.16	Shing Wai Chan;Simen Theie Havenstrøm
9		Usecase	4 days?	Tue 23.02.16	Fri 26.02.16	Sander Heldal;Shing Wai Chan;Simen Theie Havenstrøm;Øyvind Skaten
10		Planlegging ferdig	1 day	Wed 02.03.16	Wed 02.03.16	
11		**1.Møte**	**5 days**	**Thu 25.02.16**	**Wed 02.03.16**	
12		Lag møte innkalling	1 day?	Thu 25.02.16	Thu 25.02.16	Shing Wai Chan
13		Send møte innkalling	1 day?	Thu 25.02.16	Thu 25.02.16	Shing Wai Chan
14		Bestem møte leder og møte referent	1 day?	Mon 29.02.16	Mon 29.02.16	Sander Heldal;Shing Wai Chan;Simen Theie Havenstrøm;Øyvind Skaten
15		Lag møte referat	1 day?	Mon 29.02.16	Mon 29.02.16	Simen Theie Havenstrøm
16		Send møte referat til deltager	3 days?	Mon 29.02.16	Wed 02.03.16	Simen Theie Havenstrøm
17		**Programmering**	**50 days**	**Mon 22.02.16**	**Fri 29.04.16**	
18		**Erwin**	**3 days**	**Mon 22.02.16**	**Wed 24.02.16**	**Sander Heldal;Simen Theie Havenstrøm**
19		Lag ERwin Schema	1 day?	Mon 22.02.16	Mon 22.02.16	Sander Heldal;Simen Theie Havenstrøm
20		Lag Diagram for data	1 day?	Mon 22.02.16	Mon 22.02.16	Sander Heldal;Simen Theie Havenstrøm
21		Definer Data	1 day?	Mon 22.02.16	Mon 22.02.16	Sander Heldal;Simen Theie Havenstrøm;Øyvind Skaten
22		Lage Schema	1 day?	Wed 24.02.16	Wed 24.02.16	Sander Heldal;Simen Theie Havenstrøm
23		Lage script	1 day?	Wed 24.02.16	Wed 24.02.16	Sander Heldal;Simen Theie Havenstrøm
24		**SQL**	**6 days**	**Wed 24.02.16**	**Wed 02.03.16**	**Sander Heldal;Simen Theie Havenstrøm;Øyvind Skaten**
25		Lag SQL med data fra Erwin	4 days?	Wed 24.02.16	Mon 29.02.16	Simen Theie Havenstrøm
26		Lag Spørringer	4 days?	Wed 24.02.16	Mon 29.02.16	Simen Theie Havenstrøm;Øyvind Skaten
27		Test Input og Output Data	4 days?	Thu 25.02.16	Tue 01.03.16	Simen Theie Havenstrøm;Øyvind Skaten
28		Fullført halve koden	0 days	Wed 02.03.16	Wed 02.03.16	
29		**Arduino**	**6 days**	**Wed 24.02.16**	**Wed 02.03.16**	**Shing Wai Chan**
30		Sett opp Arduino	4 days?	Wed 24.02.16	Mon 29.02.16	Shing Wai Chan
31		Koble opp komponenter	4 days?	Wed 24.02.16	Mon 29.02.16	Shing Wai Chan
32		Lag program for Ardunio	4 days?	Wed 24.02.16	Mon 29.02.16	Shing Wai Chan
33		Program for sensor	4 days?	Wed 24.02.16	Mon 29.02.16	Shing Wai Chan
34		Program for visning på LCD skjerm	4 days?	Thu 25.02.16	Tue 01.03.16	Shing Wai Chan
35		Test programmet	3 days?	Mon 29.02.16	Wed 02.03.16	Shing Wai Chan
36		**C-sharp**	**44 days**	**Mon 29.02.16**	**Thu 28.04.16**	
37		Lag C-sharp applikasjon struktur	3 days?	Mon 29.02.16	Wed 02.03.16	Øyvind Skaten
38		Programmer for temperaturføler	3 days	Mon 29.02.16	Wed 02.03.16	Shing Wai Chan
39		Lagring av temperaturføler	3 days	Mon 29.02.16	Wed 02.03.16	Øyvind Skaten
40		Programmer for bevegelseføler	3 days	Mon 29.02.16	Wed 02.03.16	Shing Wai Chan
41		Lagring av bevegelseføler	3 days	Mon 29.02.16	Wed 02.03.16	Øyvind Skaten
42		Visning av alle sensor verdier	3 days	Mon 29.02.16	Wed 02.03.16	Øyvind Skaten
43		Sending av e-mail	3 days	Thu 03.03.16	Mon 07.03.16	Sander Heldal
44		Sending av pdf	3 days	Mon 29.02.16	Wed 02.03.16	Sander Heldal
45		Send mail ved brudd på kommuikasjon av database	3 days	Mon 29.02.16	Wed 02.03.16	Øyvind Skaten
46		Send mail ved brudd på kommuikasjon av Arduino	3 days	Mon 29.02.16	Wed 02.03.16	Sander Heldal
47		For endring av sensor verdi	3 days	Thu 03.03.16	Mon 07.03.16	Øyvind Skaten
48		For visning av alle alarm	3 days	Thu 03.03.16	Mon 07.03.16	Øyvind Skaten
49		For visning av alle alarm log	3 days	Thu 03.03.16	Mon 07.03.16	Øyvind Skaten
50		For visning av valgte alarm	3 days	Thu 03.03.16	Mon 07.03.16	Øyvind Skaten
51		Kommunikasjon med Ardunio	3 days	Thu 03.03.16	Mon 07.03.16	Shing Wai Chan
52		Kommunikasjon med database	3 days	Thu 03.03.16	Mon 07.03.16	Simen Theie Havenstrøm;Øyvind Skaten
53		Lagre alle alarm til pdf	3 days	Thu 03.03.16	Mon 07.03.16	Sander Heldal
54		Programmer for batterispenning	3 days	Thu 03.03.16	Mon 07.03.16	Øyvind Skaten
55		Test C-sharp	4 days	Fri 04.03.16	Wed 09.03.16	Sander Heldal;Shing Wai Chan;Simen Theie Havenstrøm;Øyvind Skaten
56		Debugging	42 days?	Wed 02.03.16	Thu 28.04.16	Sander Heldal;Shing Wai Chan;Simen Theie Havenstrøm;Øyvind Skaten
57		**2.Møte**	**3 days**	**Wed 20.04.16**	**Fri 22.04.16**	
58		Lag møte innkalling	1 day?	Wed 20.04.16	Wed 20.04.16	Øyvind Skaten
59		Send møte innkalling	1 day?	Wed 20.04.16	Wed 20.04.16	Øyvind Skaten
60		2 møter ferdig	0 days	Fri 22.04.16	Fri 22.04.16	
61		Fullført hele koden	0 days	Fri 29.04.16	Fri 29.04.16	
62		**Dokumenter koder**	**4 days**	**Mon 21.03.16**	**Thu 24.03.16**	
63		Kopier C-sharp kode til pdf	3 days?	Mon 21.03.16	Wed 23.03.16	Sander Heldal;Shing Wai Chan;Simen Theie Havenstrøm;Øyvind Skaten
64		Kopier Arduino kode til pdf	3 days?	Mon 21.03.16	Wed 23.03.16	Shing Wai Chan
65		Halve rapport ferdig	0 days	Thu 24.03.16	Thu 24.03.16	
66		**Rapport**	**9 days**	**Fri 25.03.16**	**Wed 06.04.16**	
67		Lag utskrift til Rapporten	2 days	Fri 25.03.16	Mon 28.03.16	Simen Theie Havenstrøm
68		Rett opp feil I rapporten	2 days	Fri 25.03.16	Mon 28.03.16	Sander Heldal;Shing Wai Chan;Simen Theie Havenstrøm;Øyvind Skaten
69		Sjekk rapporten før innlevering	2 days	Tue 29.03.16	Wed 30.03.16	Sander Heldal;Shing Wai Chan;Simen Theie Havenstrøm;Øyvind Skaten
70		Rapport levert inn	1 day	Thu 31.03.16	Thu 31.03.16	Sander Heldal;Shing Wai Chan;Simen Theie Havenstrøm;Øyvind Skaten
71		Hele rapport ferdig	0 days	Wed 06.04.16	Wed 06.04.16	
72		**Framføring**	**8 days**	**Mon 04.04.16**	**Wed 13.04.16**	
73		Lage presentasjon	1 day?	Mon 04.04.16	Mon 04.04.16	Sander Heldal;Shing Wai Chan;Simen Theie Havenstrøm;Øyvind Skaten
74		Video/ fremvisning	1 day?	Mon 04.04.16	Mon 04.04.16	Sander Heldal;Shing Wai Chan;Simen Theie Havenstrøm;Øyvind Skaten
75		Øving av framføring	1 day?	Mon 04.04.16	Mon 04.04.16	Sander Heldal;Shing Wai Chan;Simen Theie Havenstrøm;Øyvind Skaten
76		Framføring	1 day?	Mon 04.04.16	Mon 04.04.16	Sander Heldal;Shing Wai Chan;Simen Theie Havenstrøm;Øyvind Skaten
77		Prosjekt Avsluttet	0 days	Fri 06.05.16	Fri 06.05.16	



VEDLEGG C – C# Kode – IA2112 – 2016

Home.cs (form)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using SQLCommunication;
using System.IO;
using System.Windows.Forms.DataVisualization.Charting;

namespace Prosjekt
{
    public partial class Home : Form
    {
        ArduinoCom arduinoCom;
        Arduino arduino;

        AlarmNoCharge noChargeAlarm;
        AlarmLowBattery lowBatteryAlarm;
        AlarmLowTemperature lowTemperatureAlarm;
        AlarmHighTemperature highTemperatureAlarm;
        AlarmMotion motionAlarm;
        AlarmComFault comFaultAlarm;

        List<string> emailList = new List<string>();
        Mail[] mailArray;

        const int timeListLength = 20;
        List<double> temperatureList = new List<double>();
        List<double> timeList = new List<double>();
        DateTime clock;
        double xAxisMinimum;
        double xAxisMaximum;

        int timeLeftToActivateMotion = 30;

        public static ProgramParameter ProgramParameter {get; set; }

        public Home()
        {
            InitializeComponent();


            SQLCom.AutoConnect();
            if (SQLCom.SuccessfullLogin == true)
            {
                SQL.ConnectionString = SQLCom.ConnectionString;
            }
            else
            {
                this.WindowState = FormWindowState.Minimized;
                this.ShowInTaskbar = false;
                Form frmServerConnect = new ServerConnect();
                frmServerConnect.ShowDialog();
```

```
            this.ShowInTaskbar = true;
            this.WindowState = FormWindowState.Normal;
        }
        ProgramParameter = new ProgramParameter();
        ProgramParameter.PropertyChanged += IntervalChanged;
        tmrRead.Interval = ProgramParameter.ReadInterval * 1000;
        tmrLog.Interval = ProgramParameter.LogInterval * (1000 * 60);

        arduinoCom = new ArduinoCom();
        arduino = new Arduino(arduinoCom.SendCommandToArduino(Command.ReadStatus));

        lowTemperatureAlarm = new AlarmLowTemperature
                (SQL.GetAlarmLimit(AlarmtypesEnum.LowTemp));
        lowTemperatureAlarm.Temperature = arduino.Temperature;
        lowTemperatureAlarm.LowTemperatureAlarm += LowTemperatureAlarm;

        highTemperatureAlarm = new AlarmHighTemperature
                (SQL.GetAlarmLimit(AlarmtypesEnum.HighTemp));
        highTemperatureAlarm.Temperature = arduino.Temperature;
        highTemperatureAlarm.HighTemperatureAlarm += HighTemperatureAlarm;

        noChargeAlarm = new AlarmNoCharge();
        noChargeAlarm.PowerStatusChanged += UpdatePowerStatus;
        noChargeAlarm.NoChargeAlarm += NoChargeAlarm;
        UpdatePowerStatus(null, EventArgs.Empty);

        comFaultAlarm = new AlarmComFault();
        comFaultAlarm.ComFaultAlarm += ComFaultAlarm;

        motionAlarm = new AlarmMotion();
        motionAlarm.AlarmActivated = false;
        motionAlarm.MotionAlarm += MotionAlarm;

        lowBatteryAlarm = new AlarmLowBattery(SQL.GetAlarmLimit(AlarmtypesEnum.LowBattery));
        lowBatteryAlarm.LowBatteryAlarm += LowBatteryAlarm;

        InitGraph();

        tmrRead.Start();
        tmrLog.Start();

        Update();
        UpdateTop5Alarms();
    }

    // AlarmEvents
    #region AlarmEvents

    private void ComFaultAlarm(object sender, EventArgs args)
    {
        SQL.NewAlarm(comFaultAlarm.AlarmType);

        emailList.Clear();
        emailList = SQL.GetEmailList(comFaultAlarm.AlarmType);
        Task sendEmailToSubscribersTask = Task.Factory.StartNew(()
          => SendEmailToSubscribers(emailList, comFaultAlarm.AlarmDescription));

        MessageBox.Show(comFaultAlarm.AlarmDescription, comFaultAlarm.AlarmTitle,
          MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        UpdateTop5Alarms();
    }
```

```
private void MotionAlarm(object sender, EventArgs args)
{
    SQL.NewAlarm(motionAlarm.AlarmType);

    arduinoCom.SendCommandToArduino(Command.Alarm);

    emailList.Clear();
    emailList = SQL.GetEmailList(motionAlarm.AlarmType);
    Task sendEmailToSubscribersTask = Task.Factory.StartNew(()
      => SendEmailToSubscribers(emailList, motionAlarm.AlarmDescription));

    MessageBox.Show(motionAlarm.AlarmDescription, motionAlarm.AlarmTitle,
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    UpdateTop5Alarms();
}



private void HighTemperatureAlarm(object source, EventArgs args)
{
    SQL.NewAlarm(highTemperatureAlarm.AlarmType);

    arduinoCom.SendCommandToArduino(Command.Alarm);

    emailList.Clear();
    emailList = SQL.GetEmailList(highTemperatureAlarm.AlarmType);
    Task sendEmailToSubscribersTask = Task.Factory.StartNew(()
      => SendEmailToSubscribers(emailList, highTemperatureAlarm.AlarmDescription));

    MessageBox.Show(highTemperatureAlarm.AlarmDescription,
            highTemperatureAlarm.AlarmTitle, MessageBoxButtons.OK,
            MessageBoxIcon.Exclamation);
    UpdateTop5Alarms();
}

private void LowTemperatureAlarm(object source, EventArgs args)
{
    SQL.NewAlarm(lowTemperatureAlarm.AlarmType);

    arduinoCom.SendCommandToArduino(Command.Alarm);

    emailList.Clear();
    emailList = SQL.GetEmailList(lowTemperatureAlarm.AlarmType);
    Task sendEmailToSubscribersTask = Task.Factory.StartNew(()
      => SendEmailToSubscribers(emailList, lowTemperatureAlarm.AlarmDescription));

     MessageBox.Show(lowTemperatureAlarm.AlarmDescription,
            lowTemperatureAlarm.AlarmTitle,
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    UpdateTop5Alarms();
}

void NoChargeAlarm(object sender, EventArgs e)
{
    SQL.NewAlarm(noChargeAlarm.AlarmType);

    arduinoCom.SendCommandToArduino(Command.Alarm);

    emailList.Clear();
    emailList = SQL.GetEmailList(noChargeAlarm.AlarmType);
    Task sendEmailToSubscribersTask = Task.Factory.StartNew(()
```

```
        => SendEmailToSubscribers(emailList, noChargeAlarm.AlarmDescription));

    MessageBox.Show(noChargeAlarm.AlarmDescription,
            noChargeAlarm.AlarmTitle, MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    UpdateTop5Alarms();
}

void LowBatteryAlarm(object sender, EventArgs e)
{
    SQL.NewAlarm(lowBatteryAlarm.AlarmType);

    arduinoCom.SendCommandToArduino(Command.Alarm);

    emailList.Clear();
    emailList = SQL.GetEmailList(lowBatteryAlarm.AlarmType);
    Task sendEmailToSubscribersTask = Task.Factory.StartNew(()
        => SendEmailToSubscribers(emailList, lowBatteryAlarm.AlarmDescription));

    MessageBox.Show(lowBatteryAlarm.AlarmDescription, lowBatteryAlarm.AlarmTitle,
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    UpdateTop5Alarms();
}
#endregion

// Updates if power is connected
void UpdatePowerStatus(object source, EventArgs e)
{
    if (noChargeAlarm.PowerConnected == true)
    {
        lblPowerAdapter.Text = "Tilkoblet";
        lblPowerAdapter.ForeColor = SystemColors.Highlight;
    }
    else
    {
        lblPowerAdapter.Text = "Ikke tilkoblet";
        lblPowerAdapter.ForeColor = Color.Firebrick;
    }
}

// Inits graph with XaxisMin and Max
private void InitGraph()
{
    xAxisMinimum = DateTime.Now.ToOADate();
    xAxisMaximum = DateTime.Now.AddSeconds
            (ProgramParameter.ReadInterval * timeListLength).ToOADate();
    graph.ChartAreas[0].AxisX.Minimum = xAxisMinimum;
    graph.ChartAreas[0].AxisX.Maximum = xAxisMaximum;
}

// Updates graph data
void UpdateGraphData()
{
    clock = DateTime.Now;
    timeList.Add(clock.ToOADate());
    temperatureList.Add(arduino.Temperature);
    if (timeList.Count > timeListLength)
    {
        timeList.RemoveAt(0);
        temperatureList.RemoveAt(0);
    }
    xAxisMinimum = timeList[0];
    if (timeList.Count >= timeListLength) xAxisMaximum = timeList[timeListLength - 1];
```

```
}
// Updates graph with new data. Use task for datacalculations to save some resources on
main thread.
void UpdateGraph()
{
    Task updateGraphDataTask = Task.Factory.StartNew(UpdateGraphData);
    Task.WaitAll(updateGraphDataTask);
    graph.Series[0].Points.Clear();
    graph.ChartAreas[0].AxisX.Minimum = xAxisMinimum;
    graph.ChartAreas[0].AxisX.Maximum = xAxisMaximum;

    for (int i = 0; i < timeList.Count; i++)
    {
        graph.Series[0].Points.AddXY(timeList[i], temperatureList[i]);
    }
}

// Updates top 5 alarms table
void UpdateTop5Alarms()
{
    string sqlQuery = @"SELECT TOP 5 Alarm as Alarmtype,
            Time as Tidspunkt FROM ALARM ORDER BY Time DESC";
    dgvTop5Alarms.DataSource = SQL.MakeDataGridView(sqlQuery);
}
/// <summary>
/// Send email to subscribers based on emails list
/// </summary>
/// <param name="emails">List of emails to subscribers</param>
/// <param name="alarmDescription">Description of alarmtype</param>
/// <returns></returns>
StringBuilder SendEmailToSubscribers(List<string> emails, string alarmDescription)
{
    StringBuilder allEmails = new StringBuilder();
    mailArray = new Mail[emailList.Count];

    for (int i = 0; i <= mailArray.GetUpperBound(0); i++)
    {
        int index = i;
        mailArray[index] = new Mail(emailList[index], "Alarmsystem", alarmDescription);
    }
    for (int i = 0; i <= mailArray.GetUpperBound(0); i++)
    {
        int index = i;
        mailArray[index].Send();
    }
    for (int i = 0; i <= mailArray.GetUpperBound(0); i++)
    {
        int index = i;
        if (mailArray[index].EmailSent) allEmails.AppendLine(emails[index]);
    }
    return allEmails;
}



private void addNewUserToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form frmAddSubscriber = new AddSubscriber();
    frmAddSubscriber.ShowDialog(this);
}

private void administrateUsersToolStripMenuItem_Click(object sender, EventArgs e)
```

```
{
    Form frmAdministrateSubscribers = new AdministrateSubscribers();
    frmAdministrateSubscribers.ShowDialog(this);
}

private void avsluttToolStripMenuItem_Click(object sender, EventArgs e)
{
    Application.Exit();
}

private void aboutProgramToolStripMenuItem_Click(object sender, EventArgs e)
{
    AboutBox about = new AboutBox();
    about.ShowDialog(this);
}


private void btnExportToPdf_Click(object sender, EventArgs e)
{
    CreatePdf();
}

void CreatePdf()
{
    string sqlQuery = @"SELECT Alarm as Alarmtype,
            Time as Tidspunkt FROM ALARM ORDER BY Time DESC";
    PdfForm.CreatePdf("Viser alle alarmer i databasen", SQL.MakeDataGridView(sqlQuery));
}

private void btnShowAlarmlist_Click(object sender, EventArgs e)
{
    AlarmList frmAlarmList = new AlarmList();
    frmAlarmList.ShowDialog(this);
}

private void btnSendEmail_Click(object sender, EventArgs e)
{
    Task createPdfTask = Task.Factory.StartNew(() => CreatePdf());
    SendEmail frmSendEmail = new SendEmail();
    Task.WaitAll(createPdfTask);
    frmSendEmail.ShowDialog(this);
}

private void btnShowGraph_Click(object sender, EventArgs e)
{
    HistoricGraph frmGraph = new HistoricGraph();
    frmGraph.Show();
}

private void btnSettings_Click(object sender, EventArgs e)
{
    Settings frmSettings = new Settings();
    frmSettings.ShowDialog(this);
    lowBatteryAlarm.LowBatteryLimit = SQL.GetAlarmLimit(lowTemperatureAlarm.AlarmType);
    lowTemperatureAlarm.LowTemperatureLimit =
            SQL.GetAlarmLimit(lowTemperatureAlarm.AlarmType);
    highTemperatureAlarm.HighTemperatureLimit =
            SQL.GetAlarmLimit(highTemperatureAlarm.AlarmType);
}

private void IntervalChanged (object sender, EventArgs e)
{
```

```
    tmrRead.Interval = ProgramParameter.ReadInterval * 1000;
    tmrLog.Interval = ProgramParameter.LogInterval * (1000 * 60);
}

private void btnActivateMotionAlarm_Click(object sender, EventArgs e)
{
    MessageBox.Show("Bevegelsesdetektor vil aktiveres innen 30 sekund",
            "Bevegelsesdetektor", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    timeLeftToActivateMotion = 30;
    tmrActivateMotion.Start();
    lblMotion.Text = "Aktiverer - " + timeLeftToActivateMotion;
    lblMotion.ForeColor = Color.Goldenrod;

}

private void btnDeactivateMotionAlarm_Click(object sender, EventArgs e)
{
    tmrActivateMotion.Stop();
    motionAlarm.AlarmActivated = false;
    lblMotion.Text = "Deaktivert";
    lblMotion.ForeColor = SystemColors.Highlight;
}

private void tmrRead_Tick(object sender, EventArgs e)

{
    arduino.Update(arduinoCom.SendCommandToArduino(Command.ReadStatus));
    lowTemperatureAlarm.Temperature = arduino.Temperature;
    highTemperatureAlarm.Temperature = arduino.Temperature;
    motionAlarm.InAlarm = arduino.PirEnabled;
    comFaultAlarm.ArduinoConnected = arduino.Connected;
    lblTemperature.Text = arduino.Temperature.ToString() + " °C";
    lblBatteryPercent.Text = lowBatteryAlarm.CurrentBatteryPercent.ToString() + " %";
    if (arduino.Connected) lblArduinoConnected.Text = "Connected";
    else lblArduinoConnected.Text = "Not Connected";
    if (lowBatteryAlarm.CurrentBatteryPercent > lowBatteryAlarm.LowBatteryLimit)
            lblBatteryPercent.ForeColor = SystemColors.Highlight;
    else lblBatteryPercent.ForeColor = Color.Firebrick;
    UpdateGraph();
}

private void tmrActivateMotion_Tick(object sender, EventArgs e)
{
    if (timeLeftToActivateMotion > 0)
    {
        timeLeftToActivateMotion = timeLeftToActivateMotion - 1;
        lblMotion.Text = "Aktiverer - " + timeLeftToActivateMotion;
    }
    else
    {
        tmrActivateMotion.Stop();
        motionAlarm.AlarmActivated = true;
        lblMotion.Text = "Aktivert";
        lblMotion.ForeColor = SystemColors.Highlight;
    }
}

private void tmrLog_Tick(object sender, EventArgs e)
{
    SQL.LogTemp( arduino.Temperature);
}
```

```
    }
}
```

Alarm.cs (class)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Microsoft.Win32;
using SQLCommunication;

namespace Prosjekt
{
    abstract class Alarm
    {
        protected bool state;
        protected bool oldState;
        public string AlarmDescription { get; protected set; }
        public string AlarmTitle { get { return "Alarm"; } }
        public AlarmtypesEnum AlarmType { get; protected set; }
    }

    class AlarmLowTemperature : Alarm
    {
        // This class depends on that Properties is set to raise event for low temperature alarm
        protected double lowTempLimit;
        protected double temperature;

        public delegate void LowTemperatureEventHandler(object source, EventArgs args);
        public event LowTemperatureEventHandler LowTemperatureAlarm;

        // Constructor
        public AlarmLowTemperature(double lowTempLimit)
        {
            AlarmType = AlarmtypesEnum.LowTemp;
            AlarmDescription = SQL.GetDescription(AlarmType);
            this.lowTempLimit = lowTempLimit;
        }

        protected void OnLowTemperatureAlarm()
        {
            if (LowTemperatureAlarm != null)
            {
                LowTemperatureAlarm(this, EventArgs.Empty);
            }
        }

        public double LowTemperatureLimit
        {
            get { return lowTempLimit; }
            set
            {
                lowTempLimit = value;
                oldState = state;
                if (temperature > lowTempLimit)
                {
                    state = false;
                }
```

```
            else if (oldState == false)
            {
                state = true;
                OnLowTemperatureAlarm();
            }
        }
    }

    public double Temperature
    {
        get { return temperature; }
        set
        {
            temperature = value;
            oldState = state;
            if (temperature > lowTempLimit)
            {
                state = false;
            }
            else if (oldState == false)
            {
                state = true;
                OnLowTemperatureAlarm();
            }
        }
    }
}

class AlarmHighTemperature : Alarm
{
    // This class depends on that Properties is set to raise event
   for high temperature alarm
    protected double highTempLimit;
    protected double temperature;
    public delegate void HighTemperatureEventHandler(object source, EventArgs args);
    public event HighTemperatureEventHandler HighTemperatureAlarm;

    // Constructor
    public AlarmHighTemperature (double highTempLimit)
    {
        AlarmType = AlarmtypesEnum.HighTemp;
        AlarmDescription = SQL.GetDescription(AlarmType);
        this.highTempLimit = highTempLimit;
    }

    protected void OnHighTemperatureAlarm()
    {
        if (HighTemperatureAlarm != null)
        {
            HighTemperatureAlarm (this, EventArgs.Empty);
        }
    }
    public double Temperature
    {
        get { return temperature; }
        set
        {
            temperature = value;
            oldState = state;
            if (temperature < highTempLimit)
            {
                state = false;
```

```
            }
            else if (oldState == false)
            {
                state = true;
                OnHighTemperatureAlarm();
            }
        }
    }

    public double HighTemperatureLimit
    {
        get { return highTempLimit; }
        set
        {
            highTempLimit = value;
            oldState = state;
            if (temperature < highTempLimit)
            {
                state = false;
            }
            else if (oldState == false)
            {
                state = true;
                OnHighTemperatureAlarm();
            }
        }
    }
}

class AlarmMotion : Alarm
{
    // This class raises event when AlarmActivated property is set to true
    public bool AlarmActivated { get; set; }

    public delegate void MotionAlarmEventHandler(object sender, EventArgs args);
    public event MotionAlarmEventHandler MotionAlarm;

    // Constructor
    public AlarmMotion()
    {
        AlarmType = AlarmtypesEnum.Motion;
        AlarmDescription = SQL.GetDescription(AlarmType);

    }

    protected void OnMotionAlarm()
    {
        if (MotionAlarm != null)
        {
            MotionAlarm (this, EventArgs.Empty);
        }
    }

    public bool InAlarm
    {
        get { return state; }
        set
        {
            oldState = state;
            state = value;
            if (AlarmActivated && oldState == false && state == true)
            {
```

```
                OnMotionAlarm();
            }
        }
    }
}

class AlarmNoCharge : Alarm
{
    // This class subscribes to System Event PowerModeChanged which is a static propery.
    // Thats why the descructor is used
    public bool PowerConnected { get; protected set; }

    public delegate void PowerStatusChangedEventHandler (object source, EventArgs args);
    public event PowerStatusChangedEventHandler PowerStatusChanged;
    public delegate void NoChargeAlarmEventHandler(object source, EventArgs args);
    public event NoChargeAlarmEventHandler NoChargeAlarm;

    // Constructor
    public AlarmNoCharge()
    {
        AlarmType = AlarmtypesEnum.NoCharge;
        AlarmDescription = SQL.GetDescription(AlarmType);
        SystemEvents.PowerModeChanged += PowerChange;
        CheckPowerLineStatus();

        // Displays warning message at startup if power is not connected
        if (!PowerConnected) MessageBox.Show
          (AlarmDescription, AlarmTitle, MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }

    private void CheckPowerLineStatus()
    {
        switch (SystemInformation.PowerStatus.PowerLineStatus)
        {
            case PowerLineStatus.Online:
                PowerConnected = true;
                break;
            default:
                PowerConnected = false;
                break;
        }
    }
    private void PowerChange(object sender, PowerModeChangedEventArgs ev)
    {
        oldState = state;
        CheckPowerLineStatus();
        OnPowerStatusChanged();

        if (oldState == false && PowerConnected == false)
        {
            state = true;
            OnNoChargeAlarm();
        }
    }

    protected void OnPowerStatusChanged ()
    {
        if (PowerStatusChanged != null)
        {
            PowerStatusChanged(this, EventArgs.Empty);
        }
    }
```

```
    protected void OnNoChargeAlarm()
    {
        if (NoChargeAlarm != null)
        {
            NoChargeAlarm(this, EventArgs.Empty);
        }
    }

    // Destructor
    // Quote from MSDN: "Because this is a static event, you must detach your event
    //   handlers when your application is disposed, or memory leaks will result."
    ~AlarmNoCharge()
    {
        SystemEvents.PowerModeChanged -= PowerChange;
    }
}

class AlarmLowBattery : Alarm
{
    // This class create a timer to update the CurrentBatteryPercent Property ever 20 sec
    public double CurrentBatteryPercent { get; protected set; }
    public double LowBatteryLimit { get; set; }
    public delegate void LowBatteryAlarmEventHandler(object source, EventArgs args);
    public event LowBatteryAlarmEventHandler LowBatteryAlarm;

    // Constructor
    public AlarmLowBattery(double lowBatteryPercent)
    {
        AlarmType = AlarmtypesEnum.LowBattery;
        AlarmDescription = SQL.GetDescription(AlarmType);
        LowBatteryLimit = lowBatteryPercent;

        // Creates timer and ChechBatteryPercent method to tick-event
        Timer batteryUpdate = new Timer();
        batteryUpdate.Interval = 20000;
        batteryUpdate.Start();
        batteryUpdate.Tick += CheckBatteryPercent;

        CheckBatteryPercent(null, EventArgs.Empty);

        // Displays warning at startup if batterypercent is lower then limit
        if (CurrentBatteryPercent <= LowBatteryLimit) MessageBox.Show
          (AlarmDescription, AlarmTitle, MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }

    protected void CheckBatteryPercent(object sender, EventArgs e)
    {
        CurrentBatteryPercent =
                Math.Round( (SystemInformation.PowerStatus.BatteryLifePercent * 100), 1);
        oldState = state;
        if (CurrentBatteryPercent > LowBatteryLimit)
        {
            state = false;
        }
        else if (oldState == false)
        {
            state = true;
            OnLowBatteryAlarm();
        }
    }
```

```
        protected void OnLowBatteryAlarm()
        {
            if (LowBatteryAlarm != null)
            {
                LowBatteryAlarm(this, EventArgs.Empty);
            }
        }
    }

    class AlarmComFault : Alarm
    {
        // This class could only raise an event when the arduino have been connected
       to the computer.
        // If it isn't connected at startup, it should not trigger alarmevent.
        protected bool arduinoConnected;
        protected bool startup;
        public delegate void ComFaultAlarmEventhandler(object sender, EventArgs args);
        public event ComFaultAlarmEventhandler ComFaultAlarm;

        // Constructor
        public AlarmComFault()
        {
            AlarmType = AlarmtypesEnum.ComFault;
            AlarmDescription = SQL.GetDescription(AlarmType);
            startup = true;
        }

        protected void OnComFaultAlarm()
        {
            if (ComFaultAlarm != null)
            {
                ComFaultAlarm(this, EventArgs.Empty);
            }
        }

        public bool ArduinoConnected
        {
            get { return arduinoConnected; }
            set
            {
                arduinoConnected = value;
                oldState = state;
                if (arduinoConnected)
                {
                    startup = false;
                    state = false;
                }
                else if (!arduinoConnected && oldState == false && startup == false)
                {
                    state = true;
                    OnComFaultAlarm();
                }
            }
        }
    }

}
```

AddSubscriber.cs (form)

```
using System;
```

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using SQLCommunication;

namespace Prosjekt
{
    public partial class AddSubscriber : Form
    {
        public AddSubscriber()
        {
            InitializeComponent();
        }

        // Close form if cancel-button is clicked
        private void btnCancel_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void btnAddUser_Click(object sender, EventArgs e)
        {
            bool resultTlfConvert;
            string email = txtEMail.Text;
            string firstName = txtFirstName.Text;
            string lastName = txtLastName.Text;
            Int32 tlfNumber;
            resultTlfConvert = int.TryParse(txtPhoneNumber.Text, out tlfNumber);

            // Test if txtPhoneNumber successfully converts to int
            if (resultTlfConvert == true)
            {
                Subscriber sub = new Subscriber(email, firstName, lastName, tlfNumber);

                // Add subscriber if it don't exist in database
                if (!SQL.SubscriberExists(sub))
                {
                    SQL.AddSubscriber(sub);
                    foreach (CheckBox chk in grpAlarmtypes.Controls)
                    {
                        if (chk.Checked)
                        {
                            // Removes chk from checkbox-name, and casts
                            string to AlarmtypesEnum. Then it subscribes to alarmtype.
                            string temp = chk.Name;
                            temp = temp.Replace("chk", "");
                            AlarmtypesEnum alarm = (AlarmtypesEnum)
                                  Enum.Parse(typeof(AlarmtypesEnum), temp);
                            sub.SubscribeTo(alarm);
                        }
                    }
                    MessageBox.Show("Bruker lagt til i databasen", "",
                            MessageBoxButtons.OK, MessageBoxIcon.Information);
                    ClearAll();
                }
```

```
                // If subscriber exists, display error-message
                  else if (SQL.SubscriberExists(sub)) MessageBox.Show("Bruker med samme e-post
eksisterer i databasen", "Feil", MessageBoxButtons.OK, MessageBoxIcon.Error);

                  // Display this error-message if everything else fails
                  else MessageBox.Show("Oppretting av bruker feilet",
                        "Feil", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }

            // Converting to int fails of txtPhoneNumber holds characters
            else
            {
                MessageBox.Show("Telefonnummer skal kun inneholde tall",
                      "Feil", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        // Clears all text- and checkboxes
        void ClearAll()
        {
            foreach (Control ctrl in pnlBoxes.Controls)
            {
                if (ctrl is TextBox) ((TextBox)ctrl).Clear();
            }
            foreach (CheckBox chkBox in grpAlarmtypes.Controls)
            {
                chkBox.Checked = false;
            }
        }

    }
}
```

AdministrateSubscribers.cs (form)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using SQLCommunication;

namespace Prosjekt
{
    public partial class AdministrateSubscribers : Form
    {
        Subscriber sub;

        public AdministrateSubscribers()
        {
            InitializeComponent();

            UpdateTable();
        }
```

```
        private void dgvSubscribers_CellClick(object sender, DataGridViewCellEventArgs e)
        {
            if (e.RowIndex >= 0)
            {
                ClearAll();
                DataGridViewRow row = this.dgvSubscribers.Rows[e.RowIndex];

                // Makes new subscriber object based on info in row
                sub = new Subscriber(row.Cells[0].
                    Value.ToString(), row.Cells[1].Value.ToString(),
                    row.Cells[2].Value.ToString(), Convert.ToInt32(row.Cells[3].Value));

                // Displays in textboxes
                txtEMail.Text = sub.Email;
                txtFirstName.Text = sub.FirstName;
                txtLastName.Text = sub.LastName;
                txtPhoneNumber.Text = sub.Telephone.ToString();

                // Gets list with alarmtypes
                List<string> alarmList = SQL.GetSubscriberAlarmList(sub);

                // Add chk at beginning of each string so checkboxes could be checked
                for (int i = 0; i < alarmList.Count; i++)
                {
                    alarmList[i] = "chk" + alarmList[i];
                    alarmList[i] = alarmList[i].Replace(" ", "");
                }
                foreach (CheckBox chk in grpAlarmtypes.Controls)
                {
                    if (alarmList.Contains(chk.Name)) chk.Checked = true;
                }
            }
        }

        // When datasource in datagridview is successfully loaded, clear selection, so no row is
active
        private void dgvSubscribers_DataBindingComplete(object sender,
DataGridViewBindingCompleteEventArgs e)
        {
            dgvSubscribers.ClearSelection();
        }

        private void btnUpdateSubscriber_Click(object sender, EventArgs e)
        {
            // Update subsriber in database
            SQL.UpdateSubscriber(sub, txtEMail.Text, txtFirstName.Text,
              txtLastName.Text, Convert.ToInt32(txtPhoneNumber.Text));

            // Update alarmtypes for subscriber
            foreach (CheckBox chk in grpAlarmtypes.Controls)
            {
              // Removes chk from checkbox-name, and casts string to AlarmtypesEnum. Then update
              alarmtype for subscriber
                string temp = chk.Name;
                temp = temp.Replace("chk", "");
                AlarmtypesEnum alarm = (AlarmtypesEnum)Enum.Parse(typeof(AlarmtypesEnum), temp);
                if (chk.Checked)
                    sub.SubscribeTo(alarm);
                if (!chk.Checked)
                    sub.UnsubscribeTo(alarm);
            }
```

```
            MessageBox.Show("Abonnent " + sub.Email + " oppdatert", "Info",
MessageBoxButtons.OK, MessageBoxIcon.Information);
            UpdateTable();
        }

        private void btnDeleteSubsriber_Click(object sender, EventArgs e)
        {
            // Delete from database only if user answer yes in messagebox
            DialogResult dialogResult = MessageBox.Show("Er du sikker på at du vil slette " +
sub.Email + " fra databasen?", "Advarsel", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
            if (dialogResult == DialogResult.Yes)
            {
                SQL.DeleteSubscriber(sub);
                UpdateTable();
                ClearAll();
            }
        }

        void UpdateTable()
        {
            string sqlQuery = @"SELECT Email as 'E-post', FirstName as Fornavn, LastName as
Etternavn, Telephone as 'Tlf nummer'  From SUBSCRIBER";
            dgvSubscribers.DataSource = SQL.MakeDataGridView(sqlQuery);
        }

        // Clears all text- and checkboxes
        void ClearAll()
        {
            foreach (Control ctrl in pnlBoxes.Controls)
            {
                if (ctrl is TextBox) ((TextBox)ctrl).Clear();
            }
            foreach (CheckBox chkBox in grpAlarmtypes.Controls)
            {
                chkBox.Checked = false;
            }
        }
    }
}
```

Alarmlist.cs (form)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using SQLCommunication;
using System.IO;

namespace Prosjekt
{
    public partial class AlarmList : Form
    {
        string sqlQuery;
```

```
        public AlarmList()
        {
            InitializeComponent();

            UpdateAlarmListAll();
        }

        // Update alarmlist table with all alarms
        void UpdateAlarmListAll()
        {
            sqlQuery = @"SELECT Alarm as Alarmtype, Time as Tidspunkt FROM ALARM ORDER BY Time
DESC";
            dgvAlarms.DataSource = SQL.MakeDataGridView(sqlQuery);
        }

        // Update alarmlist table based on date
        void UpdateAlarmListBetweenDates()
        {
            string dateFrom = string.Format("{0}.{1}.{2}",
                    dateTimeFrom.Value.Year, dateTimeFrom.Value.Month, dateTimeFrom.Value.Day);
            string dateTo = string.Format("{0}.{1}.{2}", dateTimeTo.Value.Year,
                    dateTimeTo.Value.Month, dateTimeTo.Value.Day+1);
            sqlQuery = string.Format("SELECT Alarm as Alarmtype, Time as Tidspunkt FROM ALARM
              WHERE Time >= '{0}' and Time < '{1}' ORDER BY Time DESC", dateFrom, dateTo);
            dgvAlarms.DataSource = SQL.MakeDataGridView(sqlQuery);
        }

        void CreatePdf()
        {
            if (rdoShowAllAlarms.Checked) PdfForm.CreatePdf("Viser alle alarmer i databasen",
              SQL.MakeDataGridView(sqlQuery));
            else
            {
                string header = string.Format("Viser alarmer fra {0}, til og med {1}",
                     dateTimeFrom.Text, dateTimeTo.Text);
                PdfForm.CreatePdf(header, SQL.MakeDataGridView(sqlQuery));
            }
        }

        private void btnExportToPdf_Click(object sender, EventArgs e)
        {
            CreatePdf();
        }

        // Open sendmail form
        private void btnSendEmail_Click(object sender, EventArgs e)
        {
            CreatePdf();
            SendEmail frmSendEmail = new SendEmail();
            frmSendEmail.ShowDialog(this);
        }

        #region Disable/enable controls
        private void dateTimeFrom_ValueChanged(object sender, EventArgs e)
        {
            UpdateAlarmListBetweenDates();
        }

        private void dateTimeTo_ValueChanged(object sender, EventArgs e)
        {
            UpdateAlarmListBetweenDates();
        }
```

```
        private void rdoShowFromTo_CheckedChanged(object sender, EventArgs e)
        {
            dateTimeFrom.Enabled = true;
            dateTimeTo.Enabled = true;
            UpdateAlarmListBetweenDates();
        }

        private void rdoShowAllAlarms_CheckedChanged(object sender, EventArgs e)
        {
            dateTimeFrom.Enabled = false;
            dateTimeTo.Enabled = false;
            UpdateAlarmListAll();
        }
        #endregion
    }
}
```

Arduino.cs (class)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Prosjekt
{
    class Arduino
    {
        #region
        private List<string> ArduinoValuelist = new List<string>(); //local list
        private string data;
        private string rawData;
        private double temp;
        private bool pirEnabled;
        public bool Connected { get; protected set; }
        #endregion

        public Arduino(string data)
        {
            if (data != null)
            {
                Connected = true;
                rawData = data;
                SplittTekst();
                CheckPirValue();
                CheckTemperatureValue();
            }
            else Connected = false;
        }

        /// <summary>
        /// Finish: 22.03.2016
        /// Trim data string.
        /// Store all value in each different row in list.
        /// </summary>
        /// <param name="value"></param>
        private void SplittTekst()
```

```
{
    data = rawData;
    data = data.Replace("\t", "");
    data = data.Replace("\r", "");
    data = data.Replace('.', ',');
    ArduinoValuelist = data.Split(' ').ToList();
}

/// <summary>
/// Finish: 25.03.2016
/// Determine if Pir state is On or Off
/// Return true or false to PirEnabled
/// </summary>
/// <returns></returns>
private void CheckPirValue()
{
    pirEnabled = false;
    foreach (var c in ArduinoValuelist)
    {
        if (c.StartsWith("#P:"))
        {
                string pir = c;

                if (pir.Contains("ON")) pirEnabled = true;
                else pirEnabled = false;
        }
    }
}

/// <summary>
/// Finish: 25.03.2016
/// Split temp from ArduinoValueList to Temperature property
/// </summary>
private void CheckTemperatureValue()
{
    foreach (string val in ArduinoValuelist)
    {
        if (val.StartsWith("#T:"))
        {
            string temporary = val;
            temporary = temporary.Remove(0, 3);
            temp = Convert.ToDouble(temporary);
        }
    }
}

/// <summary>
/// Updates properties if arduino is connected
/// </summary>
/// <param name="data">If string is empty, arduino is not connected</param>
public void Update(string data)
{
    if (data != null)
    {
        Connected = true;
        rawData = data;
        SplittTekst();
        CheckPirValue();
        CheckTemperatureValue();
    }
    else Connected = false;
}
```

```
        public double Temperature
        {
            get { return temp; }
        }

        public bool PirEnabled
        {
            get { return pirEnabled; }
        }
    }
}
```

ArduinoCom.cs (class)

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Security.AccessControl;
using System.Text;
using System.Threading.Tasks;
using  System.IO.Ports;
using System.Runtime.InteropServices;
using System.Windows.Forms;

namespace Prosjekt
{
    public enum Command { ReadStatus = 1, Alarm = 2 }

    public class ArduinoCom
    {
        SerialPort Port = new SerialPort();
        public event EventHandler NoArduinoConnection;

        private int connectAttempts = 0;

        private string ComName;
        private int BaudRate;
        public string Commando = "";

        #region Constructor

        public ArduinoCom(string Com, int baud)
        {
            ComName = Com;
            BaudRate = baud;
        }
        public ArduinoCom()
        {
            AutoSetPortName();
        }
        #endregion

        #region Set and get Comport name and BaudRate

        /// <summary>
        /// Finish: 19.03.2016
        /// Get Com name and Baud rate
        /// And set them to Serialport
```

```
        /// </summary>
        /// <param name="Com"></param>
        /// <param name="Baud"></param>
        public void ManualSetPortAndBaudValue(string Com, int Baud)
        {
            ComName = Com;
            BaudRate = Baud;
            SetPortAndValue();
        }
        public void ManuelSetBaud(int baud)                                         // Set the
communication speed
        {
            BaudRate = baud;
        }

        /// <summary>
        /// Finish: 19.03.2016
        /// Find arduino port name.
        /// Store the port name to ComName.
        /// </summary>
        public void AutoSetPortName()
        {
            foreach (string portName in SerialPort.GetPortNames())
            {
                Port = new SerialPort(portName);
                if (Port.IsOpen == false) ComName = portName;
            }

        }

        public void SetPortAndValue()
        {
            try
            {
                Port.PortName = ComName;
            }
            catch (Exception)
            {
                MessageBox.Show("No Connection to Arduino");
            }

            Port.BaudRate = BaudRate;
        }                                       // Sett the Port And Baud
        #endregion

        #region These code are use for open and send command to arduino

        /// <summary>
        /// Finish: 22.03.2016
        /// Send command code to request data value from arduino
        /// Check if port is open, and check if arduino is connected
        /// Rais a event to say there is not connection
        /// </summary>
        /// <param name="command"></param>
        /// <returns></returns>
        public string SendCommandToArduino(Command command)
        {
            string ArduinoValue = null;
            if (Port.IsOpen)
            {
                try
                {
```

```
            if (command == Command.ReadStatus)
            {
                Port.Write("1");
                ArduinoValue = Port.ReadLine();
            }
            else if (command == Command.Alarm)
            {
                Port.Write("2");
                ArduinoValue = Port.ReadLine();
            }
        }
        catch (Exception)
        {
            MessageBox.Show("Port is Busy right now");
        }
    }
    else if (connectAttempts > 5)
    {
        AutoSetPortName();
        connectAttempts = 0;
    }
    else
    {
        try
        {
            Port.Open();
            if (command == Command.ReadStatus)
            {
                Port.Write("1");
                ArduinoValue = Port.ReadLine();
            }
            else if (command == Command.Alarm)
            {
                Port.Write("2");
                ArduinoValue = Port.ReadLine();
            }
        }
        catch (Exception)
        {
            connectAttempts++;
            if (NoArduinoConnection != null) NoArduinoConnection(this, EventArgs.Empty);
        }
    }
    return ArduinoValue;
}

/// <summary>
/// Finish: 22.03.2016
/// Only use for testing
/// Send command line "7" to stop/reset arduino
/// </summary>
/// <param name="command"></param>
public void SendOnlySingleCommand(string command)
{
    if (Port.IsOpen) Port.Write(command);
    else
    {
        try
        {
            Port.Open();
            Port.Write(command);
        }
```

```
                catch (Exception)
                {
                    MessageBox.Show("Port is Busy right now");
                }
            }
        }
        public void SendOnlySingleCommand()
        {
            if (Port.IsOpen) Port.Write(Commando);
            else
            {
                Port.Open();
                Port.Write(Commando);
            }
        }
        public void ClosePort()
        {
            if (Port.IsOpen) Port.Close();
            else Port.Close();
        }
        #endregion

    }
}
```

HistoricGraph.cs (form)

```
using SQLCommunication;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;

namespace Prosjekt
{
    public partial class HistoricGraph : Form
    {
        public HistoricGraph()
        {
            InitializeComponent();
        }


        void UpdateGraph()
        {
            // Clear all points in graph
            graph.Series[0].Points.Clear();

            // Gets datatable with logged temperatures from database
            string sqlQuery;
            string dateFrom = string.Format("{0}.{1}.{2}", dateTimeFrom.Value.Year,
                      dateTimeFrom.Value.Month, dateTimeFrom.Value.Day);
            string dateTo = string.Format("{0}.{1}.{2}", dateTimeTo.Value.Year,
                     dateTimeTo.Value.Month, dateTimeTo.Value.Day + 1);
```

```
            sqlQuery = string.Format("SELECT Time as Tidspunkt,
                    Temperature as Temperatur FROM Temperature WHERE Time >= '{0}' and
                    Time < '{1}' ORDER BY Time", dateFrom, dateTo);
            DataTable table = SQL.MakeDataGridView(sqlQuery);

            // Adds each row to graph
            for (int i = 0; i < table.Rows.Count; i++)
            {
                graph.Series[0].Points.AddXY(table.Rows[i][0], table.Rows[i][1]);
            }
        }

        private void dateTimeFrom_ValueChanged(object sender, EventArgs e)
        {
            UpdateGraph();
        }

        private void dateTimeTo_ValueChanged(object sender, EventArgs e)
        {
            UpdateGraph();
        }
    }
}
```

Mail.cs (class)

```
using System.Net;
using System.Net.Mail;
using System.Windows.Forms;

namespace Prosjekt
{
    public class Mail
    {
        // Sets up standard e-mail properties, like e-mail server(smtpAddress), portnumber and
SSL (encrypted connection)
        private string smtpAddress = "smtp.live.com"; //smtp.live.com for hotmail.
smtp.gmail.com for gmail. smtp.mail.yahoo.com for Yahoo!
        private int portNumber = 587;
        private bool enableSSL = true;
        private string emailFrom = "alarmgruppe3@hotmail.com";
        private string password = "hitalarm123";
        private string attachmentFilename = null;
        public bool EmailSent { get; private set; }
        MailMessage mail;

        // Constructor wihtout filename
        public Mail (string emailTo, string subject, string body)
        {
            mail = new MailMessage();
            EmailSent = false;
            mail.From = new MailAddress(emailFrom);
            mail.To.Add(emailTo);
            mail.Subject = subject;
            mail.Body = body;
            mail.IsBodyHtml = false;
        }

        // Constructor with filename
        public Mail (string emailTo, string subject, string body, string attachmentFilename)
```

```
        {
            mail = new MailMessage();
            EmailSent = false;
            mail.From = new MailAddress(emailFrom);
            mail.To.Add(emailTo);
            mail.Subject = subject;
            mail.Body = body;
            mail.IsBodyHtml = false;
            this.attachmentFilename = attachmentFilename;
        }

        /// <summary>
        /// Send email, and set EmailSent to true if email is sent
        /// </summary>
        public void Send()
        {
            if (attachmentFilename != null) mail.Attachments.Add(new
Attachment(attachmentFilename));

            SmtpClient smtp = new SmtpClient(smtpAddress, portNumber);
            smtp.Credentials = new NetworkCredential(emailFrom, password);
            smtp.EnableSsl = enableSSL;
            smtp.Send(mail);
            EmailSent = true;
        }
    }
}
```

Pdf.cs (class)

```
using System.Data;
using System.IO;
using System.Diagnostics;
using MigraDoc.DocumentObjectModel;
using MigraDoc.DocumentObjectModel.Tables;
using MigraDoc.Rendering;

namespace Prosjekt
{

    public class PdfForm
    {
        static Document document;
        static DataTable dataTable;

        static string headerText;
        static string filename = "Alarmlist.pdf";
        static string completeFilename;

        public static void CreatePdf(string headerString, DataTable table)
        {
            // Store variables, and create migradoc-document
            headerText = headerString;
            dataTable = table;
            document = CreateDocument();

            // Renders the migradoc-document as a pdf document
            PdfDocumentRenderer renderer = new PdfDocumentRenderer
                    (true, PdfSharp.Pdf.PdfFontEmbedding.Always);
            renderer.Document = document;
```

```
    renderer.RenderDocument();

    // If Pdf directory don't exist, create it
    string directory = @"Pdf\";
    if (!Directory.Exists(directory))
    {
        Directory.CreateDirectory(directory);
    }

    completeFilename = directory + filename;

    // Save the pdf-document, and open it
    renderer.PdfDocument.Save(completeFilename);
    Process.Start(completeFilename);
}

private static Document CreateDocument()
{
    // Creates new Migradoc-document and defines some Info
    Document document = new Document();
    document.Info.Author = "IA-Gruppe 3";
    document.Info.Title = "How to create PDF's";

    //Creates header of the document
    CreateHeader(document);

    //Creates the table after the header
    CreateTable(document, dataTable);

    return document;
}

private static void CreateHeader(Document document)
{
    // Adds new section to the migradoc-document
    Section section = document.AddSection();

    // Adds the top headers to the document some formatting
    Paragraph paragraph = section.AddParagraph("Alarmliste");
    paragraph.Format.Font.Size = 22;
    paragraph = section.AddParagraph(headerText);
    paragraph.Format.Font.Size = 14;
    paragraph.Format.SpaceAfter = 5;
}

private static void CreateTable(Document document, DataTable  dataTable)
{
    // Adds new table to migradoc-document with some formatting
    Table table = new Table();
    table.Borders.Width = 0.75;
    table.Format.Font.Size = 14;
    table.Rows.Height = 20;

    // Adds two columns
    Column column = table.AddColumn(Unit.FromCentimeter(5));
    column.Format.Alignment = ParagraphAlignment.Left;
    table.AddColumn(Unit.FromCentimeter(7));

    // Add the heading-row with fixed headings and blue color
    Row row = table.AddRow();
    row.Shading.Color = Colors.LightSkyBlue;
    Cell cell = row.Cells[0];
```

```
            cell.AddParagraph("Alarmtype");
            cell = row.Cells[1];
            cell.AddParagraph("Tidspunkt");

            // For every row in datatable, add new row to migradoc-document
            for (int i = 0; i < dataTable.Rows.Count; i++)
            {
                // Read datetime from datatable, and add a new row to the table
                string datetime = dataTable.Rows[i][1].ToString();
                row = table.AddRow();

                // Adds a light blue color for every other line in table for readability
                if (!IsEven(i)) row.Shading.Color = Colors.AliceBlue;

                // Add text from datatable to the two new cells.
                    Remove seconds from datetimestring
                cell = row.Cells[0];
                cell.AddParagraph(dataTable.Rows[i][0].ToString());
                cell = row.Cells[1];
                cell.AddParagraph(datetime.Remove(16));
            }

            // Adds the newly created table in the bottom of the migradoc-document
            document.LastSection.Add(table);
        }

        private static bool IsEven(int value)
        {
            return value % 2 == 0;
        }

        public static string CompleteFilename
        {
            get { return completeFilename; }
        }
    }

}
```

ProgramParameter.cs (class)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.ComponentModel;
using System.Windows.Forms;

namespace Prosjekt
{
    // Class to save and read log- and readinterval.
    // File saved as cfg file in the same folder as program.exe
    public class ProgramParameter : INotifyPropertyChanged
    {
        private string filename = "config.cfg";
        private int[] intervals = new int[2];

        public event PropertyChangedEventHandler PropertyChanged;
```

```
// Constructor
public ProgramParameter()
{
    ReadFromFile();
    if (ReadInterval == 0) ReadInterval = 5;
    if (LogInterval == 0) LogInterval = 5;
}

// Alternative constructor that is not used
public ProgramParameter(int readInterval, int logInterval, string path)
{
    filename = path;
    ReadInterval = readInterval;
    LogInterval = logInterval;
    SaveToFile();
}

// Readinterval property, save to file when set
public int ReadInterval
{
    get { return intervals[0]; }
    set
    {
        if (value >= 2 && value <= 10)
        {
            intervals[0] = value;
            OnPropertyChanged("ReadInterval");
            SaveToFile();
        }
        else throw new ArgumentOutOfRangeException
            ("ReadInterval må være mellom 2 og 10 sekunder");
    }
}

// Loginterval property, save to file when set
public int LogInterval
{
    get { return intervals[1]; }
    set
    {
        if (value >= 1 && value <= 60)
        {
            intervals[1] = value;
            OnPropertyChanged("LogInterval");
            SaveToFile();
        }
        else throw new ArgumentOutOfRangeException
            ("LogInterval må være mellom 1 og 60 min");
    }
}

/// <summary>
/// Saves log- and readinterval to file
/// </summary>
public void SaveToFile()
{
    FileStream fs = new FileStream(filename, FileMode.OpenOrCreate,
            FileAccess.ReadWrite);
    StreamWriter sw = new StreamWriter(fs);
    foreach (int i in intervals)
    {
```

```
                sw.WriteLine(i);
            }
            sw.Close();
        }

        /// <summary>
        /// Reads log- and readinterval from file
        /// </summary>
        /// <returns></returns>
        public int[] ReadFromFile()
        {
            FileStream fs = new FileStream(filename, FileMode.OpenOrCreate, FileAccess.Read);
            StreamReader sr = new StreamReader(fs);

            for (int i = 0; i <= intervals.GetUpperBound(0); i++)
            {
                intervals[i] = Convert.ToInt32(sr.ReadLine());
            }

            sr.Close();
            return intervals;
        }

        // Propery changed event inherited from interface
        protected virtual void OnPropertyChanged(string propName)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(propName));
            }
        }


    }
}
```

SendMail.cs (form)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Prosjekt
{
    public partial class SendEmail : Form
    {
        string emailAdress;
        List<string> fullEmailList;

        public SendEmail()
        {
            InitializeComponent();

            fullEmailList = SQLCommunication.SQL.GetFullEmailList();
```

```
            cboEmail.Items.Clear();

            foreach (string email in fullEmailList)
            {
                cboEmail.Items.Add(email);
            }
        }

        private void rdoOtherUser_CheckedChanged(object sender, EventArgs e)
        {
            txtEmail.Enabled = true;
            cboEmail.Enabled = false;
            btnSendEmail.Enabled = false;
        }

        private void cboEmail_SelectedIndexChanged(object sender, EventArgs e)
        {
            if (cboEmail.Text != null) btnSendEmail.Enabled = true;
            else btnSendEmail.Enabled = false;

        }

        private void rdoUserInDatabase_CheckedChanged(object sender, EventArgs e)
        {
            cboEmail.Enabled = true;
            txtEmail.Enabled = false;
        }

        private void txtEmail_TextChanged(object sender, EventArgs e)
        {
            if (txtEmail.Text != null) btnSendEmail.Enabled = true;
            else btnSendEmail.Enabled = false;
        }

        private void btnSendEmail_Click(object sender, EventArgs e)
        {
            if (rdoUserInDatabase.Checked) emailAdress = cboEmail.Text;
            else emailAdress = txtEmail.Text;
            Mail mail = new Mail(emailAdress, "Alarmliste på pdf",
                    "Vedlagt ligger alarmliste på pdf", PdfForm.CompleteFilename);
            Task sendMail = Task.Factory.StartNew(() => mail.Send());
            Task.WaitAll(sendMail);
            if (mail.EmailSent)
            {
                MessageBox.Show("E-post sent!");
                this.Close();
            }
        }
    }
}
```

ServerConnect.cs (form)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
using System.Windows.Forms;
using SQLCommunication;

namespace Prosjekt
{
    public partial class ServerConnect : Form
    {
        public ServerConnect()
        {
            InitializeComponent();
        }

        // Checks connection, and if it is successfull, open Home form
        private void btnConnect_Click(object sender, EventArgs e)
        {
            SQLCom.StartCom(txtServerName.Text);
            if (SQLCom.SuccessfullLogin == true)
            {
                SQL.ConnectionString = SQLCom.ConnectionString;
                this.Close();
            }
            else if (SQLCom.SuccessfullLogin == false)
            {
                MessageBox.Show("Feil ved innlogging til database. Har du skrivet rett
                     servernavn?", "Feil", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            else
            {
                MessageBox.Show("Det oppstod en uventet feil ved innlogging til database.
                     Vennligst kontakt programutvikler. Programmet vil nå avsluttes",
                    "Feil", MessageBoxButtons.OK, MessageBoxIcon.Error);
                Application.Exit();
            }
        }

        // Quit application if cancel is clicked
        private void btnCancel_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void ServerConnect_FormClosed(object sender, FormClosedEventArgs e)
        {
            if (SQLCom.SuccessfullLogin == false)
            {
            Application.Exit();
            }
        }
    }
}
```

Settings.cs (form)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using SQLCommunication;

namespace Prosjekt
{
    public partial class Settings : Form
    {
        //ProgramParameter programParameter = new ProgramParameter();


        public Settings()
        {
            InitializeComponent();

            lowTempLimitUpDown.Value =
                    Convert.ToDecimal(SQL.GetAlarmLimit(AlarmtypesEnum.LowTemp));
            highTempLimitUpDown.Value =
                    Convert.ToDecimal(SQL.GetAlarmLimit(AlarmtypesEnum.HighTemp));
            lowBatteryLimitUpDown.Value =
                    Convert.ToDecimal(SQL.GetAlarmLimit(AlarmtypesEnum.LowBattery));

            readIntervalUpDown.Value = Home.ProgramParameter.ReadInterval;
            logIntervalUpDown.Value = Home.ProgramParameter.LogInterval;

        }

        private void lowTempLimitUpDown_ValueChanged(object sender, EventArgs e)
        {
            SQL.UpdateAlarmLimit(AlarmtypesEnum.LowTemp,
                    Convert.ToDouble(lowTempLimitUpDown.Value));
        }

        private void highTempLimitUpDown_ValueChanged(object sender, EventArgs e)
        {
            SQL.UpdateAlarmLimit(AlarmtypesEnum.HighTemp,
                     Convert.ToDouble(highTempLimitUpDown.Value));
        }

        private void lowBatteryLimitUpDown_ValueChanged(object sender, EventArgs e)
        {
            SQL.UpdateAlarmLimit(AlarmtypesEnum.LowBattery,
                    Convert.ToDouble(lowBatteryLimitUpDown.Value));
        }

        private void cboReadInterval_SelectedIndexChanged(object sender, EventArgs e)
        {
            int interval;
            //bool resultIntervalConvert = int.TryParse(cboReadInterval.Text, out interval);
            //if (resultIntervalConvert) ProgramParameter. = interval * 1000;
            //else MessageBox.Show("Ugyldig verdi");
        }


        private void readIntervalUpDown_ValueChanged(object sender, EventArgs e)
        {
            Home.ProgramParameter.ReadInterval = (int)(readIntervalUpDown.Value);
        }

        private void logIntervalUpDown_ValueChanged(object sender, EventArgs e)
        {
```

```
            Home.ProgramParameter.LogInterval = (int)(logIntervalUpDown.Value);
        }
    }
}
```

Sql.cs (class)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;
using System.Data;

namespace SQLCommunication
{
    //This enum is used to easily get the syntax of the alarmtype right.
    public enum AlarmtypesEnum { NoCharge, HighTemp, LowTemp, LowBattery, ComFault, Motion }

    //Class SQLCom takes care of every interaction between the C#-program and the SQLDatabase.
    //All methods are static, which means they can be invoked without having to create
    //an instance of SQLCom.
    public class SQL
    {

        #region Fields
        private static string conString = @"Data Source=SIMEN-PC;Initial Catalog = Alarmsystem;
                                Integrated Security = True";
        private static SqlConnection con;
        private static SqlCommand command;
        #endregion

        #region Methods
        //Adds a new subcriber in the database
        public static void AddSubscriber(Subscriber sub)
        {
            if (SubscriberExists(sub) == false)
            {
                try
                {
                    con.Open(); //Opening the connection to the database
                    command.CommandText = "NewSubscriber"; //Calling the
                            stored procedure to add a new subscriber
                    command.Parameters.Clear(); //Clears the parameterlist
                    command.Parameters.Add(new SqlParameter("@Email", sub.Email));
                           //Further adding the parameters necessary for the procedure
                    command.Parameters.Add(new SqlParameter("@FirstName", sub.FirstName));
                    command.Parameters.Add(new SqlParameter("@LastName", sub.LastName));
                    command.Parameters.Add(new SqlParameter("@Telephone", sub.Telephone));
                    command.ExecuteNonQuery(); //Executing
                }
                catch (Exception error) //if error occurs do:
                {
                    Console.WriteLine(error.Message);
                }
                finally
                {
                    con.Close(); //Closes the connection to database at either successful or
unsuccessful method execution
```

```
        }
    }
}

//Updates a subscriber specified by the oldEmail parameter
public static void UpdateSubscriber(Subscriber sub, string newEmail,
       string newFirstName, string newLastName, int newPhoneNumber)
{
    if (SubscriberExists(sub))
    {
        try
        {
            con.Open();
            command.CommandText = "UpdateSubscriber";
            command.Parameters.Clear();
            command.Parameters.Add(new SqlParameter("@Email", sub.Email));
            command.Parameters.Add(new SqlParameter("@NewEmail", newEmail));
            command.Parameters.Add(new SqlParameter("@FirstName", newFirstName));
            command.Parameters.Add(new SqlParameter("@LastName", newLastName));
            command.Parameters.Add(new SqlParameter("@Telephone", newPhoneNumber));
            command.ExecuteNonQuery();
            sub.Email = newEmail;
            sub.FirstName = newFirstName;
            sub.LastName = newLastName;
            sub.Telephone = newPhoneNumber;
        }
        catch (Exception error)
        {
            Console.WriteLine(error.Message);
        }
        finally
        {
            con.Close();
        }
    }
}

//LogTemp inserts a double value in the temperature history
public static void LogTemp(double temp)
{
    try
    {
        con.Open();
        command.CommandText = "MeasureTemp";
        command.Parameters.Clear();
        command.Parameters.Add(new SqlParameter("@Temperature", temp));
        command.ExecuteNonQuery();
    }
    catch (Exception error)
    {
        Console.WriteLine(error.Message);
    }
    finally
    {
        con.Close();
    }
}

//NewAlarm inserts a alarm in the alarm history
public static void NewAlarm(AlarmtypesEnum alarmtype)
{
    try
```

```
    {
        con.Open();
        command.CommandText = "NewAlarm";
        command.Parameters.Clear();
        command.Parameters.Add(new SqlParameter("@Alarm", alarmtype.ToString()));
        command.ExecuteNonQuery();
        con.Close();
    }
    catch (Exception error)
    {
        Console.WriteLine(error.Message);
    }
    finally
    {
        con.Close();
    }
}

//UpdateAlarmLimit changes the alarm limits in the database
public static void UpdateAlarmLimit(AlarmtypesEnum alarmtype, double newLimit)
{
    try
    {

        con.Open();
        command.CommandText = "UpdateLimit";
        command.Parameters.Clear();
        command.Parameters.Add(new SqlParameter("@AlarmType", alarmtype.ToString()));
        command.Parameters.Add(new SqlParameter("@NewLimit", newLimit));
        command.ExecuteNonQuery();
        con.Close();
    }
    catch (Exception error)
    {
        Console.WriteLine(error.Message);
    }
    finally
    {
        con.Close();
    }
}

//GetLimit returns the alarm limit for a specific alarmtype as a double
public static double GetAlarmLimit(AlarmtypesEnum alarmtype)
{
    try
    {
        con.Open();
        command.CommandText = "GetLimit";
        command.Parameters.Clear();
        command.Parameters.Add(new SqlParameter("@AlarmType", alarmtype.ToString()));
        double result = ((double)command.ExecuteScalar());
        con.Close();
        return result;
    }
    catch (Exception)
    {
        //throw new NotImplementedException();
        return 0;
    }
    finally
    {
```

```
                con.Close();
            }
        }

        //GetDescription returns the description for a specified alarmtype as a string
        public static string GetDescription(AlarmtypesEnum alarmtype)
        {
            try
            {
                con.Open();
                command.CommandText = "GetDescription";
                command.Parameters.Clear();
                command.Parameters.Add(new SqlParameter("@AlarmType", alarmtype.ToString()));
                string result = ((string)command.ExecuteScalar());
                return result;
            }
            catch (Exception error)
            {
                return "Failed to get description: \n" + error.Message;
            }
            finally
            {
                con.Close();
            }

        }

        //MakeDataGridView returns DataTable which can be used to fill a DataGridView.
        //Datatable contains data specified in the sqlQueryToFillTable, which means you have to
pass
        //a valid sql query to invoke the method without errors
        public static DataTable MakeDataGridView(string sqlQueryToFillTable)
        {
            try
            {
                con.Open();
                SqlDataAdapter sda = new SqlDataAdapter(sqlQueryToFillTable, con);
                DataTable dt = new DataTable();
                sda.Fill(dt);
                return dt;
            }
            catch (Exception error)
            {
                Console.WriteLine(error.Message);
                return null;
            }
            finally
            {
                con.Close();
            }
        }

        //Removes a subscriber from the database
        public static void DeleteSubscriber(Subscriber sub)
        {
            try
            {
                con.Open();
                command.CommandText = "DeleteSubscriber";
                command.Parameters.Clear();
                command.Parameters.AddWithValue("@Email", sub.Email);
                command.ExecuteScalar();
```

```
            }
            catch (Exception error)
            {
                Console.WriteLine(error.Message);
            }
            finally
            {
                con.Close();
            }
        }

        //This method adds a subscriber to an alarm, in other words, the specified subscriber
will
        //get notifications when this type of alarms is raised
        public static void SubscribesTo(Subscriber sub, AlarmtypesEnum alarmtype)
        {
            try
            {
                con.Open();
                command.CommandText = "NewSubscribesTo";
                command.Parameters.Clear();
                command.Parameters.AddWithValue("@Email", sub.Email);
                command.Parameters.AddWithValue("@Alarmtype", alarmtype.ToString());
                command.ExecuteNonQuery();
            }
            catch (Exception error)
            {
                Console.WriteLine(error.Message);
            }
            finally
            {
                con.Close();
            }
        }

        //Unsubscribes a subscriber from email notifications for a given alarmtype
        public static void UnsubscribeTo(Subscriber sub, AlarmtypesEnum alarmtype)
        {
            try
            {
                con.Open();
                command.CommandText = "UnSubscribesTo";
                command.Parameters.Clear();
                command.Parameters.AddWithValue("@Email", sub.Email);
                command.Parameters.AddWithValue("@Alarmtype", alarmtype.ToString());
                command.ExecuteNonQuery();
            }
            catch (Exception error)
            {
                Console.WriteLine(error.Message);
            }
            finally
            {
                con.Close();
            }
        }

        //Checks if a subscriber already exists in the database
        public static bool SubscriberExists(Subscriber sub)
        {
            try
            {
```

```
                con.Open();
                command.CommandType = CommandType.Text;
                command.CommandText = "SELECT EMAIL FROM SUBSCRIBER WHERE EMAIL = '" + sub.Email
+ "'";
                object test = command.ExecuteScalar();
                if (test != null)
                {
                    //MessageBox.Show("Bruker er allerede lagt til i databasen!");
                    //Console.WriteLine("Bruker er allerede lagt til i databasen!");
                    return true;
                }
                else return false;
            }
            catch (Exception error)
            {
                Console.WriteLine(error.Message);
                return false;
            }
            finally
            {
                con.Close();
                command.CommandType = CommandType.StoredProcedure;
            }
        }

        //Gets a list of the alarms the subscribers subscribe to
        public static List<string> GetSubscriberAlarmList(Subscriber sub)
        {
            List<string> AlarmList = new List<string>();
            SqlDataReader rdr = null;
            try
            {
                con.Open();
                command.CommandType = CommandType.Text;
                command.CommandText = @"SELECT Alarm FROM SUBSCRIBESTO
                    WHERE Email = '" + sub.Email + "'";
                rdr = command.ExecuteReader();
                while (rdr.Read())
                {
                    string alarm = rdr["Alarm"].ToString();
                    AlarmList.Add(alarm);
                }
                return AlarmList;

            }
            catch (Exception error)
            {
                Console.WriteLine(error.Message);
                return null;
            }
            finally
            {
                con.Close();
                command.CommandType = CommandType.StoredProcedure;
            }
        }

        public static List<string> GetEmailList(AlarmtypesEnum alarmtype)
        {
            List<string> EmailList = new List<string>();
            SqlDataReader rdr = null;
            try
```

```
    {
        con.Open();
        command.CommandType = CommandType.Text;
        command.CommandText = @"SELECT Email FROM SUBSCRIBESTO
            WHERE Alarm = '" + alarmtype + "'";
        rdr = command.ExecuteReader();
        while (rdr.Read())
        {
            string Email = rdr["Email"].ToString();
            EmailList.Add(Email);
        }
        return EmailList;

    }
    catch (Exception error)
    {
        Console.WriteLine(error.Message);
        return null;
    }
    finally
    {
        con.Close();
        command.CommandType = CommandType.StoredProcedure;
    }
}

public static List<string> GetFullEmailList()
{
    List<string> EmailList = new List<string>();
    SqlDataReader rdr = null;
    try
    {
        con.Open();
        command.CommandType = CommandType.Text;
        command.CommandText = @"SELECT Email FROM SUBSCRIBER";
        rdr = command.ExecuteReader();
        while (rdr.Read())
        {
            string email = rdr["Email"].ToString();
            EmailList.Add(email);
        }
        return EmailList;

    }
    catch (Exception error)
    {
        Console.WriteLine(error.Message);
        return null;
    }
    finally
    {
        con.Close();
        command.CommandType = CommandType.StoredProcedure;
    }
}

#endregion

public static string ConnectionString
{
    get { return conString; }
    set
```

```
            {
                conString = value;
                con = new SqlConnection(conString); //initializes a new SQLConnection
                command = new SqlCommand()
                { Connection = con, CommandType = CommandType.StoredProcedure };
                     //initializes a new SQLCommand of type "Stored Procedure"
            }
        }
    }
}
```

SqlCom.cs (class)

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;
using SQLCommunication;
using System.Diagnostics;
using System.IO;
using System.Windows.Forms;

namespace SQLCommunication
{
    class SQLCom
    {
        private static string conString;
        private static string databaseName = "Alarmsystem";
        private static string dataSource;
        private static string databaseSuccessfullyCreatedString = "Opprettelse av database
vellykket";
        private static string errorString;
        private static string logFile = @"Database\CreateDatabase.log";
        private static bool autoConnecting= false;
        private static bool successfulLogin = false;
        private static bool databaseExist = false;
        private static bool databaseSuccessfullyCreated = false;
        private static SqlConnection con;
        private static SqlCommand command;

        public static bool AutoConnect()
        {
            autoConnecting = true;
            dataSource = System.Environment.MachineName;
            StartCom(dataSource);

            if (successfulLogin == true)
            {
                autoConnecting = false;
                return successfulLogin;
            }
            else if (successfulLogin == false)
            {
                dataSource = @".\SQLEXPRESS";
                StartCom(dataSource);
                autoConnecting = false;
                return successfulLogin;
```

```
            }
            else
            {
                errorString = "En feil oppstod, kontakt programutvikler!";
                MessageBox.Show(errorString, "Feil", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                autoConnecting = false;
                return successfulLogin;
            }
        }

        public static void StartCom (string serverName)
        {
            try
            {
                conString = string.Format("Data Source ={0}; Initial Catalog
                     = master; Integrated Security = True; Connection Timeout=5", serverName);
                con = new SqlConnection(conString);
                con.Open();
                successfulLogin = true;
                dataSource = serverName;
                con.ChangeDatabase(databaseName);
                con.Close();
                conString = string.Format("Data Source ={0}; Initial Catalog ={1}; Integrated
                    Security = True; Connection Timeout=5", dataSource, databaseName);
                databaseExist = true;

            }
            catch (Exception error)
            {
                //Don't display errormessage on autologin
                if (autoConnecting == true && error.Message.Contains
                    ("'Alarmsystem' does not exist"))
                {
                    CreateDatabase();
                }
                else if (error.Message.Contains("error: 26") || error.Message.Contains
                    ("error: 40") && autoConnecting == false)
                {
                    errorString = "Feil servernavn";
                    MessageBox.Show(errorString);
                }
                // Create database if it doesn't exist
                else if (error.Message.Contains("'Alarmsystem' does not exist"))
                {
                    errorString = "Databasen eksisterer ikke";
                    CreateDatabase();
                    MessageBox.Show("Databasen eksisterer ikke.", "Feil", MessageBoxButtons.OK);
                }
                else if (autoConnecting == true)
                {
                    //Don't display errormessage on autologin
                }

                else
                {
                    MessageBox.Show(error.Message);
                }
            }
        }

        /// <summary>
```

```
        /// Uses SQLCMD.exe to create database from sql file. Search for error-messages in log
file afterwars and determines
        /// if it was successfully created
        /// </summary>
        static private void CreateDatabase()
        {
            errorString = "Error on creating database. Please do it manually.";
            try
            {
                Process sqlCreateDatabaseScript = new Process();
                sqlCreateDatabaseScript.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
                sqlCreateDatabaseScript.StartInfo.FileName = "SQLCMD.EXE";
                sqlCreateDatabaseScript.StartInfo.Arguments = string.Format
                    ("-S {0} -E -w 166 -e -i Database\\CreateDatabase.sql -o
                    Database\\CreateDatabase.log", dataSource);
                sqlCreateDatabaseScript.Start();
                if (File.Exists(logFile))
                {
                    StreamReader file = new StreamReader(logFile, true);
                    string fileContent = file.ReadToEnd();
                    if (fileContent.Contains(string.Format
                           ("Changed database context to '{0}'", databaseName)))
                    {
                        databaseExist = true;
                        if (fileContent.Contains("error") || fileContent.Contains("Error") ||
fileContent.Contains("unknown") || fileContent.Contains("Unknown"))
                        {
                            databaseSuccessfullyCreated = false;
                        }
                        else
                        {
                            databaseSuccessfullyCreated = true;
                            MessageBox.Show((databaseSuccessfullyCreatedString));
                        }
                    }
                }
                if (databaseExist == true && databaseSuccessfullyCreated == false)
                {
                    try
                    {
                        command = new SqlCommand(conString, con);
                        command.CommandText = string.Format("DROP DATABASE {0}", databaseName);
                        command.ExecuteNonQuery();
                        databaseExist = false;
                    }
                    catch (Exception)
                    {
                        MessageBox.Show(errorString);
                        //Console.WriteLine(errorString);
                    }
                }
                if (databaseExist == false || databaseSuccessfullyCreated == false)
                {
                    MessageBox.Show(errorString);
                    //Console.WriteLine(errorString);
                }
            }
            catch (Exception error)
            {
                MessageBox.Show(error.Message);
                //Console.WriteLine(error.Message);
            }
```

```
        }

        #region Properties
        public static bool DatabaseExist
        {
            get { return databaseExist; }
        }
        public static bool SuccessfullLogin
        {
            get { return successfulLogin; }
        }
        public static string DatabaseName
        {
            get { return databaseName; }
        }
        public static string ConnectionString
        {
            get { return conString; }
            set { conString = value; }
        }
        public static string ServerName
        {
            get { return dataSource; }
        }
        #endregion
    }
}
```

Subscriber.cs (class)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SQLCommunication
{
    public class Subscriber
    {

        public static event EventHandler SubscriberCreated;

        //Auto-implementerte properties
        public string Email { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public int Telephone { get; set; }

        //Constructor for the subscriber invokes the method OnSubscriberCreated,
        which raises an event
        public Subscriber(string email, string firstName, string lastName, int phoneNumber)
        {
            Email = email;
            FirstName = FormatName(firstName);
            LastName = FormatName(lastName);
            Telephone = phoneNumber;
            OnSubscriberCreated();
        }
```

```
        //Formats names with capital first letter
        private string FormatName(string name)
        {
            name = name.ToLower();
            return char.ToUpper(name[0]) + name.Substring(1);
        }

        //Method to raise the event SubscriberCreated
        protected virtual void OnSubscriberCreated()
        {
            if (SubscriberCreated != null)
            {
                SubscriberCreated(this, EventArgs.Empty);
            }
        }
        //See SQL definition for comments and implementation
        public void UpdateSubscriber(string newEmail, string newFirstName,
              string newLastName, int newPhoneNumber)
        {
            SQL.UpdateSubscriber(this, newEmail, newFirstName, newLastName, newPhoneNumber);
        }

        public void RemoveSubscriberFromSQL()
        {
            SQL.DeleteSubscriber(this);
        }

        public void SubscribeTo(AlarmtypesEnum alarmtype)
        {
            SQL.SubscribesTo(this, alarmtype);
        }

        public void UnsubscribeTo(AlarmtypesEnum alarmtype)
        {
            SQL.UnsubscribeTo(this, alarmtype);
        }

        // Method mainly for testing, to see if the "subscribesto" table worked
        public void ShowSubscribeList()
        {
            Console.WriteLine(this.FirstName + " " + this.LastName +
              " subscribes to the following alarms:");
            List<string> AlarmList = SQL.GetSubscriberAlarmList(this);
            foreach (string alarm in AlarmList)
            {
                Console.WriteLine(alarm);
            }
        }
    }
}
```

VEDLEGG D – Arduino Kode – IA2122 - 2016

Main_Arduino_Class:

```
//--------------------Include----------------------//             // Date: 21.04.2016

#include "Arduino.h"
#include "Csharp_Command.h"
#include "Temperatur.h"
#include "Pir.h"
#include "LCD.h"
#include "Buzzer.h"

bool DefaultTemperatureShow = false;

//--------------------Setup------------------------//             // Date: 21.04.2016

void setup() {

  // put your setup code here, to run once:

 Serial.begin(9600);
 temperature.TemperatureSetup();
 lcd.LCDSetup();
 pir.PirSetup();
 pir.PirCalibration(30);
 buzzer.BuzzerSetup();

}

//------------------Main--------------------------//             // Date: 21.04.2016

void loop() {

  // put your main code here, to run repeatedly:


if(DefaultTemperatureShow)
{
  lcd.Default();
}

if(Serial.available() > 0)

{
  char Comp = command.CommandCode();
  switch (Comp)

{
    case '1':       //DEFALUT
    DefaultTemperatureShow = true;
    buzzer.BuzzerReset();
    command.DataFormat();
    break;
```

```
   case'2':       // ALARM ON
   DefaultTemperatureShow = false;
   command.DataFormat();
   lcd.DeleteLine();
   buzzer.BuzzerOn();
   lcd.PrintLine("Alarm On");
   break;

}
Comp = 0;

}

}
```

Buzzer.cpp:

```
#include "Arduino.h"
#include "Buzzer.h"

const int BuzzPin = 7;                  // Pin                          // Date: 21.04.2016
int count = 0;
BuzzerClass::BuzzerClass(){}

void BuzzerClass::BuzzerSetup()          // Set buzzer pin and output          // Date: 21.04.2016

{
  pinMode(BuzzPin, OUTPUT);
}

void BuzzerClass:: BuzzerOn()            // Set pin, frequency and duration      // Date: 21.04.2016
{
  if( count < 1)
  {
    tone(BuzzPin, 1000, 2000);
    count++;
  }
}

void BuzzerClass::BuzzerReset()
{
  count = 0;
 }

BuzzerClass buzzer = BuzzerClass();
```

Buzzer.h:

```
//--------------------//          // Date: 21.04.2016
#ifndef Buzzer_h
#define Buzzer_h

class BuzzerClass
{
   public:
      BuzzerClass();
   void BuzzerSetup();
   void BuzzerOn();
   void BuzzerReset();
};

extern BuzzerClass buzzer;        // this is use to access methods
#endif
```

CSharp_Commando.cpp:

```
#include "Arduino.h"
#include "Csharp_Command.h"
#include "Pir.h"
#include "Temperatur.h"

CommandClass::CommandClass() {}
void CommandClass:: CommandSetup()

{

//Empty

}

char CommandClass:: CommandCode()                    // Read command code from program
// Date: 21.04.2016

{
  char pc = Serial.read();
  return pc;
}

void CommandClass::DataFormat()                      // Send data value to program in C-Sharp
// Date: 21.04.2016
{
  String Temp = String(temperature.TemperatureSensor(),2);
  String Pir = pir.PirSensor();

  Serial.print("#P:");
  Serial.print(Pir);
  Serial.print("\t");
  Serial.print(" #T:");
```

```
  Serial.println(Temp);
}

void CommandClass::DataFlush()

{
  while(Serial.available())
  Serial.read();
}

CommandClass command = CommandClass();
```

CSharp_Command.h:

```
//--------------------//          // Date: 21.04.2016
#ifndef Csharp_Command_h
#define Csharp_Command_h

class CommandClass
{
    public:
    CommandClass();
      void CommandSetup();
      char CommandCode();
      void DataFormat();
      void DataFlush();
};

extern CommandClass command;
#endif
```

LCD.cpp:

```
#include "Arduino.h"
#include "LCD.h"
#include "Temperatur.h"
#include <LiquidCrystal.h>
LiquidCrystal LCD(12,11,5,4,6,2);

LCDClass::LCDClass() {}

void LCDClass::LCDSetup()              // Set columns and row                    // Date: 21.04.2016
{
  LCD.begin(16,2);
 }

void LCDClass::MaxTemperatureAndPir()        // Print out temperature when its to High and Pir is
On after user choice              // Date: 21.04.2016
{
  String Temp = String(temperature.TemperatureSensor(),2);
  DeleteLine();
  PrintLine("T-High: ");
```

```
  PrintLine(Temp);
  PrintLine(" C");
  LCD.setCursor(0,1);
  PrintLine("Motion detected");
  LCD.setCursor(0,0);
}

void LCDClass::MinTemperatureAndPir()         // Print out temperature when its to Low and Pir is On
after user choice                // Date: 21.04.2016
{
  String Temp = String(temperature.TemperatureSensor(),2);
  DeleteLine();
  PrintLine("T-Low: ");
  PrintLine(Temp);
  PrintLine(" C");
  LCD.setCursor(0,1);
  PrintLine("Motion detected");
  LCD.setCursor(0,0);
}

void LCDClass::MaxTemperature()            // Print out temperature when its to High after user choice
// Date: 21.04.2016

{
  String Temp = String(temperature.TemperatureSensor(),2);
  DeleteLine();
  PrintLine("Temp High: ");
  LCD.setCursor(0,1);
  PrintLine(Temp);
  PrintLine(" C");
  LCD.setCursor(0,0);
}

void LCDClass::MinTemperature()       // Print out temperature when its to Low after user choice
// Date: 21.04.2016

{
  String Temp = String(temperature.TemperatureSensor(),2);
  DeleteLine();
  PrintLine("Temp Low: ");
  LCD.setCursor(0,1);
  PrintLine(Temp);
  PrintLine(" C");
  LCD.setCursor(0,0);
}

void LCDClass::Pir()         // Print out Pir when its On after user choice                // Date: 21.04.2016
{
  DeleteLine();
  PrintLine("Motion detected");
}
```

```
void LCDClass::Default()          // Print out temperature.          // Date: 21.04.2016
{
  String Temp = String(temperature.TemperatureSensor(),2);
  DeleteLine();
  PrintLine(Temp);
  PrintLine(" degree C");
  delay(500);
}

void LCDClass:: PrintLine(String Line)        // Print tekst                // Date: 21.04.2016
{
  LCD.print(Line);
}

void LCDClass:: DeleteLine()                                                // Delete tekst
{
  LCD.clear();
}

LCDClass lcd = LCDClass();
```

LCD.h:

```
//--------------------//          // Date: 21.04.2016

#ifndef LCD_h
#define LCD_h

class LCDClass
{
   public:
     LCDClass();
     void LCDSetup();
     void MaxTemperatureAndPir();
     void MinTemperatureAndPir();
     void MaxTemperature();
     void MinTemperature();
     void Pir();
     void Default();
     void PrintLine(String line);
     void DeleteLine();
};

extern LCDClass lcd;
#endif
```

Pir.cpp:

```
#include "Arduino.h"
#include "Pir.h"
#include "LCD.h"

const int PirPin = 13;

PirClass::PirClass() {}


void PirClass::PirSetup()                    // Set Pir                    // Date: 21.04.2016

{
 pinMode(PirPin, INPUT);
 digitalWrite(PirPin, LOW);
}

void PirClass::PirCalibration(int CalibrationTime)      // Start Calibration         // Date: 21.04.2016

{
  lcd.DeleteLine();
  lcd.PrintLine("Calibrate Sensor");
  delay(5000);
  lcd.DeleteLine();
  for(int i = 0; i< CalibrationTime; i++)
  {
   lcd.PrintLine(".");
   delay(5000);
  }

  lcd.DeleteLine();
  lcd.PrintLine("Done");
  lcd.DeleteLine();
  lcd.PrintLine("Welcome");
}


String PirClass::PirSensor()                 // Return state of alarm     // Date: 21.04.2016

{
  if(digitalRead(PirPin) == HIGH)
  {
    return "ON";
  }

  else
  {
    return "OFF";
  }
}

PirClass pir = PirClass();
```

Pir.h:

```
//--------------------//          // Date: 21.04.2016

#ifndef Pir_h
#define Pir_h


class PirClass
{
   public:
   PirClass();
   void PirSetup();
   void PirCalibration( int calibrationTime);
   static  String PirSensor();
};

extern PirClass pir;
#endif
```

Temperatur.cpp:

```
#include "Arduino.h"
#include "Temperatur.h"
#include "LCD.h"

const int SensorPin = A0;
unsigned long PreMillisA = 0;
long OnTimeA = 750;
long OffTimeA = 1000;

const int numReadings = 10;
int readings[numReadings];
int readIndex = 0;
int total = 0;
int average = 0;

TemperatureClass::TemperatureClass() {}        // Initialize all reading to 0     // Date: 21.04.2016

void TemperatureClass::TemperatureSetup()
{
  for(int thisReading = 0; thisReading < numReadings; thisReading++)
  {
    readings[thisReading] = 0;
  }
}

float TemperatureClass::TemperatureSensor()         // Return Temperature value     // Date: 21.04.2016

{
total = total - readings[readIndex];
int SensorVal = analogRead(SensorPin);
float Voltage = (SensorVal/ 1024.0) * 5.0;
```

```
float Temperature = (Voltage - .5) *100;
readings[readIndex] = Temperature;
total = total + readings[readIndex];
readIndex = readIndex +1;


if(readIndex >= numReadings)
{
  readIndex = 0;
}

average = total / numReadings;
return average;
}

void TemperatureClass::TemperatureView(unsigned long CurrentMillis)     // View temperature value
on LCD screen                        // Date: 21.04.2016

{
  if(CurrentMillis - PreMillisA >= OnTimeA)
  {
    PreMillisA = CurrentMillis;
  }

  else if(CurrentMillis - PreMillisA >= OffTimeA)
  {
    lcd.DeleteLine();
    String Temp = String(TemperatureSensor(),2);
    lcd.PrintLine(Temp);
    lcd.PrintLine(" degrees C");
  }
}

TemperatureClass temperature = TemperatureClass();
```

Temperatur.cpp:

```
//--------------------//          // Date: 21.04.2016

#ifndef Temperature_h
#define Temperature_h

class TemperatureClass
{
   public:
   TemperatureClass();
   void TemperatureSetup();
   static float TemperatureSensor();
   void TemperatureView(unsigned long CurrentMillis);
};

extern TemperatureClass temperature;
#endif
```

Kommando koder for alarm.			
Nr:	Description	Code	Information
1	Default	1	Ingen alarm er utløst, vis temperatur Verdi på LCD skjerm.
2	Alarm ON	2	Signalisere bruker at alarm er på.

SQL-Script

```
-- The following statements creates the database and tables for the Alarmsystem Database

CREATE DATABASE Alarmsystem
go
USE Alarmsystem
Go

CREATE TABLE SUBSCRIBER
(
Email char (30),
FirstName char(18) NOT NULL,
LastName char(18) NOT NULL,
Telephone int,
CONSTRAINT PK_SUBSCRIBER PRIMARY KEY (Email)
)

CREATE TABLE TEMPERATURE
(
[Time] datetime,
Temperature float NOT NULL,
CONSTRAINT PK_TEMPERATURE PRIMARY KEY ([Time])
)

CREATE TABLE ALARMTYPE
(
Alarm char(30),
Limit float,
[Description] varchar(100),
CONSTRAINT PK_ALARMTYPE PRIMARY KEY (Alarm)
```

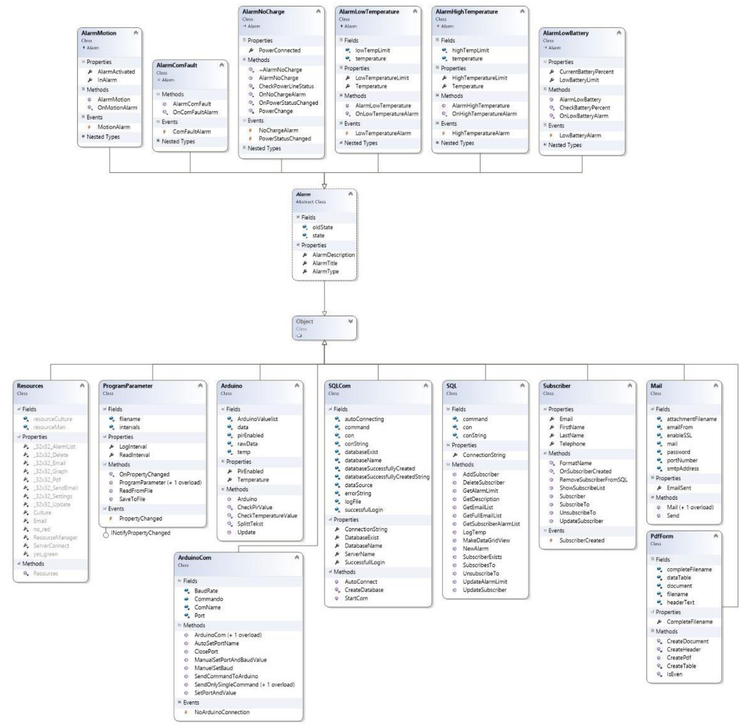
```
)

CREATE TABLE ALARM
(
Alarm char(30),
[Time] datetime,
CONSTRAINT PK_ALARM PRIMARY KEY (Alarm, [Time]),
CONSTRAINT FK_ALARMTYPE_ALARM FOREIGN KEY (Alarm)
REFERENCES ALARMTYPE (Alarm)
)

CREATE TABLE SUBSCRIBESTO
(
Alarm char(30),
Email char(30),
CONSTRAINT PK_SUBSCRIBESTO PRIMARY KEY (Alarm, Email),
CONSTRAINT FK_ALARMTYPE_SUBSCRIBESTO FOREIGN KEY (Alarm)
REFERENCES ALARMTYPE (Alarm),
CONSTRAINT FK_SUBSCRIBER_SUBSCRIBESTO FOREIGN KEY (Email)
REFERENCES SUBSCRIBER (Email)
)

/* Metode for å legge til en ny abbonent */
CREATE PROCEDURE NewSubscriber
@Email char(30), @FirstName char(18), @LastName char(18), @Telephone int
AS
INSERT INTO SUBSCRIBER
VALUES (@Email, @FirstName, @LastName, @Telephone)
GO

/* Metode for å lagre en ny temperaturmåling */
```

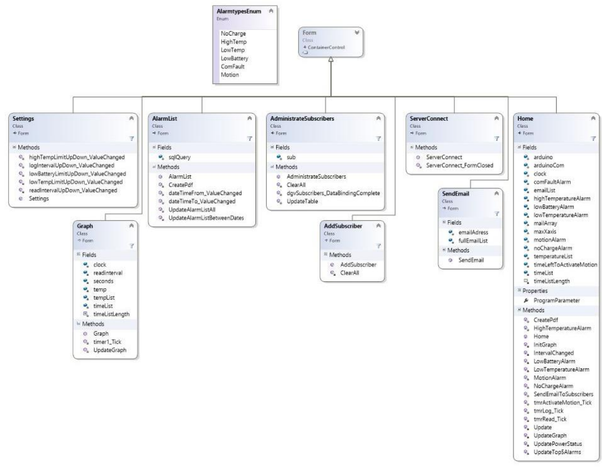
```
CREATE PROCEDURE MeasureTemp
@Temperature float
AS
INSERT INTO TEMPERATURE
VALUES (CURRENT_TIMESTAMP ,@Temperature)
GO

/* Metode for å lagrer en ny alarm i alarmhistorikken */
CREATE PROCEDURE NewAlarm
@Alarm char(30)
AS
INSERT INTO ALARM
VALUES (@Alarm, CURRENT_TIMESTAMP)
GO

/* Metode for å oppdatere grenseverdiene for alarmene */
CREATE PROCEDURE UpdateLimit
@AlarmType char(30), @NewLimit float
AS
UPDATE ALARMTYPE
SET Limit = @NewLimit
WHERE ALARMTYPE.Alarm = @AlarmType
GO

/* Metode som returnerer alarmgrense for en gitt alarm */
CREATE PROCEDURE GetLimit
@AlarmType char(30)
AS
SELECT Limit
FROM ALARMTYPE
WHERE Alarm = @AlarmType
```

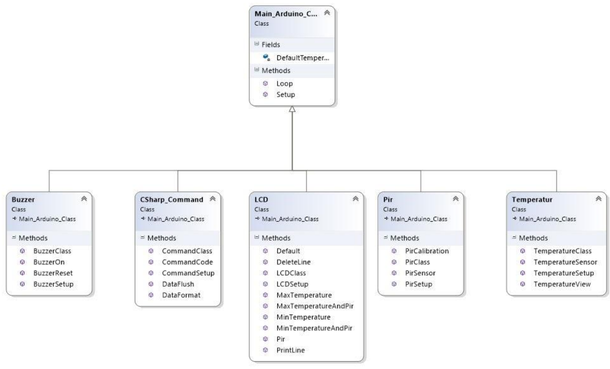
```
GO

/* Metode som returnerer beskrivelsen til en gitt alarm */
CREATE PROCEDURE GetDescription
@AlarmType char(30)
AS
SELECT [Description]
FROM ALARMTYPE
WHERE Alarm = @AlarmType
GO

-- Metode som oppdaterer en eksisterende abonnent med nye verdier
CREATE PROCEDURE UpdateSubscriber
@Email char(30), @NewEmail char(30), @FirstName char(18), @LastName char(18), @Telephone int
AS
ALTER TABLE SUBSCRIBESTO NOCHECK CONSTRAINT ALL --skrur av FK før oppdatering
UPDATE SUBSCRIBER
SET Email = @NewEmail,
FirstName = @FirstName,
LastName = @LastName,
Telephone = @Telephone
WHERE Email = @Email
UPDATE SUBSCRIBESTO
SET Email = @NewEmail
WHERE Email = @Email
ALTER TABLE SUBSCRIBESTO CHECK CONSTRAINT ALL --setter FK tilbake etter oppdatering
GO

drop procedure UpdateSubscriber

-- Metode som sletter en abonent
```

```
CREATE PROCEDURE DeleteSubscriber
@Email char(30)
AS
DELETE SUBSCRIBESTO WHERE Email = @Email
DELETE SUBSCRIBER  WHERE Email = @Email
GO

-- Metode som legger en abonent i abonentliste
CREATE PROCEDURE NewSubscribesTo
@Email char(30), @Alarmtype char(30)
AS
INSERT INTO SUBSCRIBESTO
VALUES (@Alarmtype, @Email)
GO

CREATE PROCEDURE UnsubscribesTo
@Email char(30), @Alarmtype char(30)
AS
DELETE SUBSCRIBESTO
WHERE Email = @Email
AND ALARM = @Alarmtype
GO

-- Inserts the types of alarms needed
INSERT INTO ALARMTYPE
VALUES ('HighTemp', 30.0, 'Måleverdi for temperatur er over øvre grense'),
('LowTemp', 15.0, 'Måleverdi for temperature er under nedre grense'),
('LowBattery', 20, 'Batterispenningen er under nedre grense'),
('NoCharge', null, 'Datamaskinens strømforsyning er ikke tilkoblet'),
('ComFault', null, 'Kommunikasjon med Arduino feilet'),
('Motion', null, 'Motion detected!')
```

VEDLEGG F – KlasseDiagram – IA2122 -2016







Use case diagram