PROSJEKT: IELET1002 Datateknikk (Vår-21)

Toppdokumentet

Det er ikke hensikten med dette prosjektet at det skal være styrt av en kravspesifikasjon, og innholdet i dette dokumentet må heller ikke betraktes som en kravspesifikasjon. Det handler absolutt ikke om et løp for å implementere mest mulig av den funksjonaliteten som nevnes i dette dokumentet.

Prosjektet skal snarere være en læringsreise, en inspirasjon hvor dere søker mest mulig forståelse og ferdigheter i å få et IoT-lignende system bestående av forskjellige program- og maskinvarekomponenter til å spille sammen i en større fungerende løsning. Ta dette prosjektet som en mulighet til inspirasjon og innsikt. Ikke minst innsikt i hvordan det er å jobbe i team for å skape noe sammen.

Det er fritt fram for å forkaste eller endre beskrevet funksjonalitet, og å legge til nye ideer som dere synes passer inn i scenariet. Det kan også være at dere velger å gå dypere inn i enkelte sider ved prosjektet, og vurderer det slik at dere må helt droppe enkelte punkter som er omtalt i dette dokumentet. Det er ok. En smart måte å jobbe på i så fall er å tenke godt igjennom helt fra starten hvilke typer funksjonalitet det er som henger sammen («tar du A, så må du ta B og C også»), og så går dere på utviklingen og ferdigstillelsen av den valgte gruppen funksjonalitet. Først når dere ser at dere vil komme greit i havn med den, så starter dere opp arbeidet med en annen gruppe funksjonalitet. Den framgangsmåten vil sikre at dere ikke jobber veldig bredt hele veien, men så får dere ikke helt ferdigstilt noen ting.

Det beste er om dere klarer å strukturere prosjektet slik at dere jobber bredt og fokusert på en grunnplattform som er slik at det etterpå, for et helt spekter av funksjonalitet, ikke krever så mye for å ferdigstille den ene modulen etter den andre.

Her er noen sentrale elementer som alle gruppene uansett må ha med:

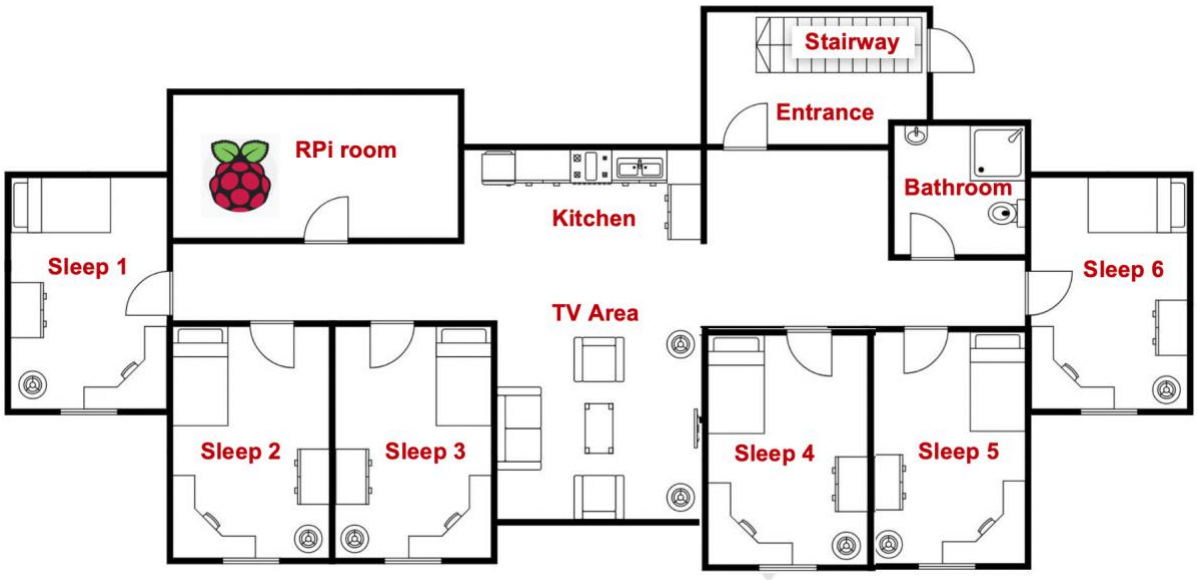
- Grunnleggende programmering av ESP32 og Raspberry Pi (eller en alternativ enhet, som eksempelvis Laptop, som kan kjøre Python), med samhandling gjennom Circus of Things

- Mer om programmering: Beherske og bruke et IDE, importere og bruke biblioteker, benytte seg av et bredt spekter av grunnleggende elementer og språklige konstruksjoner i Arduino C og Python...
- ... slik som å kunne stille opp logiske betingelser, benytte valgstrukturer for å styre programflyten, løkker for å repetere kodeblokker, egendefinerte funksjoner for å strukturere programløsningen
- ESP32: Koble til sensor- og aktuatorkretser (lyssensor, trykknapper, LEDs, temperatursensor, buzzer, 7-segmentdisplay, potensiometer, etc fra Arduino-kitet), og styre dem vha relevante instruksjoner.
- ESP32: Utnytte interne ressurser for å stille opp løsninger med avbrudd, styring av «pådragseffekten» (PWM), etc
- Kunne benytte WiFi-funksjonaliteten i ESP32 og Raspberry Pi for å kommunisere med internettplattformen CoT
- RPi: Aksessere databaser for å lese ut, lagre og analysere data – eksempelvis lese værdata fra Meteorologisk Institutt via deres offentlig tilgjengelige databasegrensesnitt (APIer)

Noe av det som vi kommer til å legge vekt på når vi vurderer prosjekt-besvarelsen, er at dere har inkludert de 3 hovedelementene (RPi, ESP32 og CoT), og gjerne energiproduksjonen fra solcellepanelet i løsningen. Videre at dere har gjort solid koding på det som er laget, at arbeidsmengden er rimelig (å kutte funksjonalitet bare for å slippe billigere unna, er ikke lurt rent vurderingsmessig), at dere har gjennomført prosjektet på en bra måte (prosjektstyring, kvalitetskontroll/testing, samarbeidet og arbeidsfordelingen innad i gruppa), og har skrevet en god rapport. Oppfinnsomhet, hardt arbeid, tilleggsfunksjonalitet, o.l. vil selvsagt telle positivt, men hovedsaken med prosjektet er likevel at gruppene opplever at de har lært mye interessant datateknikk og det å samarbeide i et team.

1. Det tenkte scenario

Prosjektet tar utgangspunkt i et kollektiv med 6 studenter, og at det råder relativt strenge smittevernskrav pga. Corona-pandemien. Byen hvor kollektivet ligger har en Smart City-profil, og det oppmuntres fra kommunens side til en bred introduksjon av teknologi-løsninger både i offentlig sektor, i det private næringsliv, og i husholdningene. Kommunen følger opp med tilskuddsprogram for prosjekter som kommer inn under Smart City-begrepet, og gjennom det har eieren av kollektivet sikret seg startmidler til et totrinns prosjekt.



Hun har i første omgang fått tilslag på et ***forprosjekt*** som skal demonstrere hvordan en ønsket sluttløsning vil bli. Dersom kommunens sjefingeniør fra Smart City-seksjonen setter sitt godkjent-stempel på forprosjekt-løsningen, vil det utløse et betydelig tilskudd til ***hovedprosjektet***, selve implementasjonen og igangkjøring av sluttløsningen.

Eieren av kollektivet har så forespurt beboerne, som alle beleilig nok er elektroingeniørstudenter, om å gjennomføre pilotprosjektet, med løfte om timelønn underveis i, samt en bonus i enden dersom det tilfredsstiller hennes funksjonelle og sjefingeniørens kommunalbyråkratiske og tekniske krav.

Beboerne (dvs. dere i prosjektgruppen) takker ja til jobben! :-)

> *Prosjektet dere skal gjennomføre tilsvarer **forprosjektet** i scenariet beskrevet over*
>
> ***Det skal demonstreres gjennom en blanding av faktisk implementasjon og simuleringer, hvordan en komplett løsning vil oppføre seg.***

2. Definisjoner og begreper

2.1. **Konsollet** - Vi tenker oss en innbygd, batteridrevet ESP32 med trykknapper og display, som hele tiden er koblet til internett via WiFi. Dette må beboerne bære med seg når de er oppholder seg i kollektivet. Benyttes for å booke ressurser, vise enkel status på disse, og annen funksjonalitet som gruppene ønsker å legge inn). Konsollet skal ikke implementeres som en selvstendig enhet, men inngår med sin ønskede funksjonalitet i det vi kan kalle *ESP32-ctrl*.

2.2. **ESP32-ctrl** – Alle de ESP32-relaterte kretsene (sensorer, aktuatorer og andre komponenter) som skal realiseres i kollektivet. Funksjonsmessig kan ESP32-ctrl deles inn i delene konsoll (se pkt. 2.1), soveromsrelatert funksjonalitet, inngangspartiet (ringeklokke + trappeoppgang). Altså en alt-i-ett-løsning, bare for å kunne demonstrere konseptene i kollektiv-eiers forprosjekt.

2.3. **Sjekke inn/ut** – Av hensyn til både smittesporing, brannvern og til informasjon for besøkende som ringer på og ønsker å treffe en spesifikk beboer, så er det ønskelig å holde rede på hvilke beboere som er tilstede i kollektivet til en hver tid. Gruppene kan foreslå og implementere en løsning på dette. Et par tips: Registrere beboeren inn i kollektivet når vedkommende taster inn pinkoden ved inngangsdøra? Sjekke ut ved å stryke konsollet over et magnetpunkt ved siden av inngangsdøra? (kan bruke Hall-effektsensoren i ESP32...)

2.4. **CoT** - Circus of Things, en IoT-plattform som benyttes både for å visualisere signaler/data som kommer fra ESP32ene eller RPien, og for å gi mulighet for styring av enkelte elementer i kollektivet via konfigurerte aktuatorer. CoT er plattformen for dashbordet (se eget punkt for dette). I tillegg så kan det holde en form for statusregistre som benyttes for kommunikasjon ESP32ene i mellom og mellom ESP32ene og RPien

2.5. **CoT Dashbord** - En visualisering av alle sentrale verdier og parametre i kollektivet, samt noen eksterne. Eksempler (ikke komplett liste, og punktene 2.2.1 -2.2.3 kan man f.eks. tenke seg kombinert i en større smartmetervisning) :

- 2.5.1. **Målinger i kollektivet**, slik som temperatur på soverommene.
- 2.5.2. ***Eksterne parametre:*** utetemperatur, vindstyrke, strømprisene
- 2.5.3. ***Løpende estimat av strømregningen*** (betales ukentlig)
- 2.5.4. ***Lokalisering av beboerne:*** Antall beboere inne i kollektivet akkurat nå (og hvor – dvs. i de tilfellene at de er på soverommet eller har booket seg inn på badet, kjøkkenet eller TV-stua. Eller har statusen «ute», altså ikke i kollektivet)

2.6. **Smartmeteret** - Et element i CoT, og som en del av dashbordet (eventuelt er det et eget/separat dashbord). Her viser dere de forskjellige former for energiforbruk det er i kollektivet, samt produksjonsbidraget fra solcellepanelet. Eksempler nedenfor, men gruppene bestemmer selv hva de ønsker å vise i Smartmeteret (sjekk gjerne hva faktiske smartmetre kan vise) :

2.6.1. ***Totalt elektrisitetsforbruk*** akkurat nå, samlet siste uke, siste kvartal, siste halvår, så langt i kalenderåret + sammenlignende data fra forrige år i samme perioder (simulerte data basert på det innprogrammerte bruksmønster til beboerne, historiske utetemperaturer, tidspunkter for sol opp/ned, skydekke, etc)

2.6.2. ***Produsert elektrisitet fra solcellepanelene*** akkurat nå, samlet siste uke, etc.

2.6.3. ***Fordelt forbruk:*** Forbruket for hvert av soverommene, dusjing, varmekabler, varmtvannstank, oppvaskmaskin, belysning, komfyr, oppvarming (panelovner)

2.7. **RPi** - En Raspberry Pi satt opp som en sentral enhet som innhenter eksterne data (meteorologiske data, strømpriser, o.l.), og som kan ha kollektivmodellen og beboernes bruksmønstre programmert inn (eventuelt delt med ESP32ene).

2.8. **Ressursene** - Dette er felles fasiliteter i kollektivet som det enten kan være bruksbegrensninger på pga. smittevernhensyn, eller som det av andre grunner må holdes rede på bruken/forbruket av. Og i tillegg slikt som det er krav om styring av fra kommunens side.

2.8.1. **Baderommet** m/toalett har egne smittevernsregler, og må bookes fra den enkelte beboers konsoll. Dere kan eksempelvis legge opp til at det kan bookes for 10 eller 30 minutter.

2.8.2. **Varmekablene** på badet kan utstyres med natt- og/eller dagsenkning.

2.8.3. **Varmtvannstanken** kan utstyres med senkning som i 2.8.2

2.8.4. **Kjøkkenet** Her finner vi i alle fall komfyren som obligatorisk ressurs som det ligger bruksbegrensninger på (maks-antall innenfor samme tidsluke) og må bookes. Kanskje også en mikrobølgeovn? Ut over komfyren, så defineres smittevernsregler, bookingsystem, etc. av hver enkelt gruppe.

2.8.5. **TV-stua** kan bookes fra konsollet, men rommer bare 3 av gangen.

2.8.6. **Soveromsbelysningen** kan styres i løpet av dagen. Jo mer dagslys, desto mindre lysstyrke i pærene

2.8.7. **Termostaten** på hvert soverom settes av hver enkelt beboer via CoT, eller eksempelvis styres automatisk med døgnplaner programmert i RPien eller i den enkelte ESP32

2.8.8. **Luftevindu**. Øverst oppe på veggen i soverommene er det et bredt/smalt vindu som kan styres av konsollet, ved at en servomotor åpner/lukker. Vinduet åpner i overkant og inn i rommet, inntil 80 grader. Luftevinduet er det eneste vinduet som lar seg åpne på soverommene, unntatt ved brann - da vil låsemekanismen i de store vinduene på soveommet automatisk åpne (brann-åpning av låsen på disse vinduene er ikke en del av prosjektet).

2.8.9. **Takvifte**. I hvert soverom er det en takvifte som den enkelte beboer kan styre via konsollet eller etter innhentet ute-temperatur, solens himmelposisjon og skydekke og målt romtemperatur. Takvifta styres av en DC-motor.

2.8.10. **Oppvaskmaskinen** må modelleres, dvs. bruken av den. Med utgangspunkt i beboernes bruksmønster/måltider må det estimeres når den er full og må kjøres, og det tilhørende el-forbruket må beregnes og rapporteres til smartmeteret.

2.8.11. **Solcellepanelet** på taket. Beregn 150 kvm. tilgjengelig takflate. Taket er flatt, men heller 20 grader Nord-->Sør. Bidraget må beregnes og være basert på tidspunktene for soloppgang og solnedgang, solas bane (høyde) på himmelen, og graden av skydekke. Disse dataene samles inn av RPien, som beregner bidraget fra panelet og viser det i smartmeteret (strøm-måleren) i Dashbordet i CoT

2.9. **Booking** – Ressursene i kollektivet må kunne bookes av beboerne, og det skal skje i konsollet. Det er opp til gruppene å foreslå en fungerende bookingløsning, som vi kan tenke oss foregår ved bruk av en kombinasjon av trykknapper, LEDs, 7-segmentdisplay. Det er også mulig å velge å benytte seriemonitoren på en tilknyttet laptop.

3. Gjennomføringen av prosjektet

Vi anbefaler at arbeidsoppgavene i prosjektet deles inn i en ESP32-del og en RPi-del, hvor førstnevnte tar for seg programmering av funksjonaliteten som skal ligge i ESP32-kontrollerne (konsollene og annen ESP32-lokalisert funksjonalitet), mens den andre dreier seg om funksjonaliteten i Raspberry Pien. Begge delene inkluderer CoT-linket funksjonalitet, og ***det er viktig å bestemme seg tidlig for hvordan synkronisering/kommunikasjonen***

ESP32ene i mellom og mellom RPien og ESP32ene skal implementeres.

3.1. **ESP32-delen** - Funksjonalitet i kollektivet m/ESP32 og CoT
Jobben som skal gjøres her kan ***deles inn i modulene som er listet opp nedenfor***. Flere av modulene vil være overlappende, og det vil også være behov for å binde sammen enkelte av dem (de avhenger av hverandre).

3.1.1. **ESP32-ctrl** (se også pkt. 2.2)
Består av en ESP32 og tilkoblete sensorer og aktuatorer. Vi kobler til alle de nødvendige eksterne komponenter for å emulere det portable konsollets funksjonalitet, og i tillegg alle de komponentene som trengs for å dekke soverommets og inngangspartiets/trappeoppgangens funksjonalitet. Dvs. at vi ikke har et separat portabelt konsoll i tillegg til et fastkonsoll for soverommet og ett for inngangspartiet, men alle tre er i det vi kan kalle "ESP32-ctrl". Denne kommuniserer primært med CoT.

- **Forslag til tilkoblete kretser/komponenter:**
  > Trykknapp(er)
  > Buzzer
  > 7-segmentdisplay og/eller Serial Monitor
  > DC-motor (m/motordriver), Servo
  > LEDs
  > Potensiometer
  > LDR (lyssensor)
  > Temperatursensor
- **Visning av informasjon**, data og statuser i 7-segment
- **Oppdatert statusvisning i CoT** skal gi varsling (lyd og visuelt). Varsling må kunne slås av.
- **Booke baderom**, kjøkken og TV-stue (i CoT)
- **Sjekke romstatus** (hvor mange er i hvert rom?). Vises i 7-segment, kan f.eks. skippe fra rom til rom med trykknapp
- **Starte/stoppe takvifte** (DC-motor)
- **Åpne/lukke luftevindu** (servo)
- **Aktivere/deaktivere termostat** (temperatur skrives inn i CoT, men ny temperatur må i tillegg aktiveres. Termostaten deaktiveres = panelovnen er PÅ hele tiden. Panelovnen kan f.eks. slås av ved å sette temp = 0 i CoT og deretter aktivere termostaten)

- **Åpne ytterdørens kodelås** med pin-kode
- **Registrere bevegelse i trappeoppgangen** (2 x break-beam). Skal brukes for å telle hvor mange som er i kollektivet, og gi indikasjon i konsollet
- **Low power-modus**. Konsollet skal kunne settes i én eller flere low power-modi, og deretter vekkes opp ved innkommende signal

3.1.2. Soverommene

- Måle romtemperaturen
- Måle lysstyrken i rommet
- Aktivere/deaktivere termostaten til panelovnen(e)
- Åpne/lukke luftevindu
- Starte/stoppe takvifte
- Aktivere/deaktivere døgnplaner og lysstyring

Se forøvrig tilleggsinformasjon ved slutten av hovedpkt. 3

3.1.3. Kjøkkenet

Denne definerer dere selv!

Stikkord: Begrenset hvor mange som kan benytte komfyr samtidig. Må bookes. Strømforbruk må registreres. Antall måltider telles for å estimere når opp-vaskmaskinen er full (benyttes i modelleringen i del 2 av prosjektet)

3.1.4. Inngangsdøren

Det er bestemt at maks antall personer som til enhver tid kan være inne i kollektivet er 18, inklusive beboerne. Besøkende må assosieres med en av beboerne. Hver beboer kan maks ha 2 besøkende av gangen. Hver ESP32-ctrl skal ha komponenter som emulerer ringeknappen og kodelåsen ved inngangsdøren.

- **Kodelåsen:** realisert med potensiometer, 3 LEDs (eller et 7-segment-display) og trykknapp(er),
- **Respons på riktig pinkode**: det sjekkes om maks antall personer i kollektivet er nådd, og hvis ja så må den beboer som har den først ankomne besøkende be vedkommende om å gå (bekreftes av beboeren med et knappetrykk).
- **Ringeknapp** (angir beboer som vedkommende skal besøke): Knappen trykkes og holdes i minst 3 sekunder, og deretter trykkes knappen det antall ganger som tilsvarer beboer-nummeret (vi antar at det henger et oppslag med

instruksjoner og beboerliste med nummer ved døra). Riktig beboer skal så få lyd og lyssignal på sin ESP32-ctrl, og må gi et signal (trykknapp) på at det er hans/hennes besøkende.

- **Respons på ringeknapp**: det sjekkes om maks antall personer i kollektivet er nådd, og hvis ja så må den beboer som har den først ankomne besøkende be vedkommende om å gå (trykknapp), men bare hvis den besøkende har vært der minst 30 minutter. Har den besøkende vært der mindre enn 30 minutter, får den som ringer på "rødt lys" og beskjed om antall minutter før han/hun kan prøve igjen.
- **Ringeknapp** (ingen angivelse av hvem av beboerne som vedkommende ønsker å treffe): det gis lyd/lyssignal på alle konsollene i kollektivet. Vi antar at en av beboerne går til inngangsdøren og sjekker. Deretter må én av beboerne gi signal (trykknapp) på at det er hans/hennes besøkende, eller så trykker den som gikk til døra 2 ganger på knappen på sitt konsoll, og det betyr at det f.eks. bare var noen som spurte etter en gateadresse el. lign. Ringeren nulles ut av systemet.

3.1.5. TV-stuen

Denne definerer dere selv!

Som for kjøkkenet så er det begrensninger i antall beboere som kan oppholde seg der samtidig. Hva er et rimelig antall? Skal TV-stua bookes i tilknytning til kjøkkenressursen? Altså, skal lage mat på kjøkkenet gi prioritet i bruk av TV-stua (spise maten der)?

3.1.6. Trappeoppgangen

- Det skal telles hvor mange som går inn og ut av kollektivet, for å holde rede på hvor mange som er tilstede. Kan løses med 2 «break beams».

3.1.7. Baderommet

Se pkt. 2.1.7

Mer om soverommene:

Der overvåkes temperaturen og lysmengden. Vi kan styre termostaten enten ved å stille den inn i CoT, eller at den følger programmerte døgnplaner. Hvert soverom har en døgnplan for hver ukedag x 3. Ganger 3 fordi vi ønsker planer for tilfellene "Er hjemme", "Er ute", "Natt". Styringen av soverommets takvifte

og luftevindu er basert på utetemperaturen, hvor mye sol som stråler på den helt vestvendte veggen, og graden av skydekke. Først skal takvifta slås på og gå en periode, før luftevinduet overtar og står åpent en periode. Både vifte og luftevindu skal også kunne styres direkte ("manuelt") fra konsollet, og da med funksjonene starte/stoppe vifte og åpne/lukke luftevindu.

4. Ting å tenke over

Programvareutviklingsmetoder, altså hvordan man griper an et større utviklingsprosjekt - er et helt fagområde i seg selv, og utenfor rammene til dette emnet. Men gjennom undervisningen har vi likevel blitt presentert for flere sunne prinsipper som inngår i metodisk utvikling av programsystemer, slik som eksempelvis **modularisering**. Det innebærer at man splitter opp funksjonaliteten i flere selvstendige moduler, hvor hver modul utfører bare en del av den ønskede funksjonaliteten.

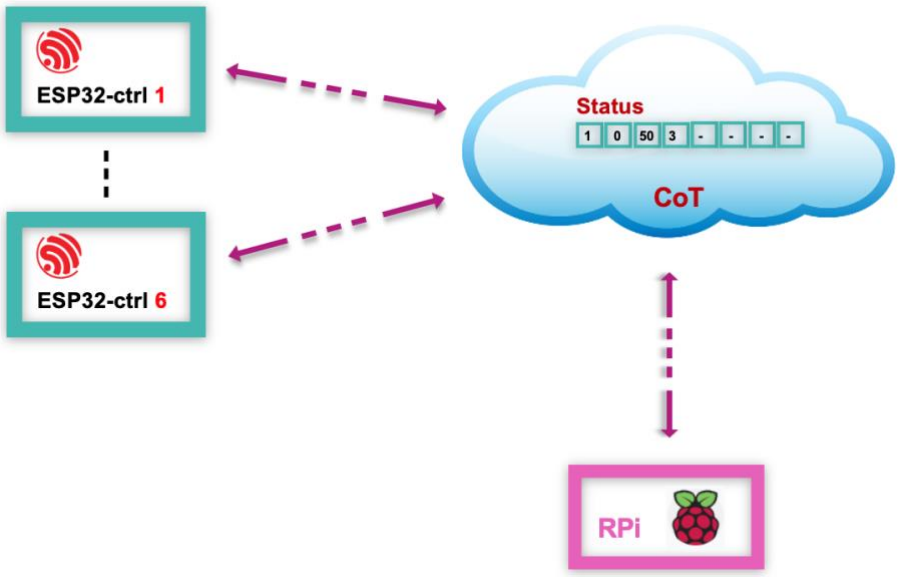
Modularisering og egendefinerte funksjoner henger sammen, selv om det bare er en del av konseptet i profesjonell programvareutvikling, men det vil være en grunnleggende del av den anbefalte tilnærmingen i dette prosjektet. Med andre ord, så anbefales en metodikk hvor den totale oppgaven dere står overfor splittes i mindre deler, som så igjen splittes i problemstillinger, og på ett eller flere nivåer i den strukturen så introduseres egendefinerte funksjoner.

Og som en hjelp i den oppdelingen vil vi i dette kapitlet beskrive/diskutere strukturen og aktuelle problemstillinger som igjen kan representere moduler, eller deler av moduler.

4.1. Status- og dataflyt – Vi har 3 hovedelementer i prosjektet: ESP32ene, Raspberry Pien, og CoT. Vi kommer raskt ut for problemstillingen: hva skal ligge på ESP32ene, og hva skal ligge på RPien? Og hvordan skal de kommunisere statusverdier.

La oss tenke litt generelt. Det vil uten tvil være behov for **dataflyt** ESP32 ↔ CoT og RPi ↔ CoT, og indirekte så dekker det også dataflyten ESP32 ↔ RPi – siden begge kan aksessere samme data i CoT. Når det gjelder **statusflyt** så vil det være behov for kommunikasjon både ESP32 ↔ ESP32 og ESP32 ↔ RPi.

Dette kan illustreres slik:

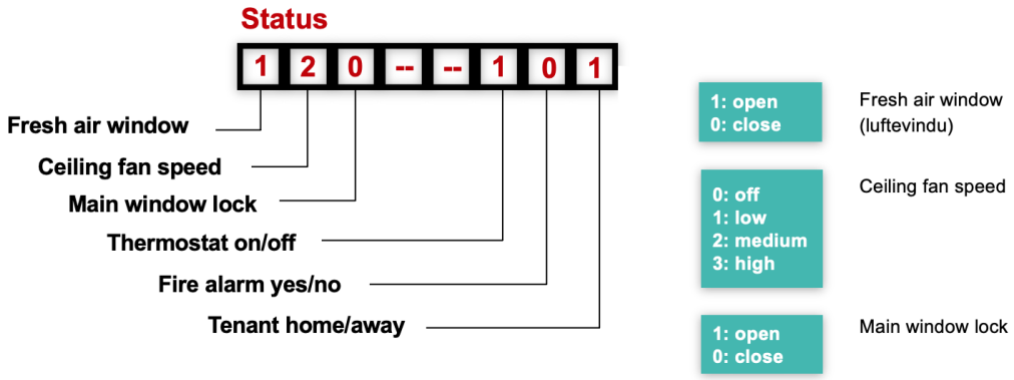


4.2. **Statusregister** – Vi kan tenke oss at enhetene kommuniserer statusverdier via en sentralt statusregister-modell. Dvs. at CoT holder ett eller flere statusregistre hvor hvert element representerer statusen til en ressurs eller et element i kollektivet.

Tenk på statusverdier som informasjon om tilstander som både ESP32ene og RPi må kjenne til – eksempelvis:

- Dørklokka er trykt. ESP32 vet det fordi den har ringeknappen tilkoblet. Avhengig av modellen for hvilket ansvar som plasseres hvor (se avsnitt 4.3 hvor dette diskuteres ytterligere), så må RPi også vite det dersom vi antar at den holder rede på hvor mange mennesker som er i kollektivet til enhver tid, hvor mange besøkende hver beboer eventuelt har, og hvor lenge de har vært der.

Figuren under viser hvordan et slikt statusregister i CoT kan være. Utvalgte statusverdier leses og oppdateres av RPien og ESP32ene, og på den måten holder enhetene hverandre oppdatert om viktige tilstander i kollektivet.



En hake ved denne modellen er at strukturen register ikke støttes av CoT, så vi må emulere registre om vi ønsker å benytte den. Å emulere betyr å få et system til å se ut som et annet. Så vi får det til å se ut som om vi har statusregistre i CoT, hvor både ESPene og RPien poller dem for nye verdier ved jevne mellomrom, og oppdaterer dem ved tilstandsendringer.

Vi vil komme tilbake til hvordan vi kan emulere statusregistre senere dersom det er behov for det, men kan hinte at det handler om å konvertere array til tall, og tall til array. Slik som i figuren over så har v i et array/register med elementene 1, 2, 0, 1, 0, 1 som vi kan tenke oss er implementert i RPien og ESP32ene, og når statusregistret skal representeres i CoT konverteres det til tallet 120101. Deretter tilbake til et array igjen når det leses i CoT av RPi eller en av ESP32ene.

4.3. **Sentralisert/Distribuert modell** - Man kan tenke seg to ytterligheter når det gjelder spørsmålet om hvem som skal kontrollere hva. En **sentralisert variant** innebærer at RPien styrer det aller meste, at «intelligensen» ligger i RPien, og ESP32ene rapporterer sensor- og statusverdier til RPien. Og når en ESP32 utfører andre oppgaver, så er det RPien som har tatt avgjørelsen og sendt den et kontrollsignal via CoT. Den andre ytterligheten er et **fullt distribuert system**, hvor alle enhetene tar avgjørelser og utfører oppgaver på et selvstendig, men likevel samhandlende grunnlag

Vi blir nødt til å gjøre valg som trekker i den ene eller andre retningen av disse to ytterlighetene, men en rent sentralisert eller distribuert modell er sannsynligvis ikke løsningen. Vi må finne en **hybrid,** og vi må være klar over hvilke konsekvenser hvert av disse valgene vil ha.

Et eksempel: Vi blir nødt til å administrere og regulere tilgangen til

kollektivets ressurser, slik som kjøkkenet. Bare 3 beboere av gangen kan benytte kjøkkenet samtidig, så vi må ha et bookingsystem. Vi kan tenke oss at booker du kjøkkenet så får du ha det i 20 minutter. I den **sentraliserte modellen** så holder RPien bookingreglene, og den holder tellingen på hvor mange som er på kjøkkenet til en hver tid. Så kan en ESP32 (beboer) spørre RPien via CoT om å få bruke kjøkkenet, og hvis RPien sier ja, så får den aktuelle ESP32 booke en av kjøkkenplassene i 20 minutter. RPien oppdaterer så sin kjøkkenoversikt. I en litt mer **distribuert modell** så kan vi tenke oss at det vedlikeholdes en teller i CoT over antall kjøkkenplasser som er tatt. ESP32ene vet at det maks kan være 3 på kjøkkenet samtidig, og sjekker derfor om telleren er < 3 når beboeren ønsker å booke kjøkkenplass. Dersom telleren er 0, 1, eller 2, så booker ESP32en en plass og inkrementerer telleren. Når de 20 minuttene er gått (eller at beboeren er ferdig før 20 minutter har gått, og «releaser» plassen), vil ESP32en dekrementere telleren. Alt helt uten innblanding fra RPien.

I vurderingen av hvor ansvaret for beslutninger skal legges, må vi ta hensyn til slike faktorer som:

> Trafikkmengde – hvilke konsekvenser får valget for data og statustrafikken i systemet
> Responstid – hva betyr det for hvor hurtig systemet kan respondere
> Fleksibiliteten – hva med framtidige utvidelser? Låser vi oss i en løsning som ikke tillater aktuell ny funksjonalitet?
> Minnestørrelsen – den er noe mindre på ESP32ene enn på RPIen
> Etc.

Et generelt råd er å tenke enkelt. Unngå intrikate løsninger. Tenk dessuten balansering av systemet – du ønsker å ha noe å gå på i alle retninger, slik at ikke systemet er presset på en eller flere ytelsesfaktorer allerede ved normal drift.

4.4. **Modellere og simulere** - Så har vi en problemstilling som er litt mer vrien: Hvordan skal vi representere livet i kollektivet? Beboerne sover der, våkner om morgenen, dusjer (energiforbruk), setter i gang vaskemaskinen (energiforbruk) fordi skittentøykurven er full, spiser frokost (må ha booket tid for å steke egg & bacon + må registrere energiforbruk + hvordan holde oversikt over hvor mange som har booket kjøkkenet og når?), setter inn i oppvaskmaskinen (når er den

full og må kjøres?), ordner seg og drar på campus, men kommer tilbake for lunsj, osv.

Vi må rett og slett lage oss modeller. Hver beboer må tenke over hva som er mønsteret for en normal uke, og kanskje for enkelhets skyld si at alle hverdagene er like, mens lørdag og søndag har hvert sitt mønster: når står du opp, når dusjer du, bruker du stekeplatene eller stekeovnen, osv. Alt som skjer i løpet av dagen(e) må vi representere i modellen.

Og vi skjønner at da handler det om å gjøre ting enkelt. F.eks. så kan vi si at en dusj koster alltid X kWh, bruk av stekeovnen koster alltid Y kWh, en kjøring av oppvaskmaskinen koster Z antall kWh, og må kjøres etter et antall frokoster, lunsjer eller middager, etc.

Hver enkelt gruppe får ta stilling til hvor omfattende de ønsker å gjøre modellene sine. Og hvorvidt modellene, samt simuleringen som baserer seg på modellene, kan det være fornuftig å samle i RPien som en overordnet oppgave, eller om de skal distribueres.

4.4.1. **Modellere og simulere energiforbruk** (primært RPi-relatert?)

Registrering av energiforbruket i kollektivet er en hovedfunksjonalitet. Vi ønsker å se på hvordan energiforbruket påvirkes av mønsteret hos den enkelte beboer, utetemperatur, vindstyrke, sol og skydekke, osv.
Vi er altså nødt til å lage oss enkle modeller for de forskjellige enhetene som trekker strøm i hybelhuset. Og hvordan værdata påvirker (primært) oppvarmingsbehovet i kollektivet.
Det er ikke svært viktig at det blir eksakte modeller, de må bare være noenlunde fornuftige. Eksempelvis så kan vi si at én kjøring av oppvaskmaskinen forbruker 0,8 kWh uansett

- Modellere forbruket (forslag):

> Dusj: fastverdi (kWh)

> Varmekabler på baderom: én kWh-verdi for normal bruk, én for natt-/dagsenking.

> Stekeplater og stekeovn: forenkles til én kWh-verdi for hver, uansett type bruk.

> Panelovner på soverommene: sett opp en enkel modell for kWh-forbruket avhengig av hva termostaten er satt til, samt utetemperatur og vindstyrke. Kanskje også sol og

skydekke, for gruppene som ønsker en ekstra utfordring. RPien henter inn værdata.

> Vaskemaskinen: anta at den kjøres X ganger i uka, og forbruker Y kWh. Videre at hver kjøring av vaskemaskinen følges av kjøring av tørketrommelen som forbruker Z kWh

> TVen i TV-stua: anta at den er slått på med et forbruk på X kWh så lenge TV-stua er booket.

> Oppvaskmaskinen: Én frokost fyller 10%, én lunsj 15% og én middag 25%. Forbruket er X kWh for hver kjøring. Den kjører når fyllingsgraden er > 90% og < 100%

- Modellere produksjonen:

> Solcellepanelet: produksjonen avhenger av solcelletypen (plukk et konkret panel og sjekk databladet), virkningsgraden, solens posisjon, skydekke, lufttemperatur, etc. RPien henter inn værdata.

4.4.2. Kommunisere statusverdier mellom ESP32 og RPi

-Booking av kjøkkentid. Skal RPi holde rede på (telle) hvor mange som har booket kjøkkentid, og så fortelle om det er ledig plass når en ESP32 ønsker å booke, eller kan ESP32ene fikse dette alene ved å sjekke et statusregister i CoT, og deretter oppdatere verdien både når beboeren som har ESP32en får kjøkkentid, og når den er over.

telle oppgi hvor mange beboere som har booket kjøkkenet nå (eller hvor mange plasser som er tilgjengelig på kjøkkenet nå, dersom vi ønsker å se på det på den måten. Eller også: er det noen ledige plasser på kjøkkenet nå? Altså tre måter å representere samme statusparameter på). ESP32 må vite det for å kunne booke kjøkkentid uten at det blir for mange som bruker det samtidig. RPi må vite det for å kunne oppdatere den statusen når noen er ferdig med kjøkkentida si, eller nye kommer inn. Men det avhenger av modellen vi velger: i noen varianter, så trenger kanskje ikke RPi være innblandet i delingen av en gitt status, fordi ESP32ene kan ta hånd om det selv, men i prinsippet så er det mer fornuftig at når alle ESP32ene må vite om, og bruker, en status, så skal RPi administrere «mekanismen» bak statusen. Skal det være mulig

å booke bare akkurat nå, og i 30 minutter, eller skal det være mulig å booke fram i tid (mer kompleks løsning)