IoT – Internetteknologier prosjekt

Gruppe 8

Daniel Pettersen

Shiwan Hassan

Eirik André Stålesen

Fra venstre: Shiwan, Eirik og Daniel

Sammendrag

Rapporten beskriver et IoT-prosjekt ved bruk av Arduino mikrokontroller med moduler. Gjennom en integrasjon av de teoretiske temaene, «IoT» og «smarte hjem» utføres det prototyper som tilfredsstiller brukerkrav. Kravene baserer seg på IoT og smarte hjem konsepter. Gjennom bruk av Arduino IDE og verktøy, par programmering og testdreven utvikling blir resultatet en implementasjon av de kartlagte brukerkravene. Implementasjonen fokuser på å sende informasjon til skyen, og styre lys gjennom en skytjeneste ved bruk av en app. Til slutt foreslås det videre utvikling i slike prosjekter, og forskning innenfor sikkerhet, da det er en meget relevant utfordring for IoT og smarte hjem.

Innholdsfortegnelse

Figur- og tabelliste

1.0 Introduksjon

Utviklingen av internetteknologier de siste årene har gjort det mulig å utforske og utvikle løsninger for smarte hjem gjennom et konsept som heter IoT (Internet of Things – tingenes internett). Denne rapporten har som mål å utforske noen muligheter som IoT byr på for smarte hjem. I følge Wang et al. er det forutsett at milliarder av fysiske ting eller objekter vil være utstyrt med ulike typer sensorer og aktuatorer, koblet til internett via heterogene tilgangsnettverk muliggjort av teknologier som innebygd «sensing», radio frekvens identifikasjon (RFID), trådløse sensor nettverk, sanntid og semantiske web-tjenester osv. (2013). Disse teknologiene kan brukes til smarte hjem implementasjoner, for å gjøre hverdagen til en vanlig bruker enklere eller mer praktisk.

Vi gjennomfører en utforskende rapport hvor vi kartlegger muligheter som IoT gir oss for smarte hjem løsninger ved bruk av en Arduino UNO mikrokontroller med tilbehør. Gjennom en analyse ved bruk av rike bilder og brukerhistorier kartlegger vi krav som gjør det mulig å utvikle prototyper basert på IoT og smarte hjem.

I rapporten utreder vi først hva smarte hjem og IoT er. Deretter forklarer vi metode, fulgt opp av analyse, design og implementasjon av prototypen. Til slutt legger vi til rette for videre forskning innenfor tema «IoT og smarte hjem» basert på det vi finner ut underveis i utviklingen av prototypen. Sentralt i prosjektet står følgende smarte hjem konseptene:

- hvordan en bruker kan styre lys uten å være fysisk tilstede i huset det gjelder.
- Bruke IoT i smarte hjem for å få nyttig informasjon fra sensorer.

Underveis i utviklingsprosessen vil flere tema innenfor IoT og smarte hjem berøres, men de overnevnte konseptene er utgangspunktet for brukerhistoriene og prototypene som er et resultat av dette.

2.0 Bakgrunn: smarte hjem, IoT og Arduino

I dette kapittelet forklarer vi vårt teoretisk utgangspunkt for prosjektet. Det er konseptet om hva smarte hjem og IoT er. Først snakker vi om de to konseptene hver for seg, og foreslår en integrasjon mellom disse to. Til slutt diskuterer vi også valg av Arduino UNO som hovedverktøy for å implementere IoT løsninger til smarte hjem.

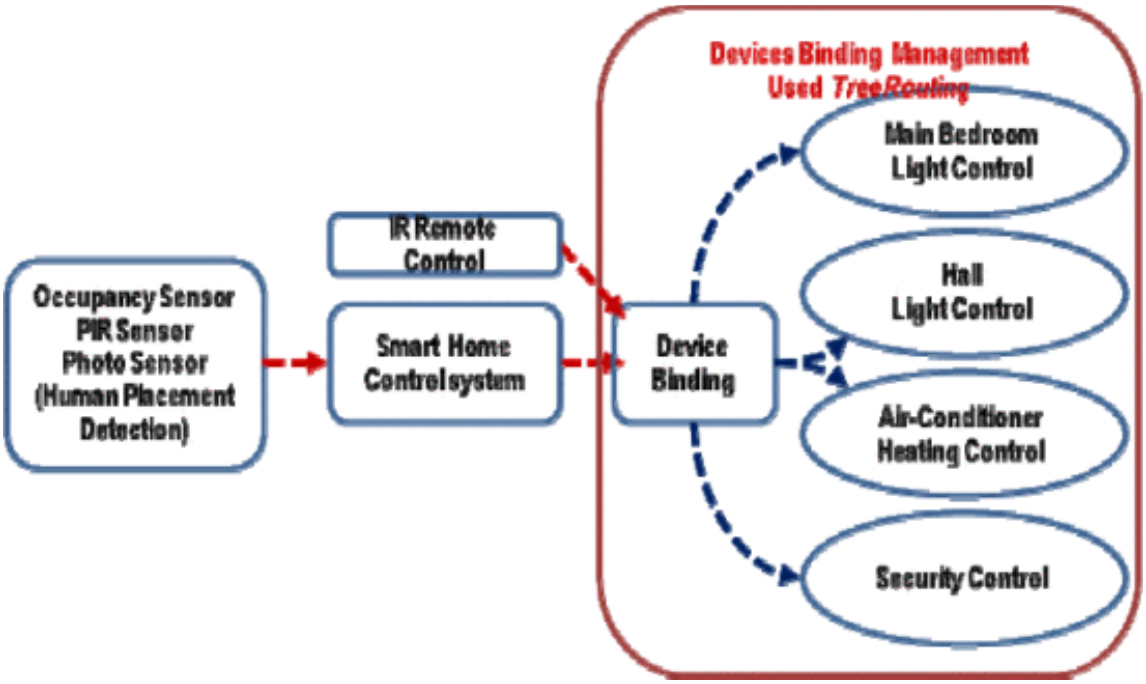
2.1 Hva er smarte hjem?

Smarte hjem er et konsept som går ut på at hus blir automatisert, og kunne fjernstyres på et eller annet vis (gjennom f. eks. PC, smarttelefon osv.) (Om Smarthus - Hva er Smart Hus?, u.d.). Enheter dette kan omfatte som regnes som en del av et hus kan være:

- Lys
- Varmepumpe
- Alarmsystem
- Andre elektriske apparater (f. eks kaffetrakter)

Basert på brukeren ønske kan et smart hus skru lyset av og på når man går inn eller ut av rommet via sensorer, eller via en knapp på mobilen som skrur alt lyset i et ønskelig rom på. Lysene kan også regulere med dagslys, slik at på dagen med høy naturlig lys, vil lysene stille seg til svak styrke/av for å spare strøm, og gradvis øke styrke/skru seg på etterhvert som det blir mørkere.

Tradisjonelt sett har smarte hjem blitt bygget rundt «ZigBee» kommunikasjonsteknologi og hjemmenettverk. Han og Lim (2010) utreder i sin artikkel om et smart hjem energistyringssystem ved bruk av ZigBee nettverk for å samordne flere ulike typer smarte hjem enheter. De oppsummerer denne måten å bygge smart hjem på med følgende modell.



Figur 1: Smart hjem kontrollsystem (Han & Lim, 2010)

Figur 1 viser sensorer som er koblet til et styringssystem (integrasjon) som kan styres med en fjernkontroll. «Device Binding» konseptet kobler alle disse enhetene sammen og gjør det mulig å styre de gjennom et og samme system. Systemene som er koblet sammen er lys-, varme- og sikkerhetsstyringssystemer, og forfatterne foreslår bedre muligheter for strømsparing (effektivisering) og overvåking i hjemmet, blant annet.

Begrensingen med slike smarte hjem løsninger er at de kun kan kontrolleres over det lokale hjemmenettverket, og kun styres ved bruk av en fjernkontroll (eller lignende enheter). Utredningen til Han & Lim viser styrker ved kontroll og integrasjon av alle smarte enheter, men er begrenset på grunn av bruken av IEEE 802.15.4 og ZigBee kommunikasjonsteknologier.

2.2 Hva er IoT?

Internet of Things (IoT, eller «tingenes internett» på norsk) handler om å kunne koble diverse enheter opp mot internett. Disse tingene skal også kunne samhandle seg i mellom med lite input fra mennesker. Dette tillater enheter å oppfatte og samle data fra verden rundt oss, og sende data over internett, hvor dette kan benyttes til å oppnå diverse mål. Giusto et al. (2010) definerer konseptet «IoT» på følgende måte:

> *"The basic idea of this concept is the pervasive presence around us of a variety of things or objects – such as Radio-Frequency IDentification (RFID) tags, sensors, actuators, mobile phones, etc. – which, through unique addressing schemes, are able to interact with each other and cooperate with their neighbors to reach common goals."*

For å få en forståelse av dette kan vi se for oss to ulike scenarioer:

- Du våkner opp en morgen med en smartklokke, som forteller kaffemaskinen at personen har våknet, som starter å trakte en kopp kaffe. Videre går du inn på baderommet, og stiller deg på badevekten, som måler vekt og progresjon over en hvis periode for å se om du har gått ned eller økt vekten. Dataen blir sendt til både klokken for å oppdatere brukeren samt til kjøleskapet, som gir en anbefaling på hva du bør spise.
- Hvis noe mangler i kjøleskapet (lite/tomt for melk) vil kjøleskapet gi beskjed til klokken/mobilen om at disse varene er tomt og at du må kjøpe mer. Kjøleskapet kan hente informasjon fra internett på hvor nærmeste butikk er, og eventuelt hvilken

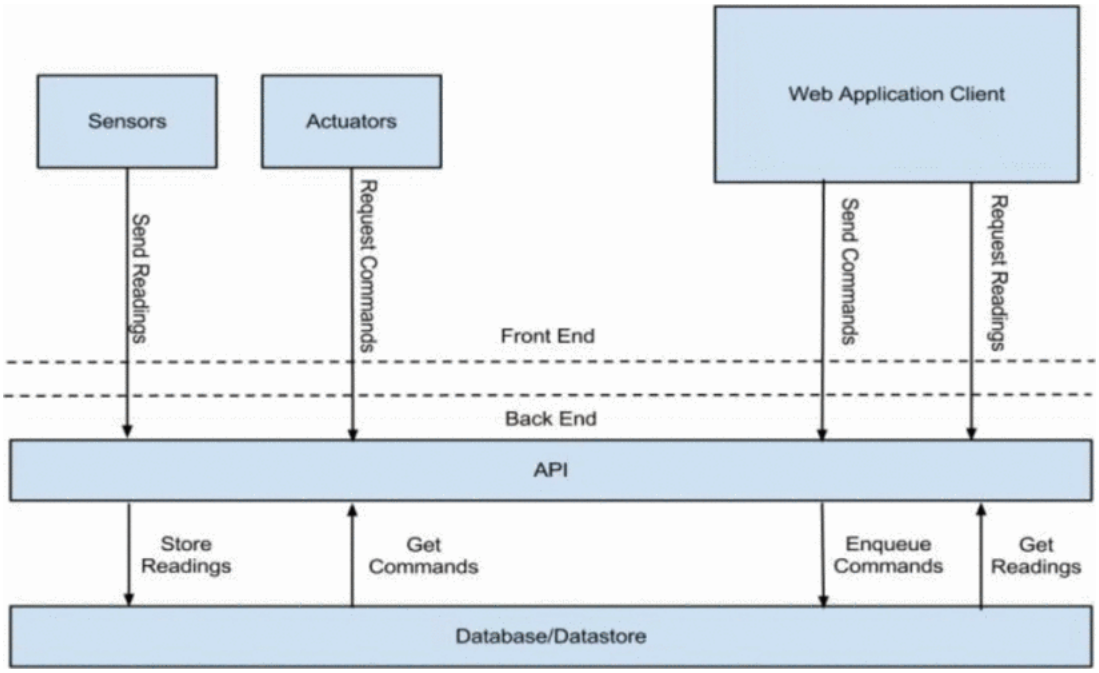
butikk har billigst melk. I tillegg kan den beregne hva det vil koste deg å kjøre til butikken, og hvor mye tid det vil ta å kjøre gjennom trafikken. Denne dataen kan sendes automatisk til din telefon for en sømløs opplevelse.

IoT gir oss et tettere bånd mellom oss selv og internett. Dette vil føre til at mer og mer av livet vårt, og våre vaner vil komme på nettet, som kan brukes da brukes til positive, men også negative formål.

2.3 Integrasjon mellom IoT og smarte hjem

Det er mulig å bygge videre på konseptet «smarte hjem» med IoT. Huset kan kobles til et nettverk og kontrolleres over internett, enten automatisk, fjernkontroll eller via en app. Små mikrokontrollere eller datamaskiner koblet til vanlige enheter i hjemmet tillater styring og kontroll med *nettverksmuligheter*. Det skal være mulig å kunne gjøre alt automatisering og kontroll i smarte hjem slik beskrevet over, men integrasjon med IoT og smarte hjem muliggjør alt dette over internett.

Soliman et al. (2014) foreslår en systemarkitektur for smarte hjem. Denne arkitekturen gjør det mulig at hver enhet koblet til internett kan ansees som en IoT enhet, og kan dermed også styres ved bruk av webteknologier.



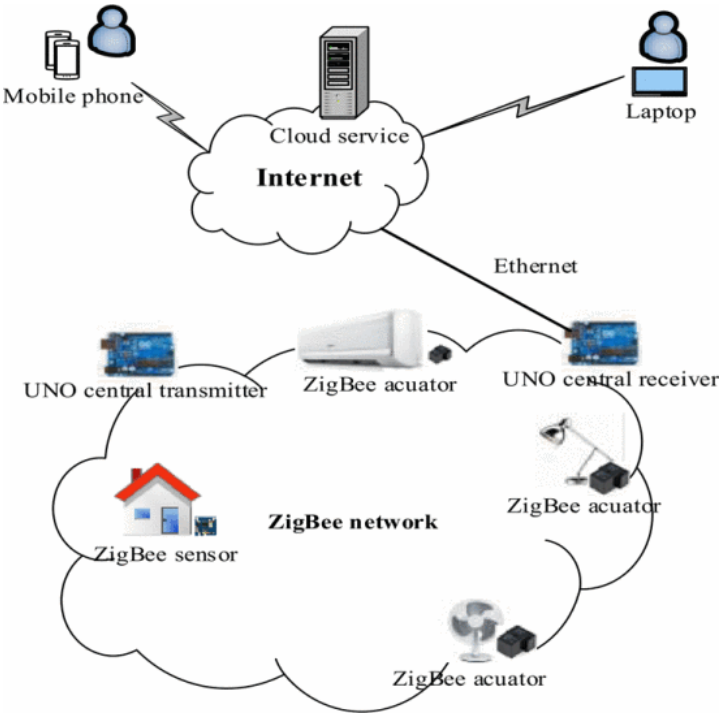
Figur 2: System architecture for Smart Home (Soliman et al. 2014)

Arkitekturen i Figur 2 oppsummeres med følgende punkter:

- Mikrokontroller-aktivert sensorer: måler hjemmetilstand; mikrokontrolleren tolker og prosesserer den instrumenterte dataen.
- Mikrokontroller-aktivert aktuatorer: mottar kommandoer overført av mikrokontrolleren for å gjennomføre bestemte handlinger. Kommandoen er utstedt basert på interaksjonen mellom mikrokontrolleren og skytjenester.
- Database / Data Store: lagrer data fra mikrokontroller-aktiverte sensorer og skytjenester til data analyse og visualisering, og fungerer som en kommando kø som er sent også til aktuatorene.
- Server- / API-laget mellom «back end» og «front end»: muliggjøre prosessering av data mottatt fra sensorene og lagring av dataen i databasen. Den mottar også kommandoer fra webapplikasjonsklienten for å styre aktuatorene og lagrer kommandoene i en database. Aktuatorene lager forespørsler for å konsumere kommandoene i databasen gjennom serveren.
- Webapplikasjon som skytjeneste: tillater å måle og visualisere sensor data, og kontrollere enheter gjennom en mobil enhet (e.g. smarttelefon).

(Soliman, Abiodun, Hamouda, Zhou, & Lung, 2014).

Integrasjon mellom IoT og smarte hjem kan oppsummeres slik i følgende modell:



Figur 3: ZigBee home network (Soliman et al. 2014)

Figur 3 på forrige side viser at internett (med en skytjeneste i bunn) tillater å styre et eksiterende ZigBee nettverk for smarte hjem. I tillegg viser implementasjonen over at Arduino UNO mikrokontroller brukes med Ethernet-protokollen som mottaker av internettilkoblingen.

Utviklingen av smarte hjem med IoT byr på veldig mange nye muligheter. IoT vokser i et voldsomt tempo, og på dagens TV og reklamer ser vi en økning av temaene smarte hjem og IoT. I følge Intel hadde vi 2 milliarder IoT enheter i 2006, mens i løpet av 2020 vil det finnes over 200 milliarder enheter. Det vil bety at i 2020 vil det være omtrent 26 IoT-enheter for hvert menneske på jorden (Intel, u.d.). Dette gjør det desto mer relevant å studere hvilke muligheter som finnes med IoT, og hvordan det er mulig å løse eventuelle utfordringer.

2.4 Arduino som verktøy for IoT

IoT gjør databehandling (computing) og kommunikasjon altgjennomtrengende både i mobil og «wearable»[1]. Dette er takket være utviklingen innenfor mikrokontrollere og mikrokontrollerplattformer. Arduino er et eksempel av mikrokontroller og plattform. Det består av et programmeringsmiljø og Arduino kretskort; programmeringsmiljøet muliggjør å håndtere, kompilere, laste opp og simulere programmer. De ulike mikrokontrollerne kommer i mange ulike størrelser og spesifikasjoner. Spesifikasjonene avgjør hastighet og funksjonalitet. Det essensielle på mange kretskort er «pins» (på norsk: pinner) som gjør det mulig å koble til eksterne enheter for å sende og motta elektriske signaler / data (Soliman, Abiodun, Hamouda, Zhou, & Lung, 2014).

Vi valgte å bruke Arduino da det er en billig plattform å utvikle på. I tillegg er den «open source» (åpen kildekode), kryssplattform og fungerer bra til å lage prototyper (Arduino, 2018). Fleksibiliteten som den åpne naturen av Arduino gir oss er stor; moduler finnes i rekke og rad for å utvide funksjonaliteten til kretskortet, og mye kan gjøres innenfor elektronikk for å få den ønskede løsningen. Vi får også utfordre oss selv da det skal en del

[1] En «wearable» enhet er en enhet som kan festes på kroppen og utfører visse funksjoner, f. eks. en smartklokke eller treningsbånd som kan loggføre aktivitetsdata og overføre det til en smartphone app som lagrer dataen i skyen. Eksempler på disse kan være Apple Watch eller Fitbit. Vanligvis har «wearables en eller annen trådløs kommunikasjons- / overføringsprotokoll som Bluetooth eller Wi-Fi.

konfigurering til for å få Wi-Fi til å fungere, i motsetning til andre plattformer som gjerne benytter seg av hele operativsystem og drivere for å gjøre prosessen enklere (på bekostning av tilpasningsmuligheter).

Vi slipper også overhead fra et operativsystem, og trenger kun da å forholde oss til implementasjonen av selve funksjonaliteten, og dermed fjerner visse feilpunkt som kan oppstå ved bruk av et OS. Arduino programmeres i C++, et språk som ansees til å være noe lavere nivå enn nyere språk som Java, C# eller Ruby, for å nevne noen (Wikibooks, 2018). Å programmere slik gir en følelse av å jobbe tettere mot «bare metal» (selve metalen), og har en positiv læringseffekt i forhold til å bygge egne implementasjoner tilpasset etter behov.

3.0 Metode

Videre diskuterer vi metoder vi brukte gjennom prosjektet. Dette omfatter metoder for kartleggingen av krav til prototypen, implementasjons- og kvalitetssikringsmetoder. Til slutt integreres metoden i en modell.

3.1 Kartlegging av krav

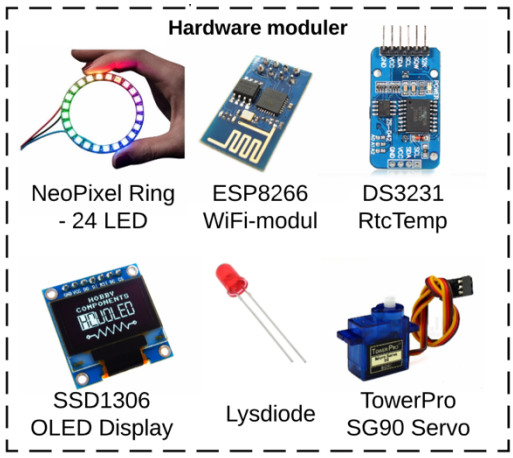
For å kartlegge kravene så har vi basert oss på noen oppdiktede hendelsesforløp tegnet i rike bilder. Vi har ikke tatt utgangspunkt i intervjuer eller innsamling av data, da prosjektet er av mindre omfang, og har som mål å dekke kun noen få grunnleggende konsepter innenfor IoT og smarte hjem. I den sammenhengen har vi valgt å ta på oss «brukerhatten» og tenke ut relevante hendelsesforløp for en bruker opptatt av å bruke IoT for å ha et «smartere hjem». Da prosjektet kun er av natur «prototype», vil det også være mer relevant å ha en analysefase senere etter å ha gjennomført noen funksjonelle tester med brukere, for å se om prototypene er levedyktige (brukervennlige og meningsfulle).

En del av kartleggingen inkluderer også å beskrive brukerhistorier. Brukerhistoriene tillater å forstå en brukers perspektiv på ønsket funksjonalitet, og begrenser samtidig hvilken funksjonalitet vi kommer til å utvikle i implementasjonen.

3.2 Verktøy

Prosjektet bør ha en IDE for å utvikle mot en mikrokontroller. Da vi landet på Arduino UNO-kompatibel mikrokontroller som tilrettelegger for IoT, måtte vi da forholde til Arduino IDE og ulike biblioteker for Arduino. I tillegg har vi ulike hardwaremoduler som brukes i prototypen.

Figur 4: Arduino IDE.



Figur 5: Hardware moduler brukt i prosjektet.

Vi anskaffet en pakke med flere hardware-moduler og en instruksbok som da gjorde det mulig for oss å eksperimentere med veldig mange prototypeimplementasjoner.

GitHub er hovedverktøyet for å lagre koden som vi utvikler i løpet av prosjektet. Koden og sin funksjonalitet blir også dokumentert. Vi bruker ikke GitHub til samarbeid, da fysisk testing av hardware er nødvendig underveis i prosjektet, og da er det upraktisk å laste ned om og om igjen biblioteker og kode på flere maskiner. Dermed baserer vi oss i større grad på parprogrammering, en utviklingsmetode vi forklarer i neste delkapittel.

3.3 Utviklingsmetode

Vi valgte par programmering som hovedmetode for utvikling. Av praktiske årsaker er det enklere at en programmerer på en maskin for å så kunne teste umiddelbart på hardware. Par programmering har imidlertid noen andre fordeler som vi mener er verdifulle for å få prototypen fungerende på kort tid og med høy kvalitet.

William et al. skriver at to resultater er antatt å forventes ved bruke av par programmering: (1) gjøre utvikling raskere og (2) forbedre software kvalitet (2000).

Resultatene kartlagt viser en redusering i antall nødvendige kodetimer, flere «test cases» fullført og bedre problemløsning, noe som øker kvaliteten på det endelige produktet.

Table 1

Percentage of Test Cases Passed*

	Individuals	Pairs
Program 1	73.4	86.4
Program 2	78.1	88.6
Program 3	70.4	87.1
Program 4	78.1	94.4

*The difference in quality levels is statistically significant to $p < 0.01$; p is the probability that these results could occur by chance.

Figur 6: Resultater fra Williams et al. (2010) - en økning i test cases gjennomført.

3.4 Kvalitetssikring (QA) med Test Driven Development (TDD)

I følge Janzen & Saiedian er «Test Driven Development (TDD)» (på norsk: testdrevet utvikling) en strategi som krever å skrive automatiserte tester før det utvikles funksjonell kode, i små, hurtige iterasjoner (2005). De oppsummerer i følgende tabell fordelene ved bruk av TDD basert på forskning:

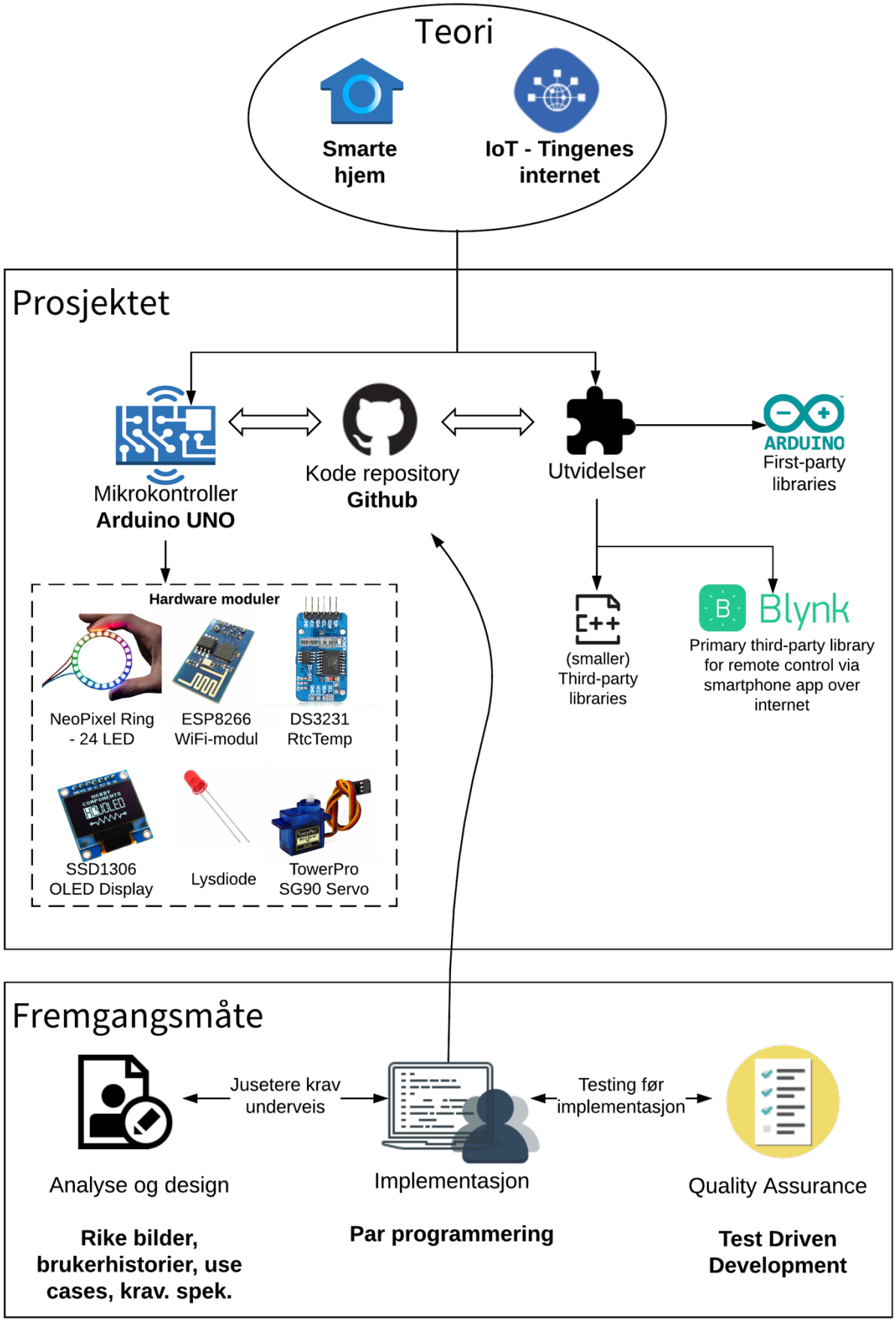
Table 2. Summary of TDD research in academia.

Controlled experiment	Number of programmers	Quality effects	Productivity effects
Kaufmann[11]	8	Improved information flow	50% improvement
Edwards[12]	59	54% fewer defects	n/a
Erdogmus[13]	35	No change	Improved productivity
Müller[14]	19	No change, but better reuse	No change
Pančur[15]	38	No change	No change

Figur 7: Oppsummering av TDD forskning i akademia (Saiedian & Janzen, 2005).

Blant annet viser det seg å være en økning i kvalitet og produktivitet. I vårt prosjekt er TDD ideelt da vi hele tiden må forsøke å bekrefte om hardware fungerer på den riktige måten. Et eksempel som vi bruker er å teste at Wi-Fi tilkobling er etablert før Blynk bibliotekene setter i gang. Dette tillater oss å fjerne mulige feil så tidlig som mulig. Vi unngår også at programmene blir «feature creeps» - at veldig mange funksjoner blir implementert, men fungerer ikke optimalt.

På neste side (Figur 8) har vi en modell som er en oppsummering og integrasjon av hele prosjektmetodikken. Dette vil gi en helhetlig forståelse for prosjektet.



Figur 8: Integrasjon av teori, prosjekt og fremgangsmåte.

4.0 Analyse og design

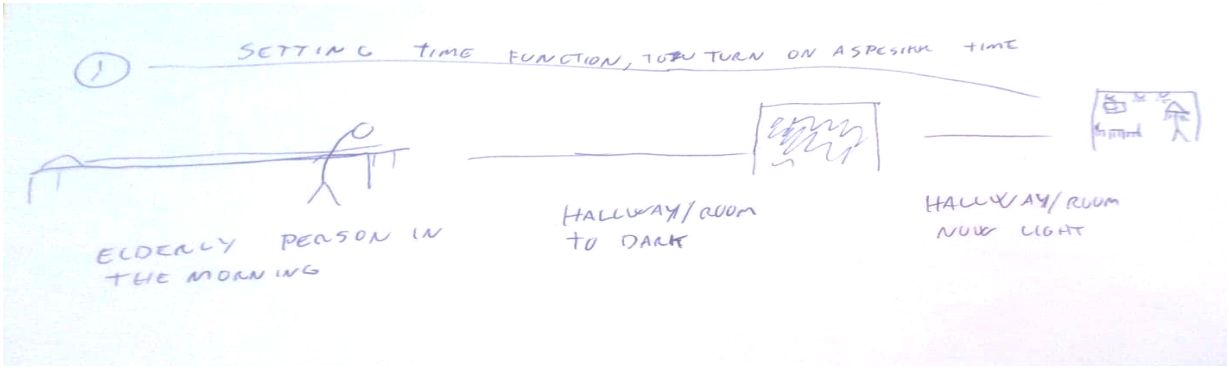
Analysen er basert på noen «oppdiktede» historier som vi ser oss vanlige brukere vil gjennomføre. Historiene i visuelt format (rike bilder) er basert på grunnleggende konsepter om smarte hjem og IoT diskutert tidligere i rapporten (Kapittel 2.0 Bakgrunn: smarte hjem, IoT og Arduino). Dette er det første steget for å forstå hvordan en bruker vil jobbe seg gjennom aktuelle utfordringer som er mulig å løse med IoT i hjemmet. Etter analysen utformer vi en kravspesifikasjonstabell basert på brukerhistoriene, og designer deretter bruksscenarioer. Dette vil gi oss nok informasjon til å implementere løsningen.

4.1 Rike bilder – første steget i analysen

Det rike bildet under (Figur 9) viser hvordan huseieren lurer på om vedkommende har slått av lyset hjemme eller ikke. Huseieren har muligheten til å sjekke og slå av lysene på telefonen gjennom internett. Arduino er utvalgt som mikrokontroller i eksempelet, men hvilken som helst vilkårlig mikrokontroller kan velges i analysen. Tekniske avgjørelser tas senere i prosessen.

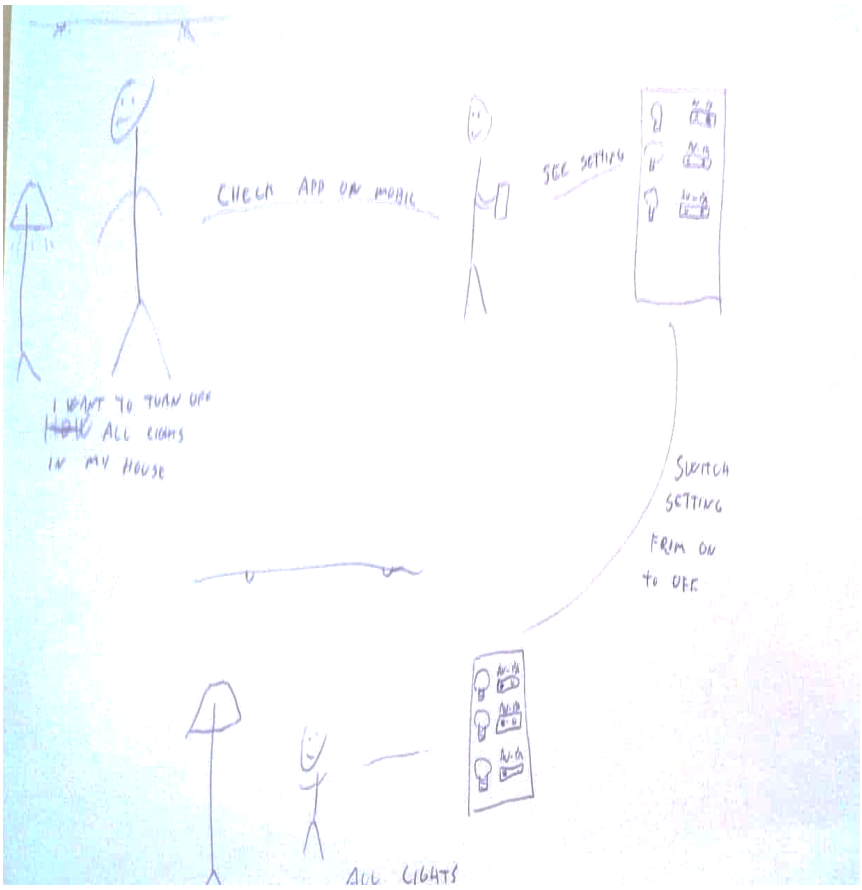
Figur 9: En bruker styrer lys gjennom app.

Det neste rike bildet (Figur 10) viser en tidsfunksjon, hvor en eldre person står opp på morgenen. Gjennom appen er det allerede satt at på et visst tidspunkt på morgenen skal lysene slås på. Dette kan i den reelle verden stemme med sovesyklysen til personen, eller tabletter / mat som må tas på et visst tidspunkt.



Figur 10: En eldre person bruker tidsfunksjoner for å automatisere lys når det er behov for det.

Det siste rike bildet (Figur 11) viser en huseier som kan styre flere lys og slå av og på lys i ulike rom. Kjernen i den siste analysen er muligheten til å kunne styre flere lys ved bare bruk av et verktøy. Vi tolker dette verktøyet som en smarttelefon.

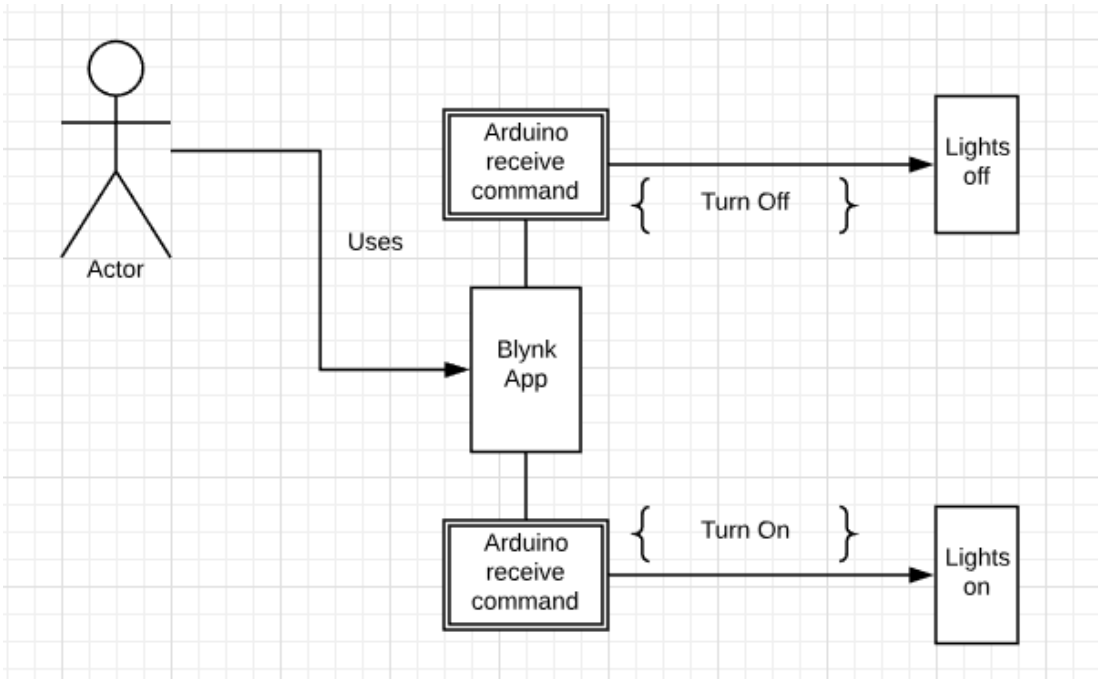


Figur 11: En huseier styrer flere lys fra en app.

4.2 Bruksscenarioer – fra analyse til design

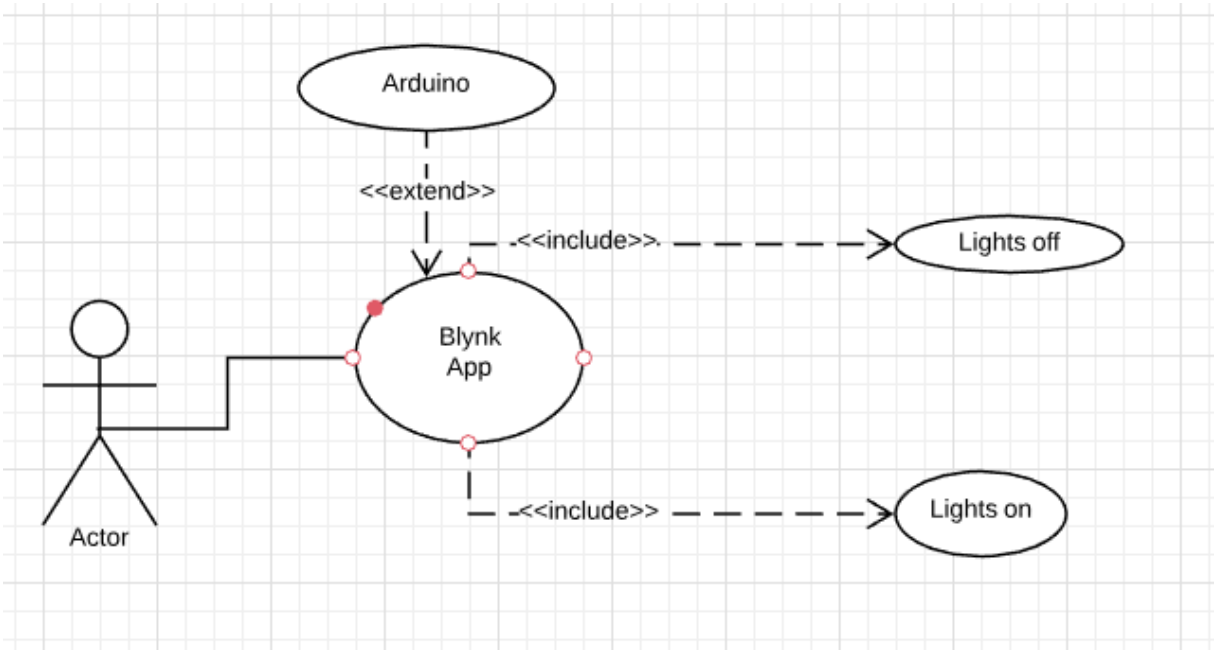
Bruksscenarioer fungerer i vårt prosjekt som broen fra analysen til design – hvordan brukeren vil interagere med ulike lag i implementasjonen. Her tas noen tekniske antakelser, som deretter kan integreres i kravspesifikasjonen i neste delkapittel. Bruksscenarioene er modellert i UML, med korte beskrivelser i bildeteksten.

Brukeren slår av og på lys:



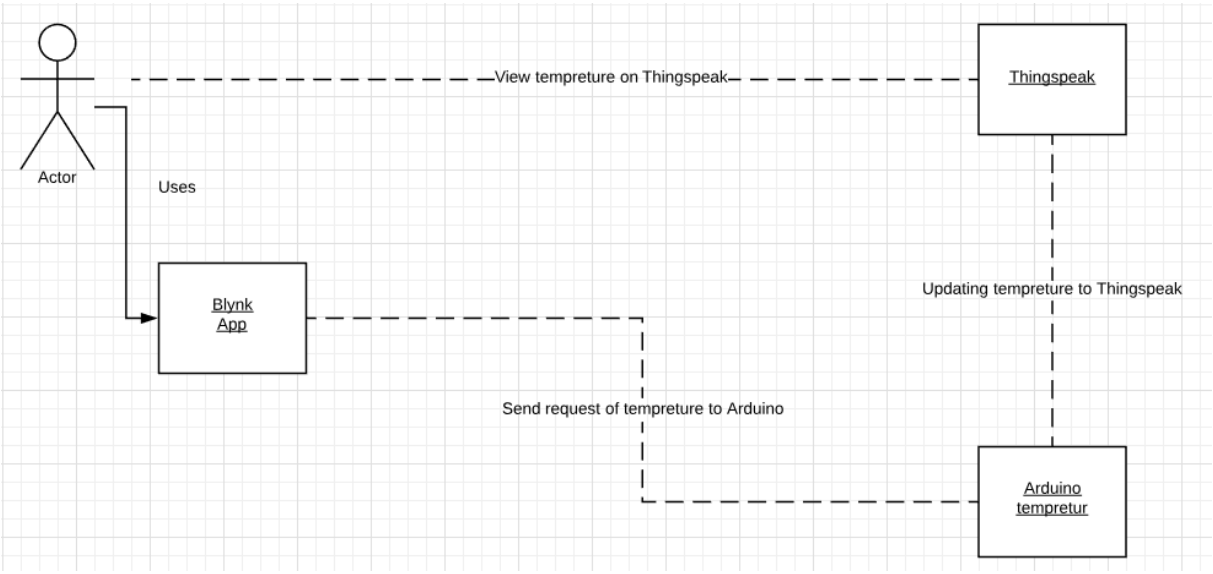
Figur 12: Sett fra brukerens perspektiv, hvordan brukeren forholder seg til Arduino og Blynk for å styre lys.

Brukeren slår av og på lys:



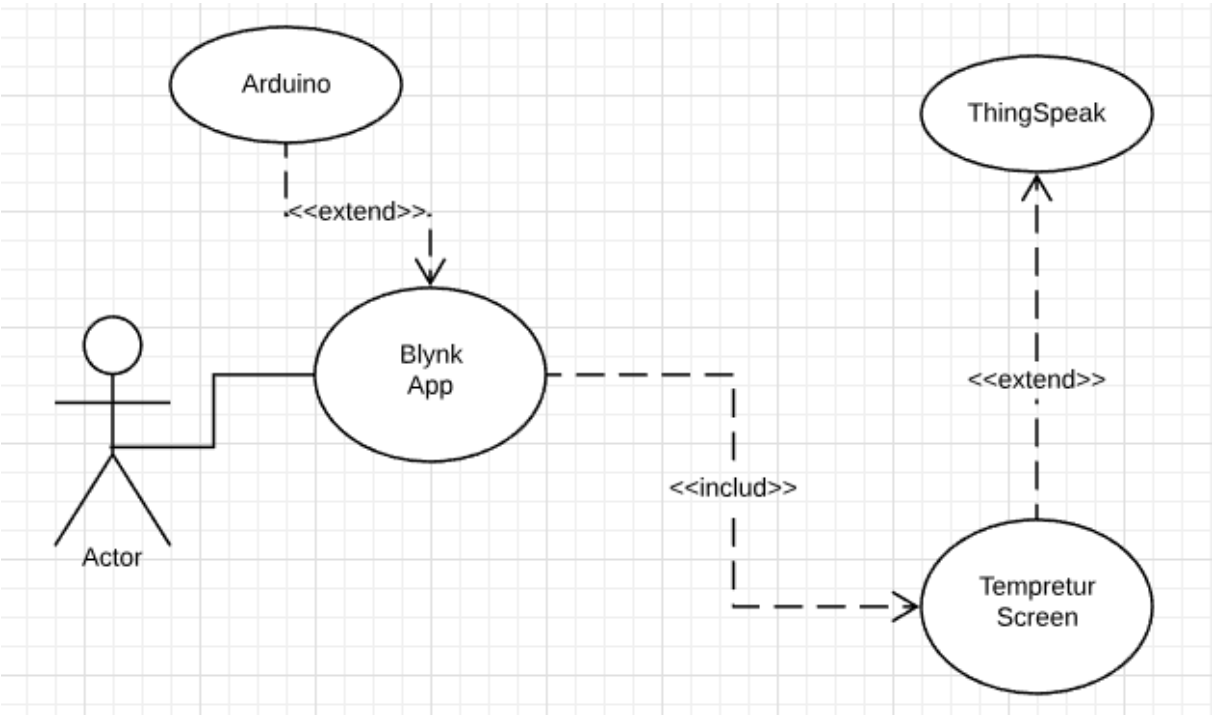
Figur 13: Sett fra systemets perspektiv, hvordan teknologiene samhandler for å styre lysene.

Bruker ser temperatur på ThingSpeak og bruker app for å motta lesing:



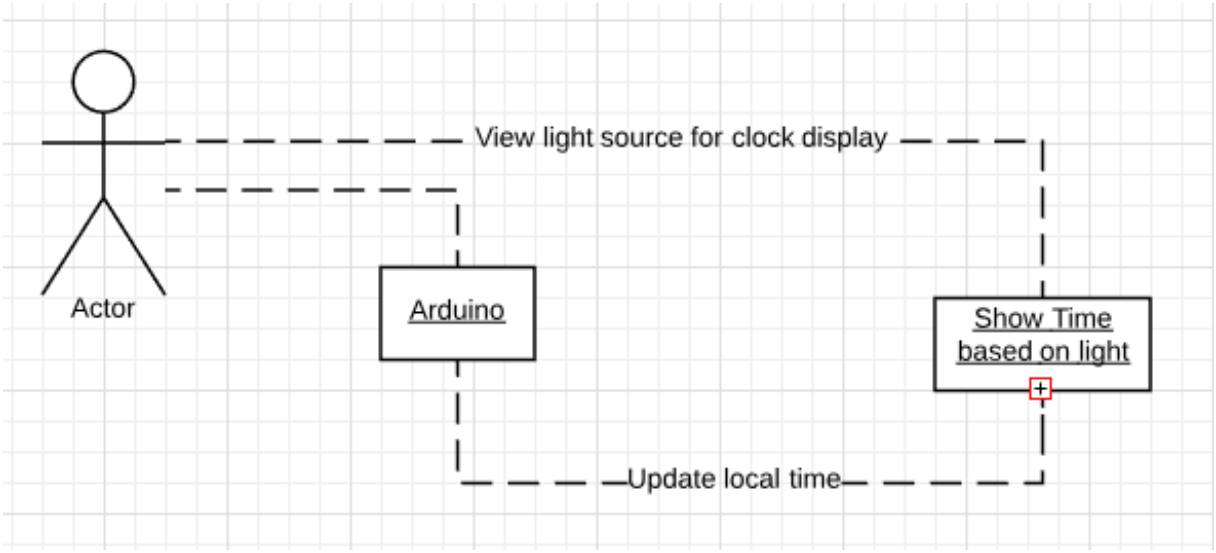
Figur 14: Sett fra brukerens perspektiv, hvor brukeren gjerne ønsker å ha en app for å se temperatur.

Bruker ser temperatur på ThingSpeak og bruker app for å motta lesing:



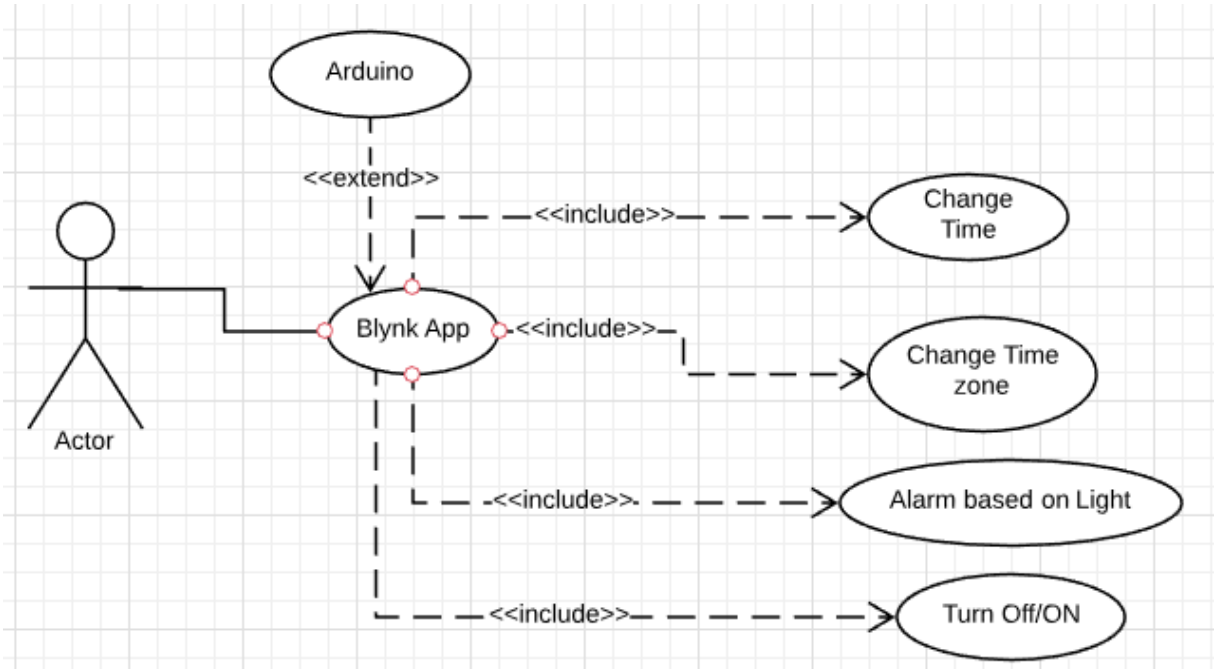
Figur 15: Sett fra systemets perspektiv, samhandlingen med Blynk for å oppdatere Thingspeak.

Vise og holde tid på Arduino:



Figur 16: Sett fra brukerens perspektiv, hvor brukeren kan se tid på skjerm og lys.

Vise og holde tid på Arduino:



Figur 17: Sett fra systemets perspektiv, hvor Blynk gir muligheter for å endre tid, tidssone, sette lysbasert alarm, og slå av og på lys.

Disse scenarioene gir oss et større grunnlag for implementasjon, men da implementasjonen utvikles iterativt så vil det vise seg at alle integrasjonene vil ikke være på plass i første omgang. Da er vi tilbake til at de mest grunnleggende kravene i kravspesifikasjonstabellen oppfylles i neste delkapittel.

4.3 Brukerhistorier og funksjonelle krav

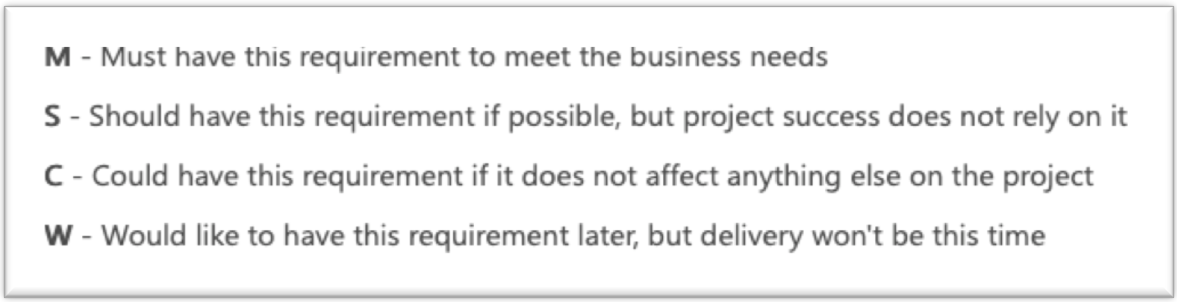
Etter analysen av de rike bildene og design av bruksscenarioer er det mulig å lage brukerhistorier for å utvide forståelsen av den etterspurte funksjonaliteten. Det vanligste er å ha brukere involvert i prosessen, men da dette ikke har vært mulig på grunn av tidsbegrensninger og omfang av prosjektet, så har vi heller valgt å lage noen historier selv, sett fra ulike typer brukerperspektiv. Vi som utviklere setter oss i skoene til en bruker som ønsker en viss funksjonalitet. Brukerhistoriene tar utgangspunkt i noen de rike bildene, og vises i andre kolonne i Tabell 1 (neste side).

Disse brukerhistoriene kan da tolkes over til tekniske krav (Tabell 1, tredje kolonne), som gir oss som utviklere verdifull informasjon på hvilken konkret funksjonalitet er etterspurt, og hvordan det kan løses teknisk.

Til slutt inkluderer vi en prioritering i MoSCoW-formatet. MoSCoW vil kunne si oss noe om hvor viktig hvert krav er å implementere. De viktigste må-kravene (Must) må oppfylles for at vårt produkt blir fullverdig. «Must»-kravene kan forstås som de mest essensielle for at produktet leverer det som er etterspurt. Kravene med lavere rangering i MoSCoW er mindre essensielle:

- Should have (bør ha for å oppfylle den ønskede funksjonaliteten).
- Could have (fint å ha, men ikke nødvendig for å oppfylle grunnfunksjonaliteten).
- Won't have (skal ikke inkluderes i det heletatt; irrelevant krav for produktet).

Duncan Haughey (2018) oppsummerer MoSCoW på følgende måte (med unntak for W-prioriteringen, som fortsatt ansees som relevant etter Project Smart sin tolkning):



Figur 18: MoSCoW-prioritering i følge Project Smart.

Tabell 1: Brukerhistorie og kravspesifikasjonstabell, med MoSCoW prioritering

Nr.	Brukerhistorie	Krav	Prioritet
1	Som en huseier, vil jeg kunne styre lysene hjemme selv når jeg er borte.	Implementere internettilkobling og kontroll funksjoner for lysene ved bruk av Arduino.	Must
2	Som en huseier, ønsker jeg å bruke min telefon eller datamaskin for å styre lysene.	Implementere en webapplikasjon som kan bli «containerized» til en mobilapplikasjon eller tilgjengelig gjennom nettleseren.	Should
3	Som en huseier, ønsker jeg at løsningen skal være brukervennlig.	Bruke webstandarder for å forholde seg til brukervennlighetsstandarder (f. eks WAI).	Should
4	Som en huseier, ønsker jeg å styre flere ting fra mine enheter, for å spare tid.	Implementere mulighet for å styre flere ting gjennom appen.	Should
5	Som en eldre person, ønsker jeg timers på lys / enheter, for å hjelpe meg å slå de av når jeg glemmer.	Implementere tidsstyringsverktøy.	Could

Grunnen til at vi ikke inkluderer akseptansekriterier er fordi vår evaluering av rangering på MoSCoW går ut fra at «Must»-kravene må oppfylles for å oppnå et relevant IoT – smart hjem produkt / prototype. Akseptansetester i forhold til brukeren er også litt utenfor domenet da det er kun vi utviklere som er brukere, men dette løser vi ved å benytte oss av Test Driven Development (TDD), hvor vi tester før vi utvikler funksjonalitet.

4.4 Ikke-funksjonelle krav

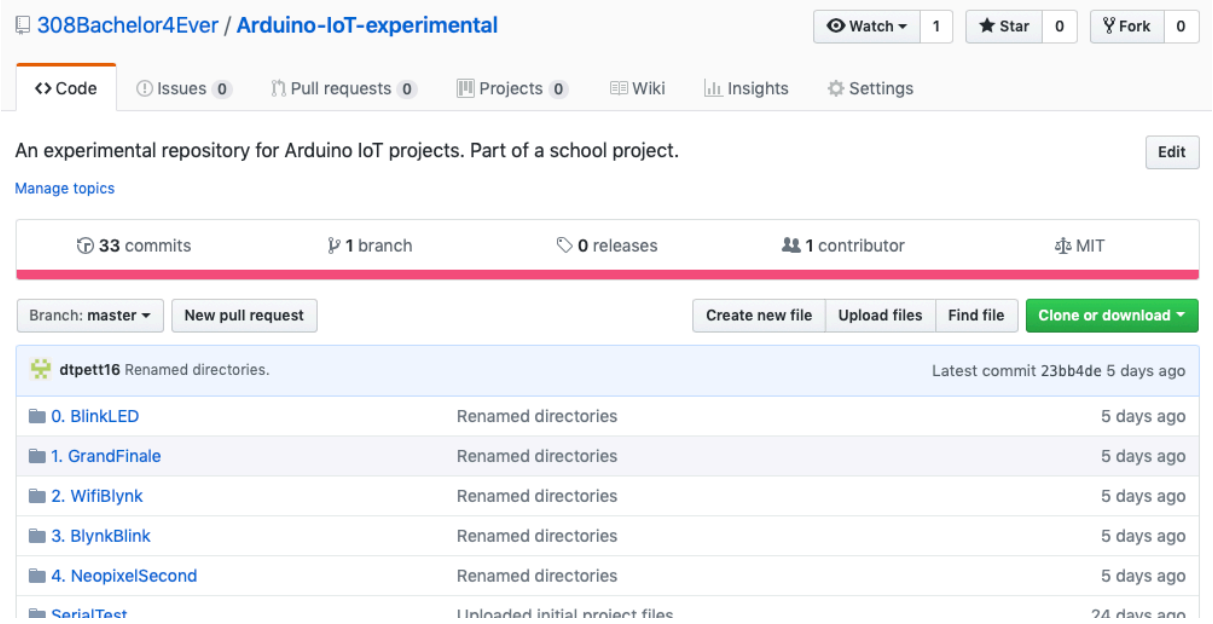
De ikke-funksjonelle kravene er i stor grad tekniske avgjørelser som skal ikke påvirke funksjonaliteten oppfattet av brukeren, men som påvirker det teknologiske rammeverket. Her har vi satt noen kriterier vi mener er relevant for prototypen.

Nr.	Krav	Prioritet
6	Bruke et rammeverk laget for Arduino som muliggjør styring av Arduino på flere plattformer.	Must
7	Bruk React Native (eller annet fleksibelt rammeverk) for å implementere en kryssplattform løsning.	Should
8	Explore the possibilities of containerizing the application for easy deployment on cloud solutions. Utforske mulighetene for å «containerize» applikasjoner for enkel distribusjon i skyløsninger.	Could

5.0 Implementasjon

Resultatet av analyse og design prosessen befinner seg i implementasjonen, tilgjengelig i vår repository:

https://github.com/308Bachelor4Ever/Arduino-IoT-experimental



Figur 19: Skjermbilde av GitHub repository

De fire viktigste mappene er dokumentert i repositoryen og oppfyller følgende funksjonalitet:

- 0. BlinkLED – Blinker LED-lyset på Arduino-brettet i intervaller. Lastes opp på Arduinoen for å konfigurere Wi-Fi modulen koblet til gjennom prototypebrettet. Uten denne mappen vil ikke Wi-Fi funksjoner være tilgjengelige.
- 1. GrandeFinale – implementerer funksjoner fra et bokprosjekt. Programmet har følgende funksjonalitet:
    - Vise tid og nåværende temperatur i rommet på en LCD.
    - Representere nåværende klokkeslett på en NeoPixel ring.
    - Representere temperatur på en servo.
    - Koble til Wi-Fi.
    - Laste opp temperaturinformasjon på ThingSpeak (krever Wi-Fi).
    - Vise på LCDen når data ble lastet opp sist.

GrandeFinale oppfyller grunnleggende IoT konsepter ved å lagre sensorinformasjon i en skytjeneste med overvåkningsformål. Denne informasjon kan komme til nytte dersom det er nødvendig å kartlegge innvendig temperatur over tid, og evt. tidsbestemme temperaturendringer i rommet basert på klokkeslett. Dette kan gi en konkret besparelse.

- 2. WifiBlynk – styre LED-diode med Blynk-appen. Bruker egen ESP8266 Wi-Fi implementasjon. Programmet har følgende funksjonalitet:
    - Koble til Wi-Fi.
    - Styre pinner på Arduino gjennom Blynk-appen.
    - Styre LED-lysdiode koblet til pinner gjennom Blynk.

WiFiBlynk oppfyller i stor grad våre to første krav i kravspesifikasjonstabellen. Disse kravene går ut på å styre lys over internett og kunne gjøre det via en app. Blynk-appen tillater å styre pinner på en Arduino som kan være koblet til ulike elektroniske apparater. I vårt tilfelle er det en enkel lysdiode som skal slås av og på.

Blynk-appen bruker en auth-token som kan hardkodes på Arduinoen. Dette tillater Arduino å snakke med vår app gjennom skytjenesten som Blynk leverer.

Vi bruker en egen Wi-Fi implementasjon for å få bedre debuginformasjon i Serial Monitor. I tillegg får vi sikre i forkant at Arduino er tilkoblet Wi-Fi, før Blynk kjøres.

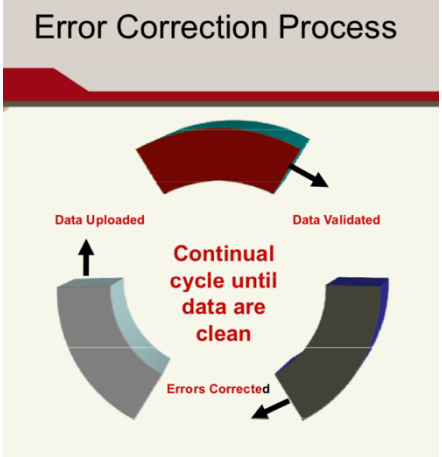
- 3. BlynkBlynk – Denne implementasjonen gjør det samme som 2. WifiBlynk, men den tar i bruk Blynk sine egne biblioteker for å opprette Wi-Fi tilkobling.

- 4. NeopixelSecond – Brukes for å teste Neopixel ringen.

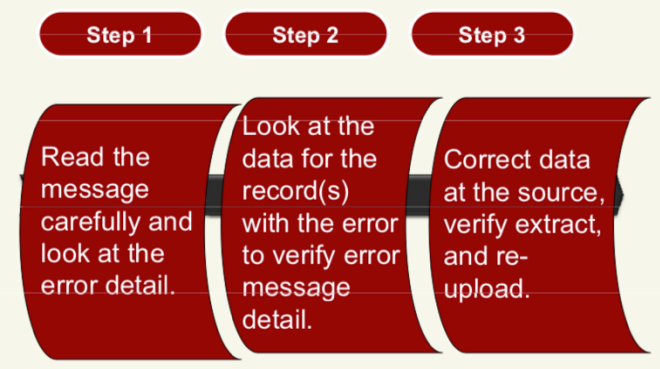
Se GitHub repositoryen for å se den dokumenterte koden.

6.0 Feil og retting syklus

I metoden forklarte vi hvorfor Test Driven Development (TDD) er nyttig i vårt prosjekt. Dette inngår som en del av feil og retting syklusen i prosjektet. I tillegg har vi en modell for feil og retting dersom det skulle oppstå uventede feil.



Figur 20: Feil og rettingsprosess



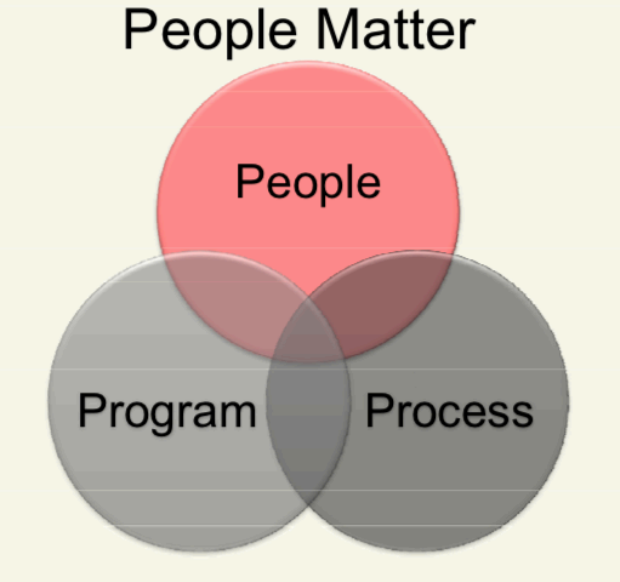
Figur 21: Forslag for å rette feil.

Modellene over (Georgia Departement of Education) beskriver en prosess som vi i stor grad har fulgt dersom problemer oppstår. Hvis feilmeldinger oppstå så prøver vi å verifisere hvorfor de oppstår, og retter eventuelle feil. Dette følges i form at en syklus frem til alle feilmeldinger er fjernet. TDD skal også bidra med å redusere feil i koden ved å skrive tester først. Vi har totalt 4 tester som sikrer at grunnfunksjonalitet er på plass.

- 4. NeopixelSecond: testing av Neopixel ring (forventet atferd).
- SerialTest: testing av seriekommunikasjon med Arduino. Til bruk for eventuell konfigurasjon.
- ThingspeakTest: testing av forventet tilkobling mot Thingspeak.
- WifiTest: testing av Wi-Fi modul med ulike biblioteker.

Debugging foregår i stor grad i Serial Monitor i Arduino IDE, da den gir oss status på de ulike modulene som er konfigurert til å sende tilbake informasjon. Dette gjelder Wi-Fi modulen, ThingSpeak implementasjonen og Blynk.

Til slutt har vi også en modell for feilretting som baserer seg rundt det brukeren forventer. Da går vi tilbake til brukerhistoriene for å bekrefte eller avkrefte at funksjonaliteten er på plass.



Figur 22: People Centric Error Correction (Georgia Department of Education).

Modellen er oppsummert i følgende punkter:

- Forstå brukerkrav
- Justere funksjonalitet etter behov
- Endre prosesser for å gjennomføre riktig implementasjon

7.0 Videre utviklingsmuligheter og forskningsområder

Gjennom utviklingen av prototypen er det en del lærdommer å få med seg som kan bidra til videre utvikling og forskning. I forhold til de Wi-Fi styrte lys implementasjonene kan Blynk oppleves som begrenset, og er kun egnet til prototypearbeid. Mindre bruk av tredjepartsbiblioteker og mer egenutviklet programvare kan hjelpe oss å utvide funksjonaliteten fremover. ThingSpeak-implementasjonen viser seg imidlertid å være fleksibel, da mange mulige datapunkter kan enkelt loggføres ved hjelp av Arduino.

IoT og smarte hjem er store tema som krever mer forskning. Her kan forskere komme på banen for å kartlegge hva som er beste praksis for å implementere smarte hjem løsninger, da utbredelsen av disse produktene blir bare større og større.

Vårt hovedforslag til videre forskning og utvikling er sikkerhet innenfor IoT og smarte hjem. Dette er på grunnlag av Blynk-implementasjonen som vi brukte, som benytter seg av auth-token generert fra en konto for å styre en bestemt mikrokontroller som har denne token i koden. Med en auth-token kreves ikke innlogging til en konto, og den kan lett endres på. Hvis flere databaser og kontoer blir involvert i en IoT implementasjon kan sikkerhet fort bli en utfordring. I tillegg har vi eksperimentert med muligheter for internettilkobling, og me hver enhet som er tilkoblet og tilgjengelig på internett innser vi at det kan være sikkerhetsutfordringer knyttet til det, både kjente og ukjente.

7.1 Utfordringen med IoT, smarte hjem og sikkerhet

Ettersom bruken av IoT øker, vil dette også bli et mer lukrativt område å angripe for hackere. Med smarte hjem kan meget sensitiv informasjon om personer som anvender disse enhetene komme på avveie. En studie gjennomført av HP i 2014 kom fram til at 70% av de mest brukte IoT-enhetene hadde store sikkerhetsutfordringer. De 10 mest populære IoT enhetene fra studiet hadde i gjennomsnitt 25 sikkerhetshull. Enhetene som ble testet var TVer, hjemmetermostater, fjernstyrte stikkontakter, og hjemmesentraler som kontrollerte flere enheter som dørlåser, alarmsystemer og garasjeporter.

De fem mest vanlige sikkerhetshullene som studiene avdekket var

- **Personvernhensyn** – 90% av testede enhetene samlet inn minst en bit av personlig informasjon via selve produktet, skyen eller mobilapplikasjonen
- **Utilstrekkelig autorisasjon** – 80% av de testede IoT enheter (inkludert sky og mobilkomponenter) klarte ikke å kreve passord som har tilstrekkelig kompleksitet og lengde.
- **Mangel på transportkryptering**- 70% av IoT-enhetene krypterte ikke kommunikasjon til internett og lokalt nettverk, mens halvparten av enhetenes mobilapplikasjoner utførte ukrypterte kommunikasjon til sky, internet eller lokalt nettverk.
- **Usikkert webgrensesnitt** – 6 av 10 enheter avdekket sikkerhetsproblemer med brukergrensesnitt. Svak legitimering av bruker og referanser som ble overført til klar tekst. 70% av enhetene med sky og mobile komponenter vil gjøre det mulig for en potensiell angriper å finnegyldige brukerkontoer ved hjelp av kontonummerering eller funksjon for tilbakestilling av passord.
- **Utilstrekkelig programvarebeskyttelse** – 60% av enhetene brukte ikke kryptering når man lastet ned programvareoppdatering. Dette er programvare som driver funksjonalitet til de testede enheter. Noen nedlastinger kunne også bli fanget opp, hentet ut og montering som et filsystem i Linux hvor programvaren kan ses og endres.

(HP Inc., 2014)

8.0 Konklusjon

IoT og smarte hjem har de siste årene åpnet nye arenaer for testing og utforskning. Vi har avdekket sentrale forståelser om smarte hjem og IoT, og hvordan disse to kan integreres. Grunnlaget i forskningen gjorde det mulig for oss å utvikle en metodesammensetning til dette prosjektet. Gjennom en helhetlig analyse og design utviklet vi en kravspesifikasjon i nok detalj til å utvikle prototyper. Deretter brukte vi par programmering og TDD for å kvalitetssikre og effektivisere utviklingsprosessen. Resultatet er en implementasjon som gir innsikt i hvordan det er mulig å bruke mikrokontrollere som Arduino UNO for å teste IoT og smarte hjem konsepter. Til slutt foreslår vi videre forskning, spesielt innenfor IoT sikkerhet, da flere og flere implementasjoner vil oppstå, og kvalitetssikring av disse implementasjonene vil være ytterst prioritet.

9.0 Refleksjonskapittel

Par programmering og TDD har vært veldig nyttige verktøy for rask utvikling av prototyper. Vi kunne imidlertid ønsket å utvikle mer spesifikke og konkrete funksjonelle prototyper tilpasset til brukerhistoriene. Vi fikk for eksempel ikke testet et lys montert hjemme, og måtte nøye oss med en LED-diode som kan slås av og på over internett.

Vi har også hatt tekniske hardware utfordringer med Arduino og Wi-Fi modulen. Problemet lå i strømforsyning, og det endte opp til slutt at Wi-Fi modulen brente opp.

Gjennom prosjektet har vi imidlertid tilegnet oss ny kunnskap innenfor IoT og smarte hjem implementasjoner. De ulike testprosjektene gav oss en forståelse for hvordan slike implementasjoner fungerer, og gir oss et meget bra utgangspunkt for videre utvikling. Til slutt har vi en «call to action» for videre forskning og utvikling i sikkerhetsområdet, da vi innså dette er en veldig konkret utfordring, selv med vår begrensede testing og enkle implementasjoner.

Referanseliste

Arduino. (2018). *What is Arduino?* Retrieved December 2018, from Gettings Started 1 Foundation -> Introduction: https://www.arduino.cc/en/Guide/Introduction

Georgia Departement of Education. (n.d.). *Best Practices for Error Correction.* Retrieved December 2018, from http://www.gadoe.org/Technology-Services/Data-Collections/Documents/DC_Conf_DataColl/Best%20Practices%20for%20Error%20Correction.pdf

Giusto, D., Iera, A., Morabito, G., & Atzori, L. (2010). *The Internet of Things: 20th Tyrrhenian Workshop on Digital Communications* (1 ed.). New York, United States: Springer-Verlag.

Han, D.-m., & Lim, J.-h. (2010). Smart home energy management system using IEEE 802.15.4 and zigbee. *IEEE Transactions on Consumer Electronics, 56*(3), 1403-1410.

Haughey, D. (2018). *MOSCOW METHOD*. Retrieved December 2018, from Project Smart ~ Exploring trends and developments in project managment today: https://www.projectsmart.co.uk/moscow-method.php

HP Inc. (2014, July 29). *HP Study Reveals 70 Percent of Internet of Things Devices Vulnerable to Attack.* Retrieved December 2018, from HP News: https://www8.hp.com/us/en/hp-news/press-release.html?id=1744676#.WP5R_ojyjb2

Intel. (n.d.). *A guide to the Internet of Things: How billions of online objects are making the web wiser*. Retrieved December 2018, from A Guide to the Internet of THings Infographic: https://www.intel.com/content/www/us/en/internet-of-things/infographics/guide-to-iot.html

Janzen, D., & Saiedian, H. (2005, September 19). Test-driven development concepts, taxonomy, and future direction. *Computer, 38*(9), 43-50.

Om Smarthus - Hva er Smart Hus? (n.d.). Retrieved December 2018, from smart-house: http://www.smartbuilding.no/hva-er-et-smarthus.html

Soliman, M., Abiodun, T., Hamouda, T., Zhou, J., & Lung, C.-H. (2014). Smart Home: Integrating Internet of Things with Web Services and Cloud Computing. *2013 IEEE 5th International Conference on Cloud Computing Technology and Science* (pp. 317-320). Bristol: IEEE.

Wang, C., Daneshmand, M., Dohler, M., Mao, X., Hu, R. Q., & Wang, H. (2013, October 10). Guest Editorial Special Issue on Internet of Things (IoT): Architecture, Protocols and Services. *IEEE Sensors Journal, 13*(10), 3505-3510.

Wikibooks. (2018, November 2). *C++ Programming: Programming languages, an introduction*. Retrieved December 2018, from Wikibooks: Open books for an open world: https://en.wikibooks.org/wiki/C%2B%2B_Programming/Programming_Languages

Williams, L., Kessler, R., Cunningham, W., & Jeffries, R. (2000, Jul/Aug). Strengthening the case for pair programming. *IEEE Software, 17*(4), 19-25.