Høgskulen på Vestlandet

Cloud-IoT Access Control System

Gruppe 23

DAT110 - Project 4

Cloud-IoT Access Control System

Navn	**Studentnummer**
Joachim Leiros	587728
Andreas Seljeset	587725
Fred Christiansen	587736
Sindre Holtan	587727

Innholdsfortegnelse

Introduksjon

Vi skulle utvikle et låsesystem som baserer seg på en bevegelsessensor(PIR) og to knapper som fungerer som et tastatur bestående av knapp en og to. Låsen skal kun åpne seg dersom knappene blir tastet inn i rett kombinasjon, i vårt tilfelle knapp en etterfulgt av knapp to. I tillegg til tastaturet har vi tre lys som signaliserer tre forskjellige tilstander for systemet, rød tilsvarer låst, gul tilsvarer ventende og grønn tilsvarer åpent.

Systemet starter i låst tilstand og tastaturet er inaktivt. Når det blir registrert bevegelse hos PIR går systemet over i ventende tilstand og tastaturet åpnes for inndata. En bruker taster så inn koden, gul lampe vil blinke for hvert tastetrykk. Dersom bruker taster korrekt kode åpnes låsen i ti sekunder før den går tilbake til låst tilstand. Skulle brukeren taste inn feil kode går systemet tilbake til låst tilstand.

For å gjennomføre prosjektet har vi utviklet en har utviklet en hardware implementasjon i tinkerCAD samt en software implementasjon av skytjeneste og enhetskommunikasjon i Java.

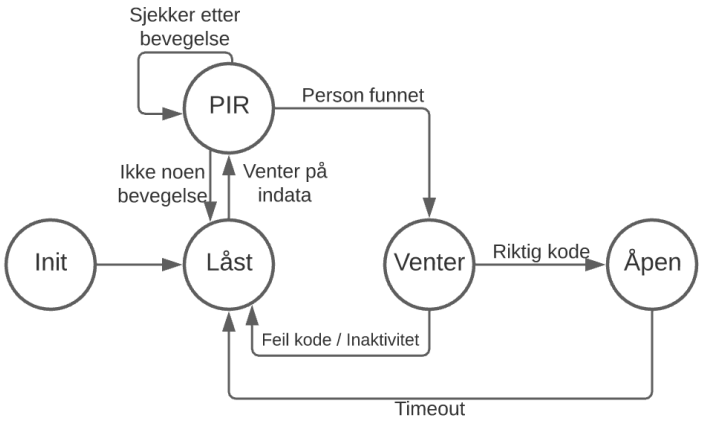
Link til TinkerCAD kan finnes TinkerCAD og JAVA finnes GitHub.

Access Control Design Model

Når vi starter tilstandsmaskinen starter prosessen låst. I låst tilstand venter programmet på inndata fra PIR sensor som kontinuerlig sjekker etter bevegelse. Dersom det ikke blir registrert noen bevegelse forblir systemet låst.

Dersom det skulle bli registrert bevegelse antar systemet at vi har en bruker som ønsker å låse opp ved bruk av tastaturet. Systemet går så over i en ventende tilstand hvor tastaturet blir gjort tilgjengelig for å motta inndata. En bruker trykker så inn sin kombinasjon av knapper for å forsøke låse opp systemet. Dersom en bruker taster korrekt kode vil systemet åpne seg og dersom en bruker taster feil går systemet tilbake mot låst.

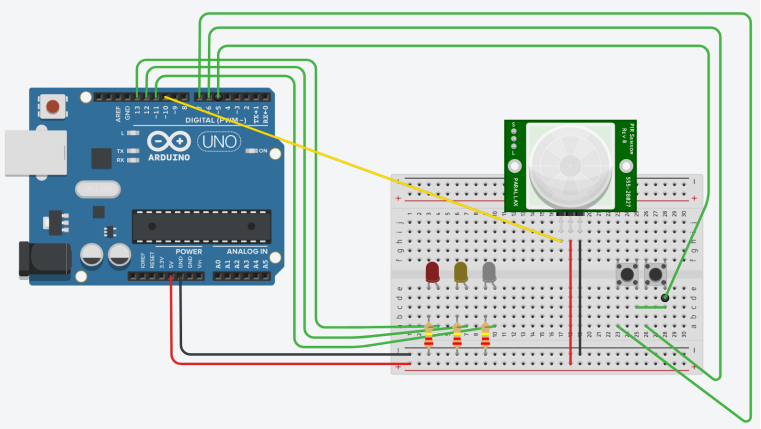
I vår tilstandsmaskin hovedsakelig tatt hensyn til kravspesifisering gitt i oppgaven for prosjektet. Systemet starter låst, når en knapp blir trykket på så skal gult lys blinke for å signalisere registrert trykk. Oppgaven har kun stilt krav om å ha to uavhengige fysiske knapper. I TinkerCAD har vi forsøkt å lage et sammensatt tastatur med disse knappene. I Java delen har vi gått bort fra dette.



Endelig tilstandsmodel

Access Control Hardware/Software Implementation

Vi implementerte tinkerCAD designet vårt til høyre slik at:
Rød led E3 går til kobling 13
Gul led E6 går til kobling 12
Grønn led E9 går til kobling 11.
PIR sensoren H16 til kobling 10 på ardruino brettet.



Systemet starter i låst tilstand i ett SWITCH-statement og det er laget til slik at den røde led pæren skal lyse fram til PIR sensoren registrerer bevegelse. Når sensoren registrerer bevegelse vil den røde pæren slukke vi går over i en ny case VENTER hvor det gule begynne og lyse for å signalisere at systemet er i ventemodus.

```
case LAAST:
 {
digitalWrite(ROED,HIGH);

      if(HIGH==digitalRead(PIR)){
      tilstand=VENTER;
      digitalWrite(ROED,LOW);
      }
}
break;
```

```
case VENTER:
{
       digitalWrite(GUL,HIGH)
       input = tastatur.getKey();


            if (input){
             Serial.print("trykk");
             blink(GUL,5);
             kode[kode_lengde] = input;
             kode_teller++;
      }

            if(kode_teller == kode_lengde-1) {
      if (!strcmp(kode,faktiskKode)){
                  digitalWrite(GUL,LOW);
                  tilstand=AAPEN;

                   } else {
                        digitalWrite(GUL,LOW);
                        tilstand = LAAST;
                         }
                }
}
break;
```

Når systemet er i ventemodus åpnes det for at knappene kan motta inndata fra bruker. Knappene er seriekoblet som et tastatur med en rad og to kolonner.

Knapp nummer en er koblet fra E23 til arduino 7, knapp nummer to er koblet fra E26 til arduino 6. Systemet får beskjed om at arduino 7 og 6 er to forskjellige kolonner. Seriekoblingen får fra C25 via C28 til arduino 5.

```
byte rowPins[1] = {5};
byte colPins[2] = {7,6};
char keymap[1][2]= {
  {'1', '2'}
};
```

Når systemet er i ventemodus så står brukeren klar til å taste inn koden, som i vårt tilfelle er satt til “12”, eller 1 etterfulgt av 2. Når korrekt kode er registrert mens systemet er i ventemodus vil vi gå over til case AAPENT som åpner systemet i 10 sekunder før det deretter låses. Dersom brukeren taster feil kode går vi tilbake til case LAAST.

```
case AAPEN:{

digitalWrite(GRONN,HIGH);
delay(10000);
tilstand=LAAST;
digitalWrite(GRONN,LOW);
}
break;
```

REST API skytjenester

For å implementere REST API måtte vi først implementere metoder som gjør det mulig for programmet å prate med skytjenesten. For å kunne utføre operasjonene som var ønsket måtte vi implementere en identifikator knyttet til loggen. Dette ble gjennomført ved å legge til en ID for accessEntry til en logg som ble lagt sammen med meldingen som ble sendt. Samtidig ble det lagd metoder for å hente ut AccessEntry og for å slette AccessEntry loggen.

```
public int add(String message) {

    int id = cid.getAndIncrement();
    AccessEntry entry = new AccessEntry(id, message);
    log.put(id, entry);
    return id;
}
```

Samtidig skulle loggen returneres som JSON. For å representere dataene som JSON benyttet vi oss av GSON sin toJson metode for å konvertere vårt Java-hashmap til JSON. Vi lagde en toJson som tok en String *js* og itererte gjennom loggen og via Gson sin toJson metode skrev disse til Stringen *js* i Json format.

```
public String toJson() {
    Gson gs = new Gson();
    String js = null;
    ConcurrentHashMap<Integer, AccessEntry> clone = new ConcurrentHashMap<Integer, AccessEntry>(log);

    js += "[";
    if (!log.isEmpty()) {
        Iterator<Entry<Integer, AccessEntry>> it = clone.entrySet().iterator();
        while (it.hasNext()) {
            Entry<Integer, AccessEntry> entry = it.next();
            js += gs.toJson(entry.getValue());
            it.remove();
            if (it.hasNext()) {
                js += ",";
            }
        }
    }
    js += "]";
    return js;
}
```

Så skulle vi implementere HTTP operasjoner slik at:

POST /accessdevice/log/ loggfører aksesseringsforsøk i loggen i HTTP spørringen.

```
post("/accessdevice/log/", (req, resp) -> {

        Gson gs = new Gson();
        AccessMessage message = gs.fromJson(req.body(), AccessMessage.class);

        int id = accesslog.add(message.getMessage());
        AccessEntry entry = new AccessEntry(id, message.getMessage());

        return gs.toJson(entry);
});
```

GET /accessdevice/log/ returnerer en JSON representasjon av aksesseringsloggen.

```
get("/accessdevice/log/", (req, resp) -> {

    return accesslog.toJson();
});
```

GET /accessdevice/log/{id} returnerer en JSON representasjon på {id}.

```
get("/accessdevice/log/:id", (req, resp) -> {
    Gson gs = new Gson();
    int id = Integer.parseInt(req.params("id"));
    AccessEntry entry = accesslog.get(id);

    return gs.toJson(entry);
});
```

PUT /accessdevice/code skal oppdatere koden til systemet i skyen. Den nye koden skal lagres i JSON format.

```
put("/accessdevice/code/", (req, resp) -> {
    Gson gs = new Gson();
    AccessCode codes = gs.fromJson(req.body(), AccessCode.class);
    accesscode.setAccesscode((codes.getAccesscode()));
    return gs.toJson(codes);
});
```

GET /accessdevice/code skal returnere den nåværende koden som er lagret på server.

```
get("/accessdevice/code/", (req, resp) -> {
    Gson gs = new Gson();
    return gs.toJson(accesscode);
});
```

DELETE /accessdevice/log/ skal slette alle rader i aksesseringsloggen og returnere en tom JSON logg.

```
public void clear() {
    log.clear();
}
```

```
delete("/accessdevice/log/", (req, resp) -> {

    accesslog.clear();

    return accesslog.toJson();
});
```

Device Communication

Vi først lagde nye konstruktører til URL og en ny client hvor vi benytter oss av OkHttp biblioteket for å gjøre arbeidet lettere da vi kan benytte oss av OkHttpClient(); som en egen klient samt benytte oss av forskjellige metoder tilbudt av rammeverket.

Vi skulle så lage en metode for *doPostAccessLog.* Metoden legger til en logg når låsen blir aksessert og gir tilbakemelding på om koden som ble registrert ble godkjent eller ikke, dette skjer ved at den sender en post-request til *localhost:8080/accessdevice/log/*. Her har vi benyttet oss av OkHttp til å bygge en POST-request som vi så sender til klienten.

```
public void doPostAccessLog(String message) {

    RequestBody body = RequestBody.create(JSON, gson.toJson(new AccessMessage(message)));
    Request req = new Request.Builder().url(URL + LOGPATH).post(body).build();

    try {
        Response resp = client.newCall(req).execute();
        String json = resp.body().string();
        System.out.println("json from doPostAccessLog: " + json);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Så skulle vi og lage metoden *doGetAccessCode.* Her benytter vi oss også av OkHttp for å bygge en get GET-request sendt til *localhost:8080/accessdevice/code/* som henter ut gjeldende kode for systemet.

```
public AccessCode doGetAccessCode() {

    Request req = new Request.Builder().url(URL + CODEPATH).get().build();

    try {
        Response resp = client.newCall(req).execute();
        String json = resp.body().string();
        System.out.println("json from doGetAccessCode: " + json);
        return gson.fromJson(json, AccessCode.class);
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}
```

System Testing

Vi testet IoT-cloud systemet ved og kjøre begge applikasjonene, skrudde på nettverksmodus i programmet, trykket på sensoren trykker på sensoren og ventet på at programmet skal gå over i ventemodus, trykker så inn koden 1,2 og fikk som forventet korrekt kode.

Vi fortsatte så med å byttet så koden i Postman til 2,1 og testet koden på nytt den første koden Vi fikk da en feilmelding i Postman om ingen tilgang. Testet så med koden 2,1 og dette funket og Postman loggen stemte overens med våre handlinger i programmet.

```
PUSHBOTTON pin 7
CHECKING
json from doGetAccessCode: {"accesscode":[1,2]}
UPDATING CODE
UNLOCKED
json from doPostAccessLog: {"id":0,"message":"UNLOCKED"}
json from doPostAccessLog: {"id":1,"message":"LOCKED"}
LOCKED
PIR SENSOR pin 2
WAIT1P
PUSHBOTTON pin 6
WAIT2P
PUSHBOTTON pin 7
CHECKING
json from doGetAccessCode: {"accesscode":[2,1]}
UPDATING CODE
LOCKED
json from doPostAccessLog: {"id":2,"message":"ACCESS DENIED"}
PIR SENSOR pin 2
WAIT1P
PUSHBOTTON pin 7
WAIT2P
PUSHBOTTON pin 6
CHECKING
json from doGetAccessCode: {"accesscode":[2,1]}
UPDATING CODE
UNLOCKED
json from doPostAccessLog: {"id":3,"message":"UNLOCKED"}
json from doPostAccessLog: {"id":4,"message":"LOCKED"}
LOCKED
```

Konsoll utskrift


```
null[
    {
        "id": 0,
        "message": "UNLOCKED"
    },
    {
        "id": 1,
        "message": "LOCKED"
    },
    {
        "id": 2,
        "message": "ACCESS DENIED"
    },
    {
        "id": 3,
        "message": "UNLOCKED"
    },
    {
        "id": 4,
        "message": "LOCKED"
    }
]
```


Postman logg

Konklusjon

I begynnelsen hadde vi ganske lite erfaring med TinkerCAD og er usikre på resten relatert til nettverks programmeringen. Men vi har fått mye hjelp og lært mye underveis. Mot slutten fikk vi en nesten fungerende versjon som vi gjorde en del feilsøking på før vi fikk et resultat som fungerte som ønsket.

I løpet av prosjektet brukte vi mye tid på å få TinkerCAD til å fungere som vi ville. Vi har prøvd flere forskjellige måter å møte oppgaven på og endte opp med et resultat som fungerer noenlunde ut fra kravspesifiseringen som var gjort.

Del B var krevende, og krevde at samtlige på gruppen jobbet mye med å forstå hvordan det skulle løses og hvordan vi skulle implementerte det. På samme måte som i del A brukte vi en mye tid på å sette oss inn i de forskjellige rammeverkene vi tok i bruk gjennom oppgaven. Samtidig syns vi det var utfordrende å lage en god metode for å oversette Java til Json samt å håndtere de forskjellige http forespørslene.