Access-Control-System

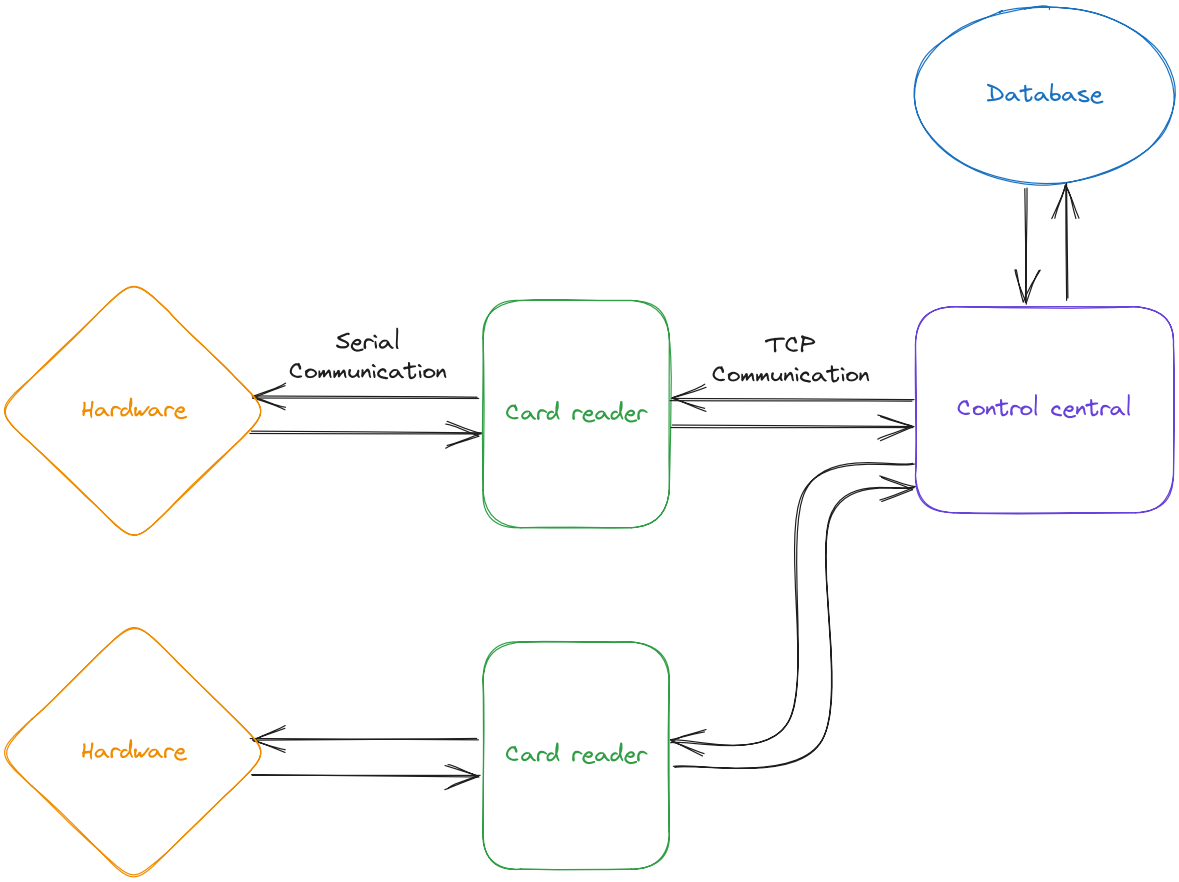
Introduksjon

[Github Hovedprosjekt]
[Github Database]

Gruppemedlemmer

- Kristian Klette, Gruppeleder
- Tor Magnus Haldaas, Database administrator
- Ryan Le
- Victor Newman
- Nora Hisdal

Kommunikasjon



Filer

SimSim

Eksternt levert kode for å simulere kortleser. Små ting som error handling og logging av tid er fikset.

Kortleser

Kortleser inneholder logikk for å motta data fra fysisk kortleser (eller simulert kortleser), og kommuniserer med sentral over TCP. Den har en mappe for seriell kommunikasjon og en mappe for TCP kommunikasjon. Klassen DoorStateManager styrer adgang for døren.

Sentral

Sentral inneholder logikk for å koble til database og håndtere funksjoner, Koble til kortleser via TCP og ta imot kommandoer fra bruker. Den har en mappe for database relatert kode, en mappe for TCP relatert kode og en mappe for konsoll interaksjoner.

Database mappe

Config

Inneholder ting som IP-adresse, brukernavn, passord osv. for databasetilkobling, men og funksjoner og parametere for å kommunisere med database.

DataClasses

Inneholder data klasser for ting som brukes i forhold til databasen.

Processing

Inneholder behandling av forespørsler for å gjøre ting som har med database å gjøre.

Services

Inneholder servicen som kan kalle på alle forespørsel behandlere ut ifra hva som trengs.

Annet

- DatabaseConnectionManager: håndterer kobling til database.
- DatabaseAccess: håndterer direkte interaksjon med database.

TCP mappe

DataClasses

Inneholder data klasser for ting som brukes i forhold til TCP kommunikasjon.

TcpRequests

Inneholder alle request typer som kan komme fra kortleser, i tillegg til respons type.

Processing

Inneholder klasser som behandler forespørsler og respons.

Annet

- TcpConnectionManager: håndterer kobling til klienter (kortlesere).
- TcpClientHandler: tar imot forespørsler fra klienter (kortlesere) og svarer de ved hjelp av behandler klasser. Passer og på korrekt lukking av TCP.
- IClientManager: brukt for å kunne sjekke ting i client handler fra behandlere da dette er påvirker noen av responsene.

Console

Gjør noe av brukerinteraksjon mer abstrakt til en viss grad. ConsoleConnectionManager har events som gjør at console kan reagere på hva som skjer i denne klassen i forhold til bruker input.

Brukerinformasjon

Sentral

Sentral brukes til å administrere brukerdata. Man kan legge til, redigere og fjerne brukere i database. Man kan og se status på tilkoblede kortlesere og database. Man skriver inn den kommandoen man vil bruke. For å koble seg til database må man kun passe på at det ligger riktig informasjon i config mappen, og dette er ikke noe en vanlig bruker trenger å gjøre. TCP klienter kobler seg til automatisk i bakgrunnen. Kommandoer er forklart i program.

Kortleser

Når man starter kortleser vil den vise ledige COM porter. man skriver inn hvilke port man vil bruke. Deretter vil programmet prøve å koble seg til TCP server (sentral). Merk: kortleser er konfigurert med en loopback IP-adresse! For å få full funksjonalitet må det gjøres endringer rundt dette, men også noe en vanlig bruker ikke trenger å gjøre. Etter at kortleser har koblet seg til seriell kommunikasjon og server vil konsollen be om kort-id og pin-kode. Kortleser interface (simulator) er nå i gang, gitt at det programmet og kjører. Når man vil låse opp dør skriver man inn kort-id og pin-kode og døren vil låse seg opp.

SimSim

Dette programmet simulerer et kortleser interface. Input 5 simulerer forsøk på å åpne dør, der output 5 viser om man får åpnet døren. Output 6 viser om døren er låst eller ikke. Output 7 viser alarm tilstand.

- Når man låser opp dør har man en tidsbegrensning for å åpne døren før den låses igjen.
- Ved forsøk på innbrudd (termistor slider) vil alarm utløses og logges i database.
- Hvis døren står åpen for lenge vil alarm utløses og logges i database.
- Når døren lukkes låses den automatisk. Dør må låses opp via kortleser program.

Kjente feil

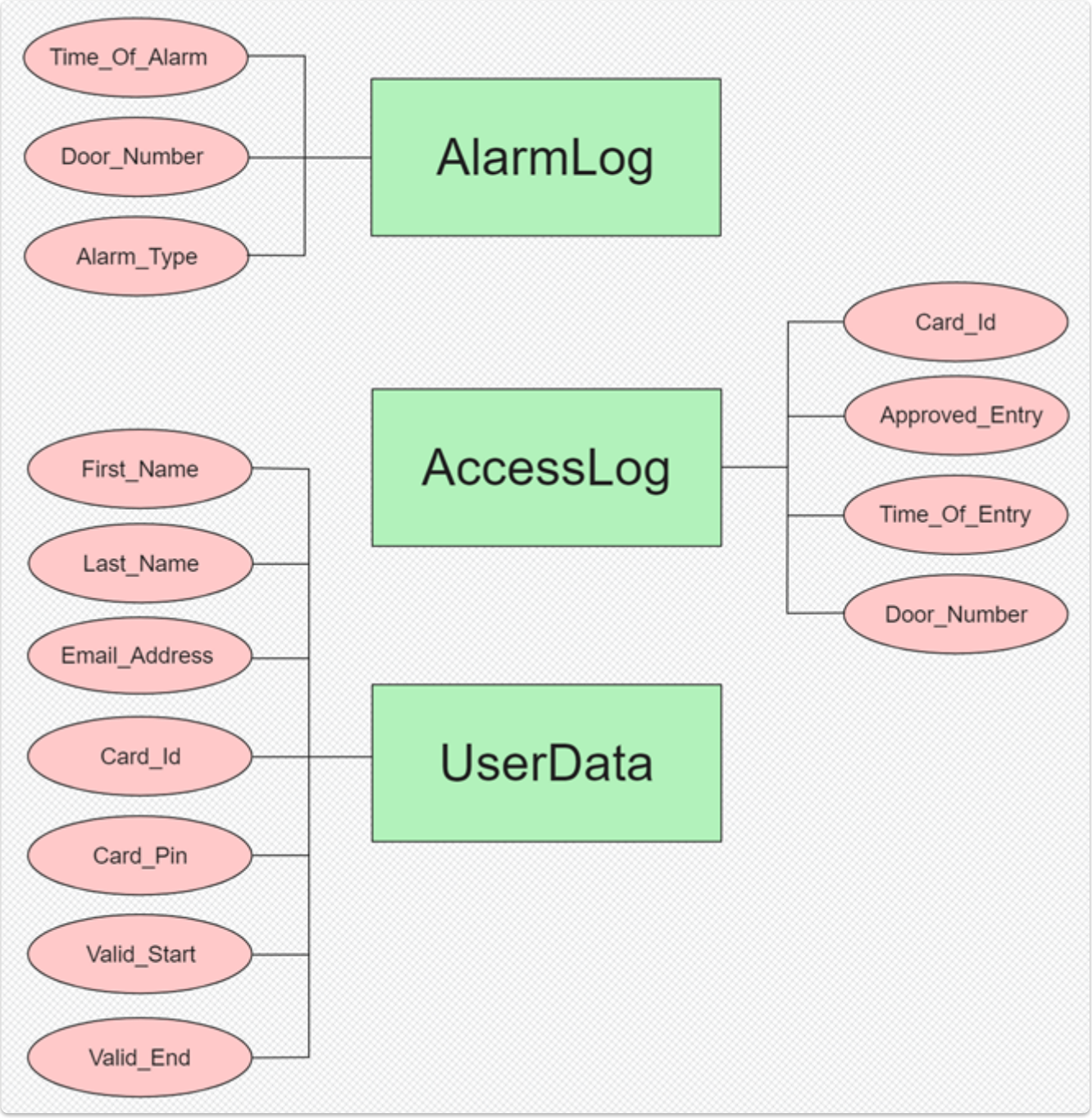
Kommunikasjon mellom kortleser interface og kortleser program er veldig treg, spesielt når den sender automatisk, som den bør gjøre som standard. Usikker på hva som gjør dette.

Mangler

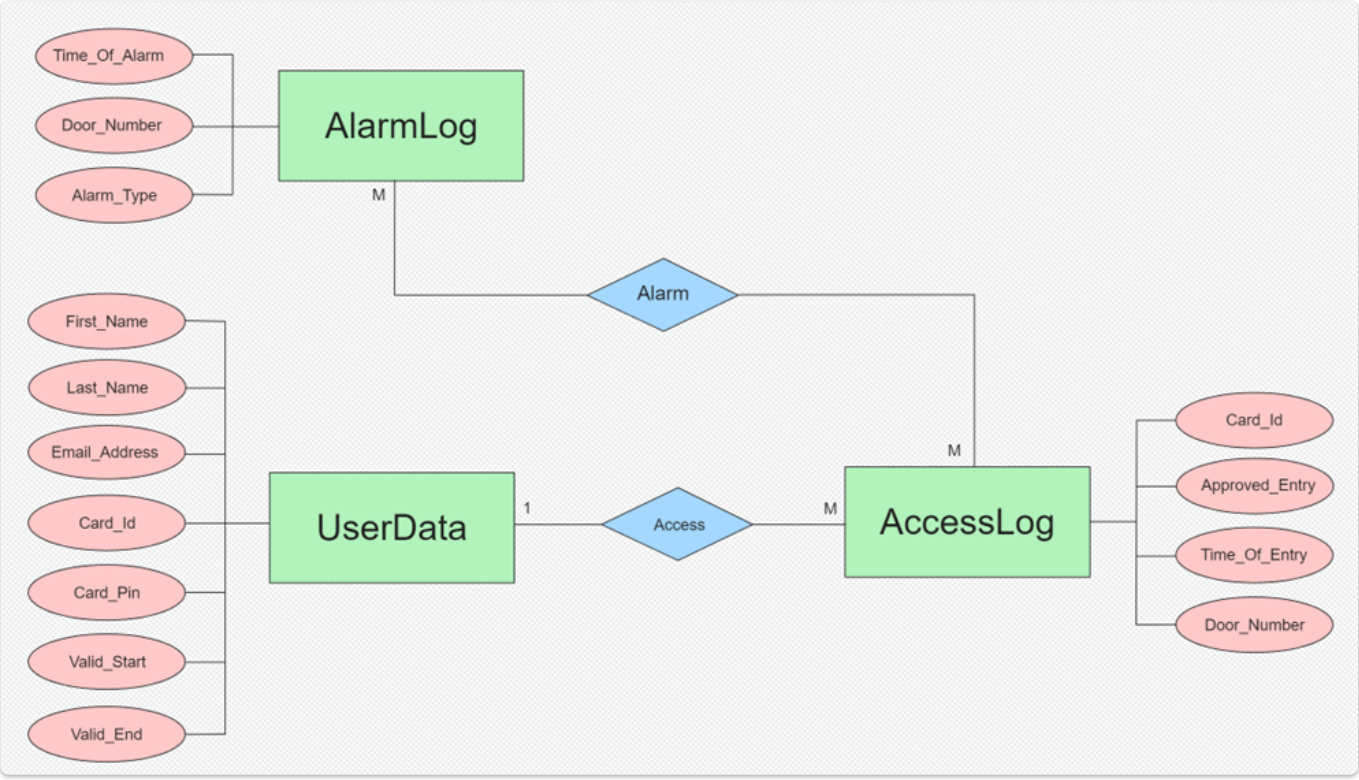
Har ingen URL forhold i databasen, dette ville vært en stor fordel for å forhindre data som ikke stemmer overens med de andre tabellene å bli lagt inn.

ER-modell

Implementert ER-Modell



Planlagt ER-Modell



SQL-kode

UserData

Tabell

```
Create table UserData
(
        First_Name varchar(20),
        Last_Name varchar(20),
        Email_Address varchar(20),
        Card_Id char(4) primary key,
        Card_Pin char(4),
        Valid_Start timestamp,
        Valid_End timestamp
);
```

AddUser(@cardId, @cardPin, @emailAddress, @firstName, @lastname, @validEnd, @validStart)

```
create or replace function AddUser(cardId_ char(4), cardPin_ char(4), emailAddress_
varchar, firstName_ varchar, lastName_ varchar, validEnd_ timestamp, validStart timestamp)

return bool as
$$
begin
        insert into userdata(first_name, last_name, email_address, card_id, card_pin,
valid_start, valid_end)
        values(firstName_, lastName_, emailAddress_, cardId_, cardPin_, validStart_,
validEnd_);
        if found then
                return 1;
        else
                return 0;
        end if;
end
$$
language plpgsql
```

UpdateUser(@cardId, @cardPin, @emailAddress, @firstName, @lastname, @previousCardId, @validEnd, @validStart)

```
create or replace function UpdateUser(cardId_ char, cardPin_ char, emailAddress_ varchar,
firstName_ varchar, lastName_ varchar, previousCardId_ char(4), validEnd_ timestamp,
validStart_ timestamp)
returns bool as
$$
begin
        update userdata
        set
                first_name = firstName_,
                last_name = lastName_,
                email_address = emailAddress_,
                card_id = cardId_,
                card_pin = cardPin_,
                valid_start = validStart_,
                valid_end = validEnd_
        where card_id = previousCardId_;
        if found then
                return 1;
        else
                return 0;
        end if;
end
$$
language plpgsql;
```

GetUserbase()

```
create or replace function GetUserbase()
returns table
(
        firstName varchar,
        lastName varchar,
        cardId char
)as
$$
begin
        return query
        select first_name, last_name, card_id from userdata order by card_ID;
end
$$
language plpgsql;
```

GetUser(@cardId)

```
create or replace function GetUser(cardId_ char(4))
returns table
(
        firstName varchar,
        lastName varchar,
        emailAddress varchar,
        cardId char,
        cardPin char,
        validStart timestamp,
        validEnd timestamp
)as
$$
begin
        return query
        select first_name, last_name, email_address, card_id, card_pin, valid_start,
valid_end from
                userdata where card_id = cardId_ order by card_id;
end
$$
language plpgsql;
```

PeekUser(@cardId)

```
create or replace function PeekUser(cardId_ char(4))
returns bool as
$$
begin
perform distinct card_id from userdata
where card_id = cardId_;
        if found then
                return 1;
        else
                return 0;
        end if;
end
$$
language plpgsql;
```

RemoveUser(@cardId)

```
create or replace function RemoveUser(cardId_ char(4))
returns bool as
$$
begin
        delete from userdata
        where card_id = cardId_;
        if found then
                return 1;
        else
                return 0;
        end if;
end
$$
language plpgsql;
```

ValidateUser(@cardId, @cardPin)

```
create or replace function ValidateUser(cardId_ char, cardPin_ char)
returns bool as
$$
begin
perform distinct card_id, card_pin from userdata
        where card_id = cardId_ and card_pin = cardPin_;
        if found then
                return 1;
        else
                return 0;
        end if;
end
$$
language plpgsql;
```

AccessLog

Tabell

```
Create table AccessLog
(
        Card_Id char(4),
        Approved_Entry bool,
        Time_Of_Entry timestamp,
        Door_Number int
);
```

AddAccessLog(@cardId, @approvedEntry, @timeOfEntry, @doorNumber)

```
create or replace function AddAccessLog(cardId_ char(4), approvedEntry_ bool, timeOfEntry_
timestamp, doorNumber_ int)
returns bool
as
$$
begin
        insert into accesslog values(cardId_, approvedEntry_, timeOfEntry_ , doorNumber_);
        if found then
                return true;
        else
                return false;
        end if;
end
$$
language plpgsql;
```

DoorReport(@doorNumber, @startDate, @endDate)

```
create or replace function DoorReport(doorNumber_ int, startDate_ timestamp, endDate_
timestamp)
returns table
(
        cardId char(4),
        approvedEntry bool,
        timeOfEntry timestamp,
        doorNumber int
)as
$$
begin
        return query
        select * from accesslog
                where door_Number = doorNumber_
                and time_of_entry between startDate_ and endDate_
                order by time_of_entry desc;
end
$$
language plpgsql;
```

AccessReport(@startDate, @endDate)

```
create or replace function AccessReport(startDate_ Date, endDate_ Date)
returns table
(
        cardId char,
        approvedEntry bool,
        timeOfEntry timestamp,
        doorNumber int
)
as
$$
begin
        return query
        select * from accesslog
                where
                        time_of_entry between startDate_ and endDate_
                order by time_of_entry desc;
end
$$
language plpgsql;
```

SuspiciousUsers()

```
create or replace function SuspiciousUsers()
returns table
(
        cardId char
)as
$$
begin
        return query
        select distinct card_id from accesslog
        group by card_id
        having count(approved_entry) ≥ 10;
end
$$
language plpgsql;
```

AlarmLog

Tabell

```
create table AlarmLog
(
        Time_Of_Alarm timestamp,
        Door_Number int,
        Alarm_Type varchar(20)
);
```

AddAlarmLog(@timeOfAlarm, @doorNumber, @alarmType)

```
create or replace function AddAlarmLog(timeOfAlarm_ timestamp, doorNumber_ int, alarmType_
varchar(7))
returns bool
as
$$
begin
        insert into AlarmLog values(timeOfAlarm_, doorNumber_, alarmType_);
        if found then
                return true;
        else
                return false;
        end if;
end
$$
language plpgsql;
```

AlarmReport(@startDate, @endDate)

```
create or replace function AlarmReport(startDate_ timestamp, endDate_ timestamp)
returns table
(
        timeOfAlarm timestamp,
        doorNumber int,
alarmType varchar
)
as
$$
begin
        return query
        select * from alarmlog
                where
                                time_of_alarm between startDate_ and endDate_
                order by time_of_alarm desc;
end
$$
language plpgsql;
```