Software Architecture Document (SAD)

Et overblik og gennemgang af projektets software arkitektur og grundlag

Morten Lund Mikkelsen

Rasmus Meldgaard Jensen

Christian Agesen Boes

Mads Lajer Rasmussen

Login & fremgangsmåde findes i afsnit 2 Anvendelsesguide til systemet

GitHub: https://github.com/rmlajer/Festivalcito.git

Hjemmeside: https://festivalcito.azurewebsites.net/

21-12-2022 Pba i IT-arkitektur

# 1. Guidelines for dokumentation & kodestruktur

Vi har lavet nedenstående dokumentation og guidelines til hvordan vi vil opbygge vores projekts kode og hvordan vi ville holde arkitekturen.
Grunden til vi har lavet dette er for at få et fælles landkort vedrørende strukturen og de retningslinjer vi skal følge for at gøre det mest mulig overskueligt for os selv.

- Alt C# skal skrives i Blazor Komponenter dvs..razor.cs.
- Alle variabler skal skrives i camelCase.
- Alle funktioner/metoder skal skrives i PascalCase.
- Der skal være konsistens i navngivning - Alle variabler og metoder skal navngives i beskrivende ord og på engelsk.
- Alt CSS skal skrives med Bootstrap i html tag.
- Hver metode skal sigte efter at have minimum en beskrivende kode kommentar når den deklareres.
- Der skal være indledende beskrivende kommentar på alle komponenter

# 2. Anvendelsesguide til systemet

*Systemet skal ses i følgende kontekst.*
Alle brugere har til formål at arbejde med vagter på musikfestivalen inddelt i forskellige fastsatte arbejdsområder. Brugere passer i en af tre overordnede kategorier:

Brugertype 1: *Koordinator af frivillige på ovennævnte festival.*

- Primær fokus hos denne bruger er at håndtere og prioritere vagter samt angive arbejdsområde til frivillige.

Brugertype 2: *Eksisterende frivillig med angivet arbejdsområde.*

- Primær fokus hos denne bruger er at finde og håndtere vagter under festivalen med henblik på at optjene point i systemet.

Brugertype 3: *Ny ansøger til stilling som frivillig på festivalen.*

- Primær fokus hos denne bruger er at oprette en ny konto med korrekt persondata således at denne kan blive givet et arbejdsområde og blive en del af bruger 2.
- Denne har begrænset adgang til systemet før et arbejdsområde er tildelt.

Med denne kontekst in mente kan systemets forskellige sider beskrives fra en brugers perspektiv:

## 2.1 Registration_LoginPage:

Brugertype 1 og 2 har her mulighed for, ved hjælp af eksisterende login information, at oprette en midlertidig session i systemet og bliver, når disse er accepteret, automatisk videresendt til den relevant side.

- Brugertype 1 sendes til CoordinatorShiftPage.
- Brugertype 2 sendes til VolunteerPage.

Brugertype 3 har på samme side mulighed for at indtaste nødvendige persondata for oprettelse af ny brugerkonto. Når en ny brugerkonto oprettes succesfuldt, sendes denne bruger videre.

- Brugertype 3 sendes til VolunteerPage

## 2.3 Coordinator_ShiftPage

Brugertype 1 lander på denne side efter succesfuldt login på Registration_LoginPage eller via navigering fra Coordinator_VolunteerPage.

På denne side kan brugertype 1 oprette ny vagter, opdatere disse med ny information og eventuelt slette vagter - Hvis database relationer tillader dette - Altså at vagter ikke har tildelte frivillige. Bruger 1 har fra denne side mulighed for at navigere til Coordinator_VolunteerPage.

## 2.4 Coordinator_VolunteerPage

Brugertype 1 lander på denne side efter succesfuldt login via navigering fra Coordinator_VolunteerPage. Fra denne side kan brugertype 1 tage brugere af type 3 fra en global liste af frivillige og oprette en tildeling af disse på det, til relevant brugertype 1, tildelte

arbejdsområde. Ligeledes kan frivillige slettes fra det relevant arbejdsområde. Dette, fortsætter accepterede database relationer, dvs. ingen eksisterende vagt tildeling.
Brugertype 1 kan herfra sende en automatisk email til frivillige med information om fremtidig festival.
Brugertype 1 har fra denne side mulighed for at navigere til Coordinator_ShiftPage.

## 2.5 VolunteerPage

Brugertyper 2 og 3 lander på denne side efter succesfuldt login på Registration_LoginPage. Både brugertyper 2 og 3 har her mulighed for at opdatere persondata. For brugertype 3 ender rejsen midlertidigt her, da denne brugertype ikke har adgang til vagter i systemet før et arbejdsområde er tildelt.
Brugertype 2 har på denne side mulighed for at se et overblik over alle vagter på det relevant arbejdsområde. Derudover kan brugertype 2 via en knap på hver vagt tage eller afmelde sig fra vagter. Vagter vises i forskellige farver alt efter status som beskrevet i toppen af siden.

Til test af systemet kan anvendes følgende logindetaljer:

Brugertype 1:

Email: madsl@mail.com
Password: 123456
Area: Security

Email: bar@coordinator.com
Password: 123456
Area: Bar

Email: stage@coordinator.com
Password: 123456
Area: stage

Brugertype 2:

Rolle: Frivillig

Email: bob@mail.com

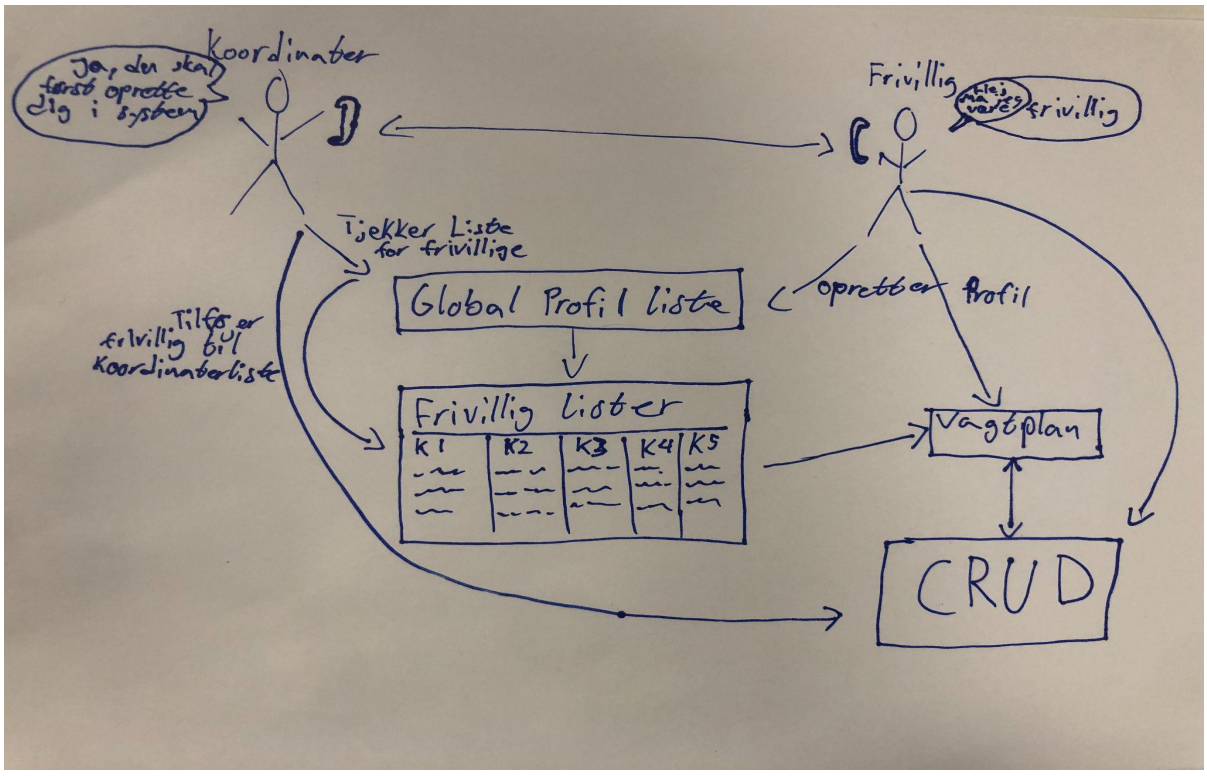
Password: 123456

Brugertype 3:

Tester kan her oprette eget login testdata som frivillig. Disse kan via første login tildeles til arbejdsområde og således flyttes til bruger 2.

# 3. Rigt billede

Tidligt i processen tegnede vi et rigt billede for at danne et fælles kort over systemets forskellige dele og interaktioner(Mathiassen, et al., [n.d.]). Dette blev primært anvendt som et diskussions værktøj i den tidlige process, primært mht. databasedesign og overordnet arkitektur.




# 4. Valg af database

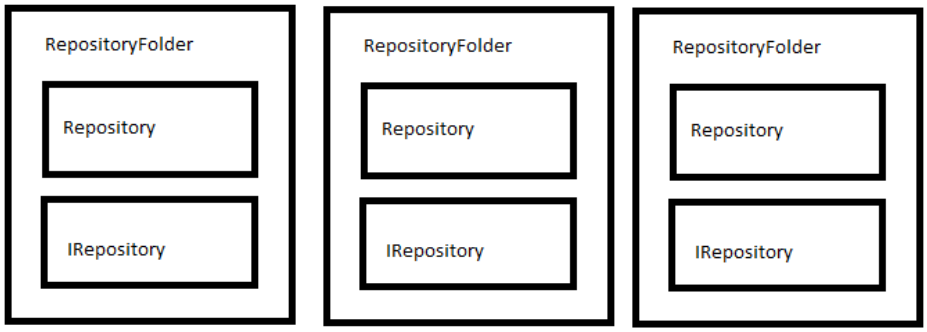
Vi som projektgruppe har haft en dialog  omkring hvilken database vi skulle vælge at arbejde med til dette projekt.

Længe stod valget på at vi gerne vil lave et hybrid database valg, mellem Mongodb og postgreSQL.
Grunden til at vi stod med disse overvejelser var, at hybrid valget vil give det bedste af to databaser som eksempel ville vi have mulighed for at lave en global liste, med alle frivillige og gemme dataen på disse gennem flere år og system-iterationer, og samtidigt have styrkerne fra en PostgreSQL relationelle database.
Vi valgte i sidste ende at arbejde med en ren postgreSQL database løsning. Eftersom denne model optimalt løser vores krav uden unødvendigt arbejde.
PostgreSQL databasens relationelle model hjælper os med at sikrer dataintegritet. Samtidig får man muligheden for at at lave et rigid schema, som giver et godt overblik.
I vores udtænkte system og hele den måde det skal sættes op på giver det mulighed for at skabe en relation mellem tabeller.

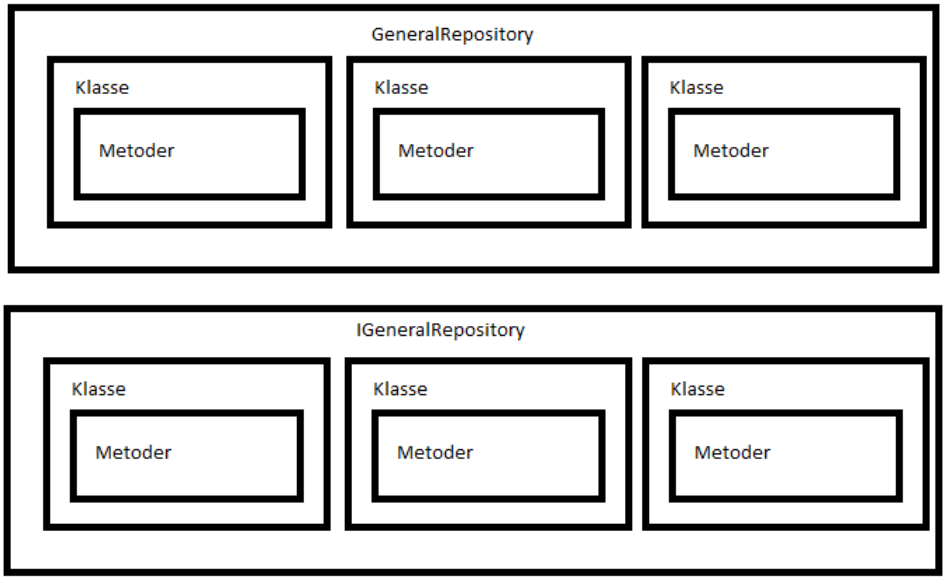
# 5. Valg af repository struktur

*Vi har i løbet af projektet haft en samtale omkring struktur af repository i vores Blazor projekt, da vi ser 2 mulige veje.*

1. Hver ressource får oprettet separat repository i egen fil, med eget interface i hver sin folder:




2. Alle Repository metoder oprettes per ressource i et enkelt GeneralRepository fil med ét generelt IGeneralRepository




Samtalen udspringer af en ïde om at anden løsning potentielt giver mest mening for os, af følgende grunde:

- Det giver et bedre visuelt overblik over Repositories
- Vi kan fortsætte kompositstrukturen hvor komponenter holdes separat, hvilket burde skabe mindre problemer internt i koden.
- Scalability er bedre, da vi har mulighed for at gro repositories horisontalt frem for første løsnings vertikale tilgang.

Den grundlæggende arkitektur i Blazor påtvinger os til at anvende tilgang nr. 1 da vi har brug for at oprette individuelle interfaces til hver service grundet netværksstrukturens anvendelse af Dependency Injection - Som desværre ligger udenfor denne opgaves scope.

# 6. Connection String

Vi har derudover også haft en samtale omkring Connection String - Som det er nu skal vi oprette et nyt PgAdminContext object hver gang vi opretter forbindelse til databasen, da denne klasse udelukkende indeholder Connection String føles det en smule redundant.
Vi taler igen om at scalability potentielt er bedre med nuværende arkitektur, men at det ikke nødvendigvis er optimalt i vores, relativt begrænsede, projekt - Dette kunne eventuelt diskuteres med arkitekter under styregruppemøde.

- *Kan Interfaces nedarve fra en global klasse som indeholder connection string og således have strengen per automatik?*
- *Er det smart at alle repositories blot nedarver fra en fælles klasse, som blot indeholder connectionstring?*

Efter diskussion med Software Arkitekt Martin Knudsen fra styregruppen, beslutter vi os for at beholde strukturen med én klasse indeholdende connection string. Andre løsninger kunne fungere, men vi fandt ikke at der ville være nogen reel håndgribelig fordel for vores system og fandt modsat nogle potentielle problemer med adgang til overflødige connection strings i tilfælde af flere database connections.

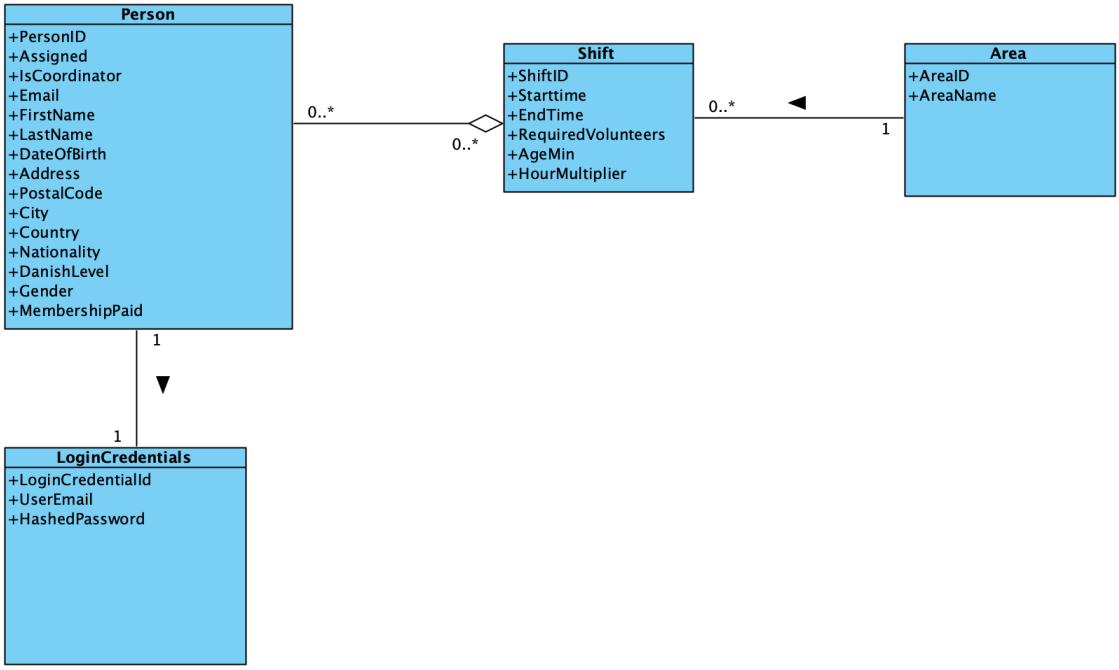
# 7. Arkitektoniske Views: 4+1

## 7.1 Logisk view

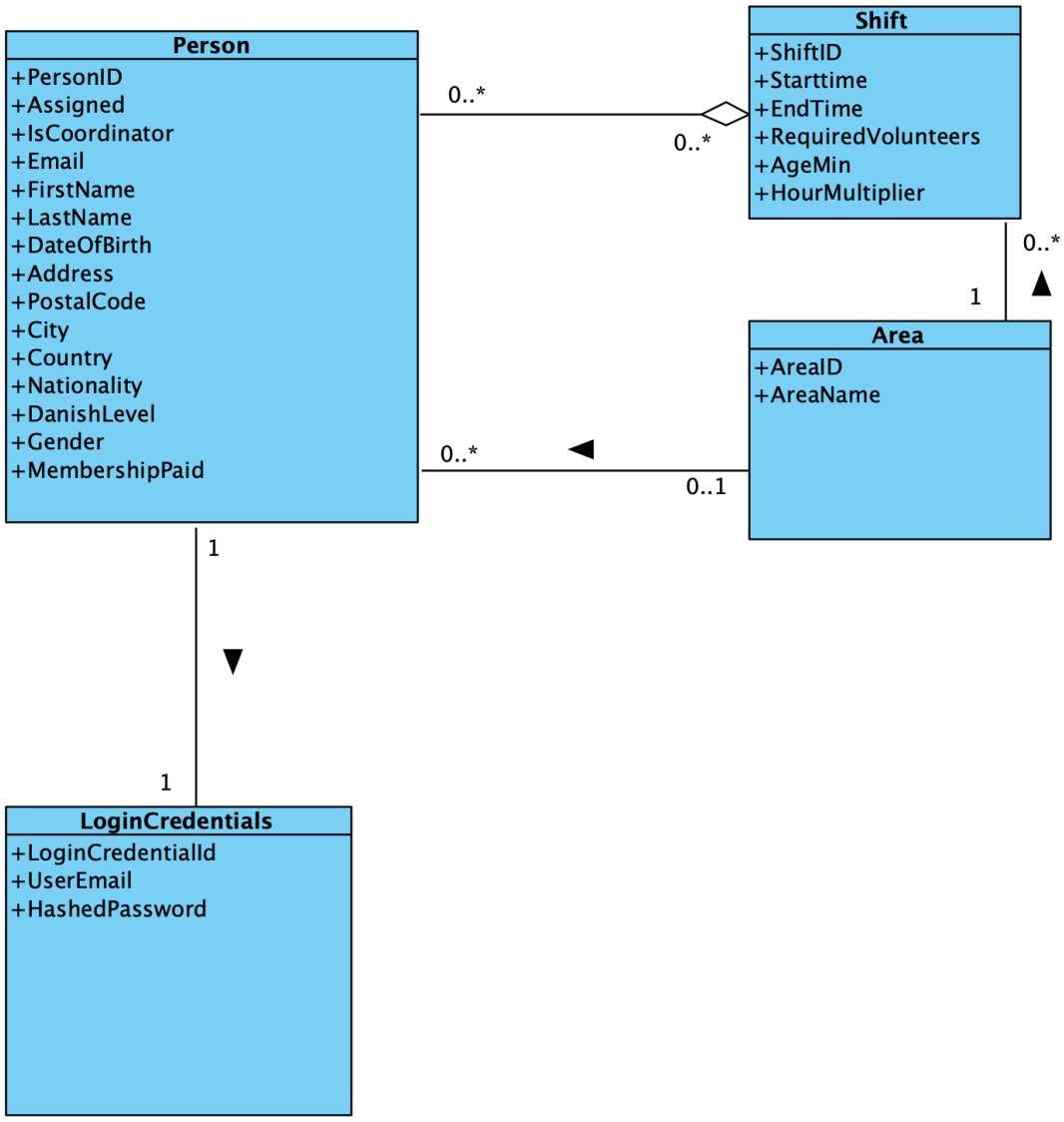
### 7.1.1 Analyse Klassediagram

Efter vi fik opbygget alle vores klasser og fundet forbindelserne mellem disse har vi tegnet vores Analyse klassediagrammet. Analyse klassediagrammet har givet hele gruppen en bredere fælles forståelse af hvordan klasserne hænger sammen i systemet(Sommer, 2020). Nedenfor ses udviklingen af klasserne gennem hvert sprint.

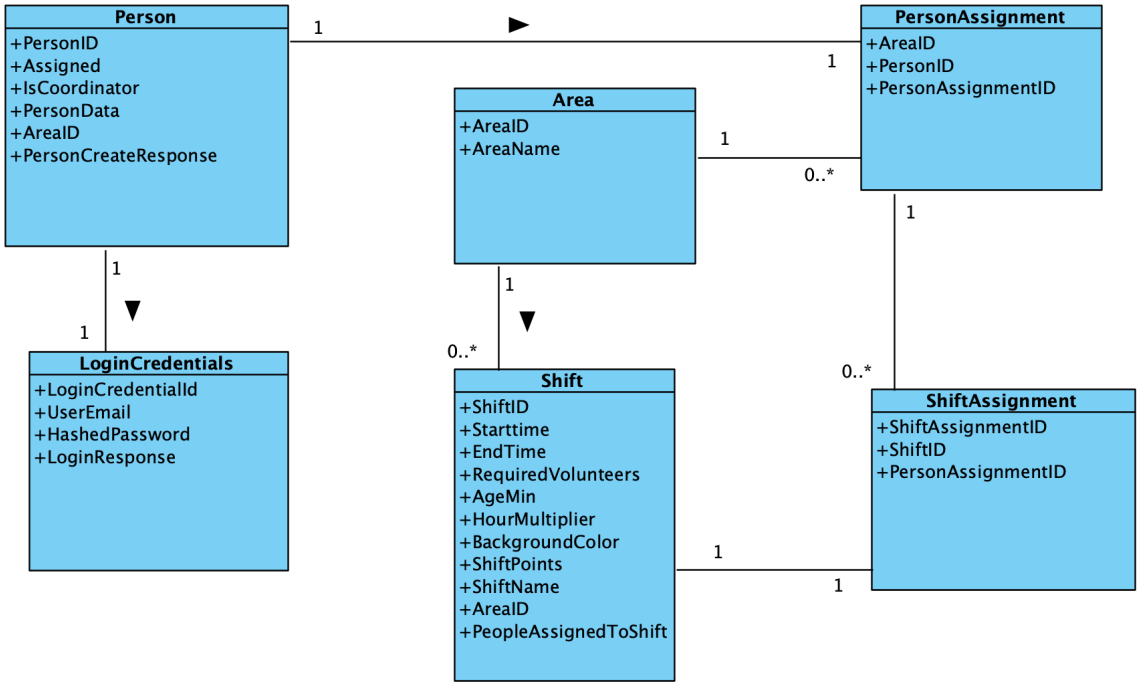
1. udkast Analyse klassediagram



2. udkast Analyse Klassediagram

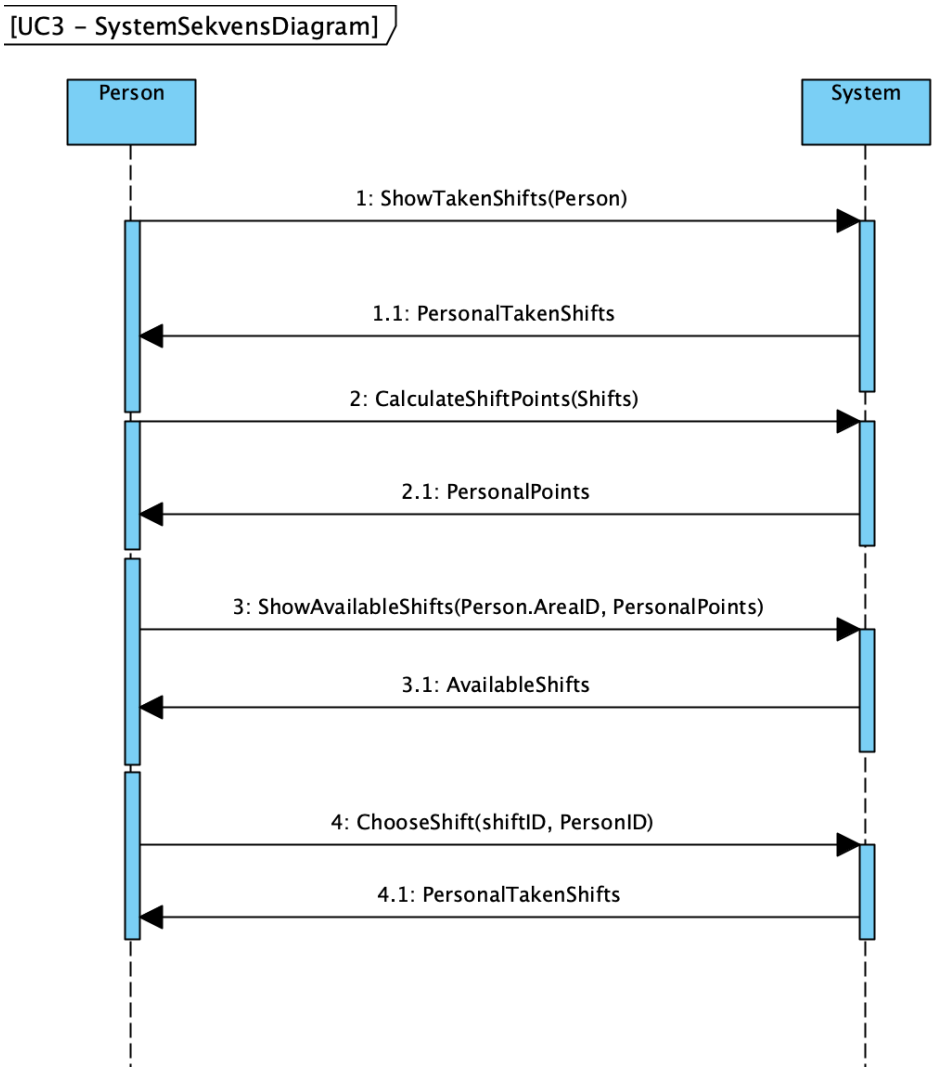


Sidste udkast & endelige version



## 7.2 Proces view

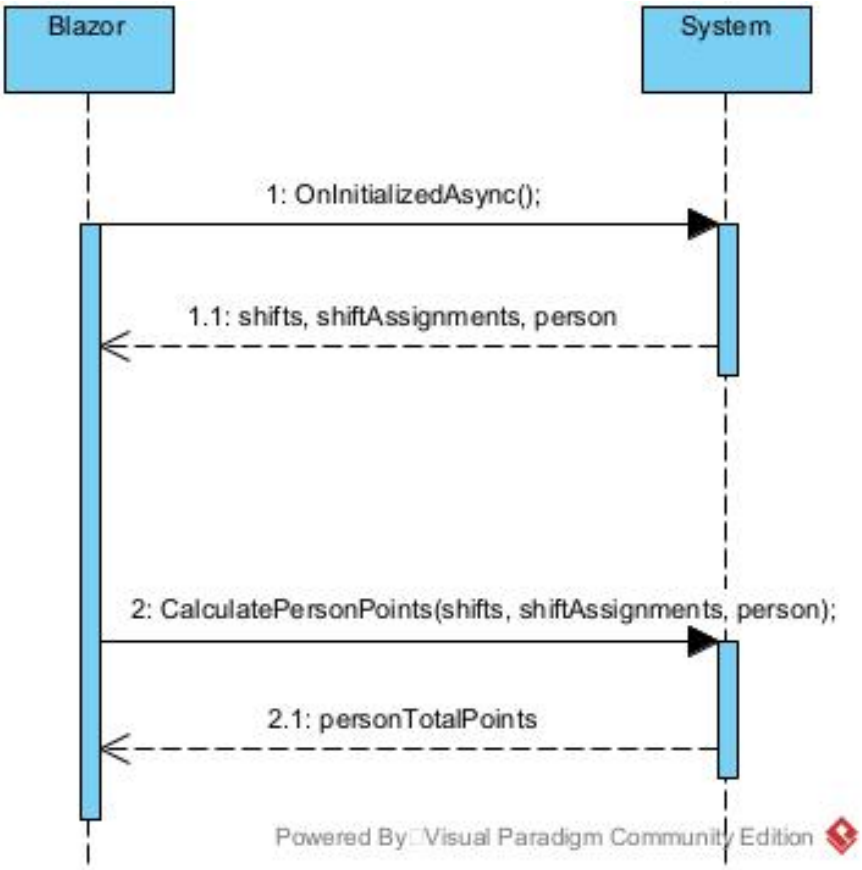
### 7.2.1 .Systemsekvensdiagram




Ovenstående diagram er skitseret hurtigt i sammenhæng med en samtale om konstruktion af metode til et pointsystem til at hjælpe den frivillige med at vælge vagter. Der ses her et systemsekvensdiagram over forløbet når brugeren skal have vist tilgængelige vagter samt vagter som vedkommende har valgt at tage. Først henter systemet de vagter som brugeren allerede har valgt. Siden bliver brugerens point beregnet ud fra længden af vagten, samt den

multiplier som en koordinater har tildelt vagten. Efterfølgende bliver vagter som er tilgængelige for brugeren vist på skærmen baseret på brugerens tilknyttede område og point. Til slut kan brugeren tilvælge de vagter som brugeren finder belejlige.

Vi har senere med udgangspunkt i den reelle arkitektur udarbejdet et simpelt systemsekvensdiagram, som ses herunder. Dette diagram bliver relativt simpelt, hvorfor vi har valgt at udarbejde et større design sekvensdiagram på samme Use Case, som findes i nedenstående afsnit.



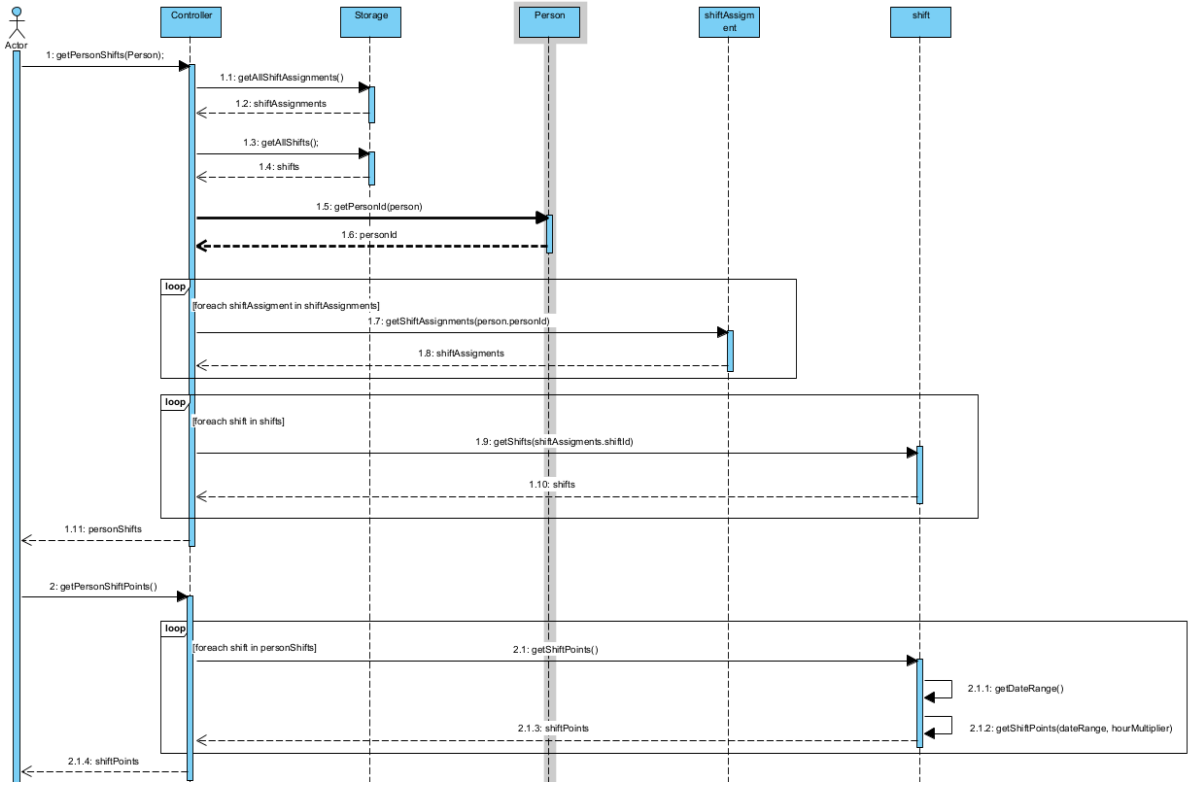

### 7.2.2 Designsekvensdiagram

Vi udviklede første udkast af følgende design sekvensdiagram lige inden vi begyndte at kode et pointsystem til frivillige. Dette tog udgangspunkt i *UC6: Read ShiftPoints* samt første udkast af vores systemsekvensdiagram. Den overordnede idé med funktionaliteten er at give den frivillige et overblik over deres nuværende vagter og en overskuelig måde at forstå hvilke

vagter ville give mening for dem at tage. Dette opnås ved at give point per vagt som en funktion af en *hourMultiplier*, hentet fra shift, ganget med time længden på shift.

I første udkast tegner vi diagrammet uden at tage hensyn til Blazor's interne arkitektur hvilket er en smule misvisende i forhold til programmets egentlige struktur da flere af metoderne vi beskriver kører som en del af OnInitializedAsync(); uden direkte input fra brugerens side. Til trods for denne mangel havde diagrammet stor værdi under vores programmerings sprint da vi fra starten havde en fælles forståelse og et relativt udførligt overblik over interaktion mellem metoder, klasser og variabler.

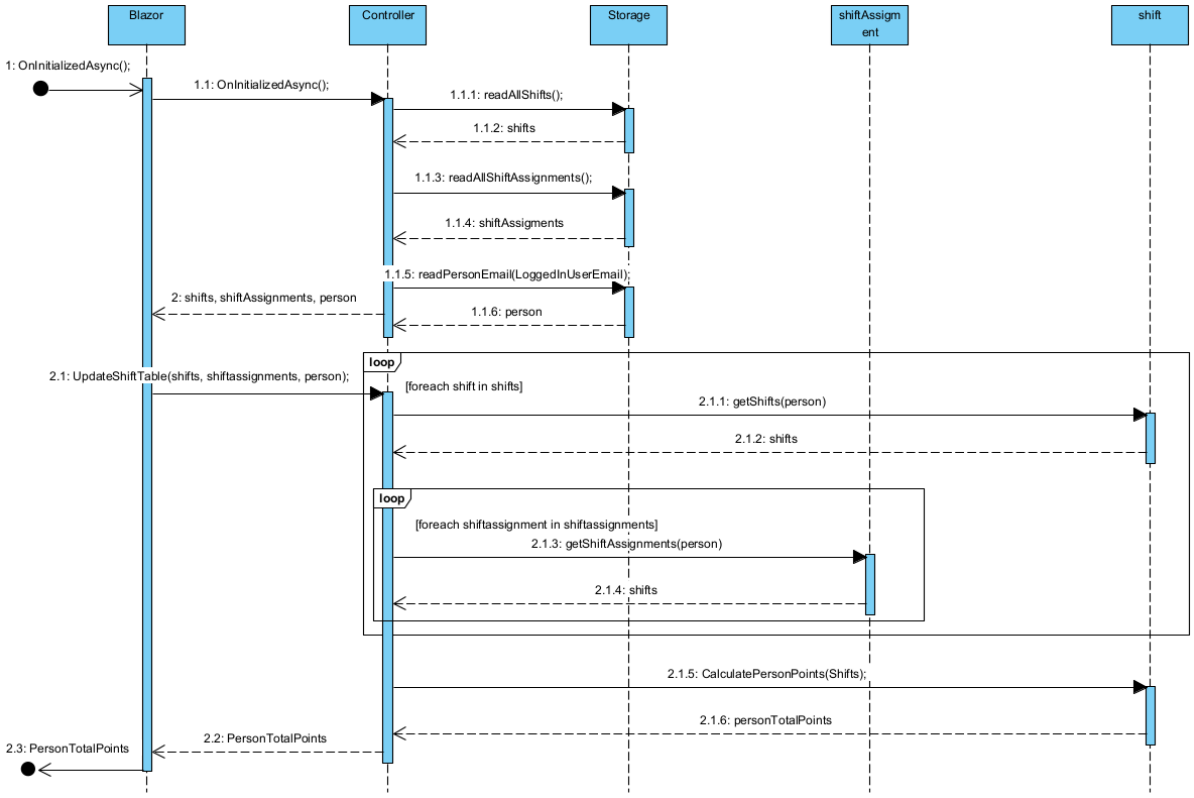
1. udkast Design Sekvensdiagram:



Sidste udkast af designsekvensdiagrammet er tegnet efter udvikling af systemet med henblik på at give et overblik over, hvad projektgruppen ser som, et af de mere komplicerede områder på applikationsniveau(Seidl, et al., 2015: kap 6).

Her tages højde for Blazor's arkitektur, hvilket resulterer i at aktøren er sat uden for diagrammet og diagrammet i stedet viser den interne kommunikation i applikationen fra *OnInitializedAsync*(); bliver kaldt når *VolunteerPage* starter til brugeren præsenteres med dennes samlede antal *shiftPoints*.

Endelige Design Sekvensdiagram:




## 7.3. Deployment view

Vi har som følge af intern samtale i projektgruppen besluttet ikke at lave et deployment diagram, da vi ikke føler at det er nødvendigt eller specielt givende for det specifikke projekt, hvilket høj grad skyldes projektet størrelse(Seidl, et al., 2015: Kap 2).

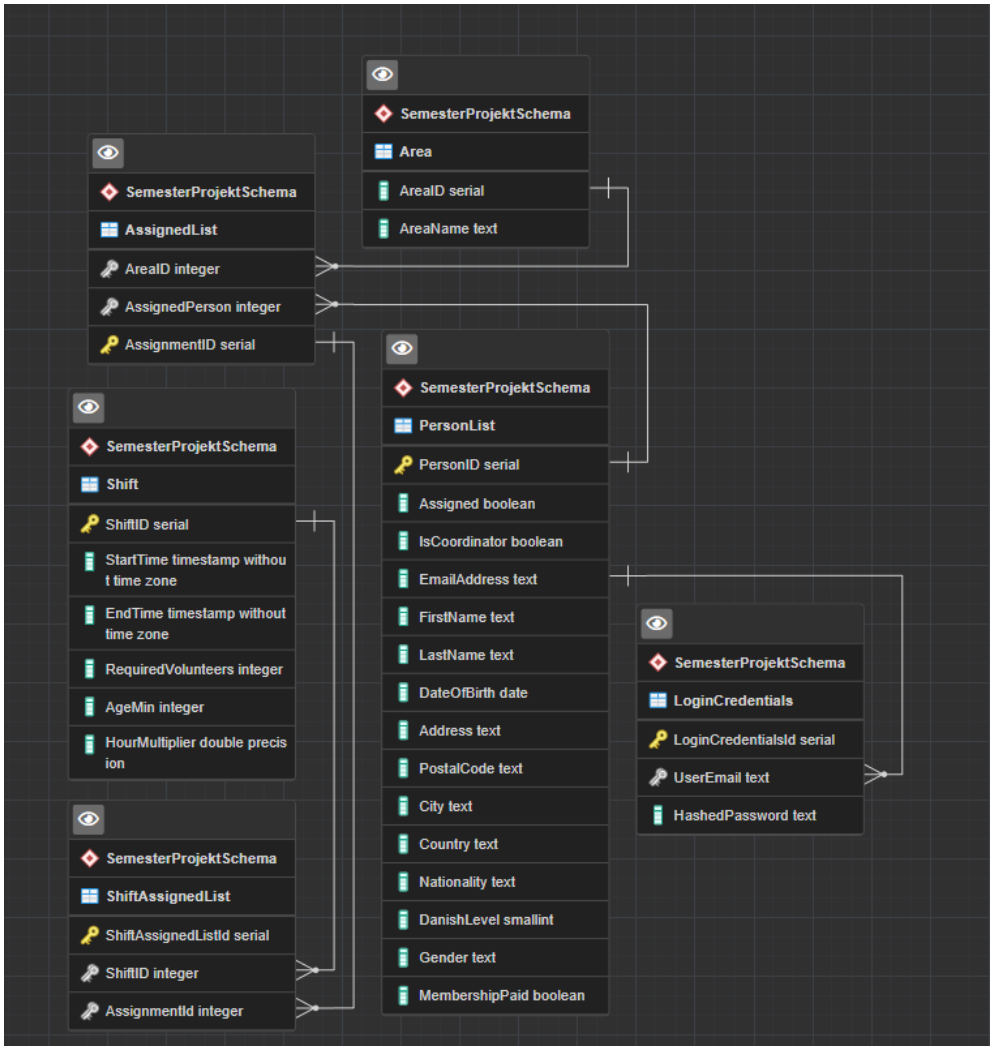
## 7.4. Data view

### 7.4.1 Entity-relationship diagram

Første udkast ERD-diagram

Første ERD-diagram laves som udgangspunkt inden alt database setup og programmering er igangsat(Meier og Kaufmann, 2019: kap 2). Vi taler herom, hvilke dele af programmet der kommer til at have brug for hvilken data. I den sammenhæng viser vi alle rækker og laver et første udkast til constraints på vores tabeller. Projektgruppen beslutter at lave separate tabeller for *AssignedList* som kan indeholde personers tilhørsforhold til arbejdsområder med henblik på mulighed for at have personer med flere arbejdsområder - Der er her enighed om at systemet i første udkast ikke nødvendigvis vil skulle have denne funktionalitet, men at det giver mulighed for fremtidig udvikling. Det noteres også at arbejdsbyrden ikke er signifikant højere.

Ligeledes behandles *ShiftAssignedList* som håndterer personers tilhørsforhold til vagter - I dette tilfælde er mange til en til mange relationen et krav for at vores system kan fungere efter hensigt, da vi har brug for at skulle have flere personer på en vagt og på samme tid flere vagter per person.

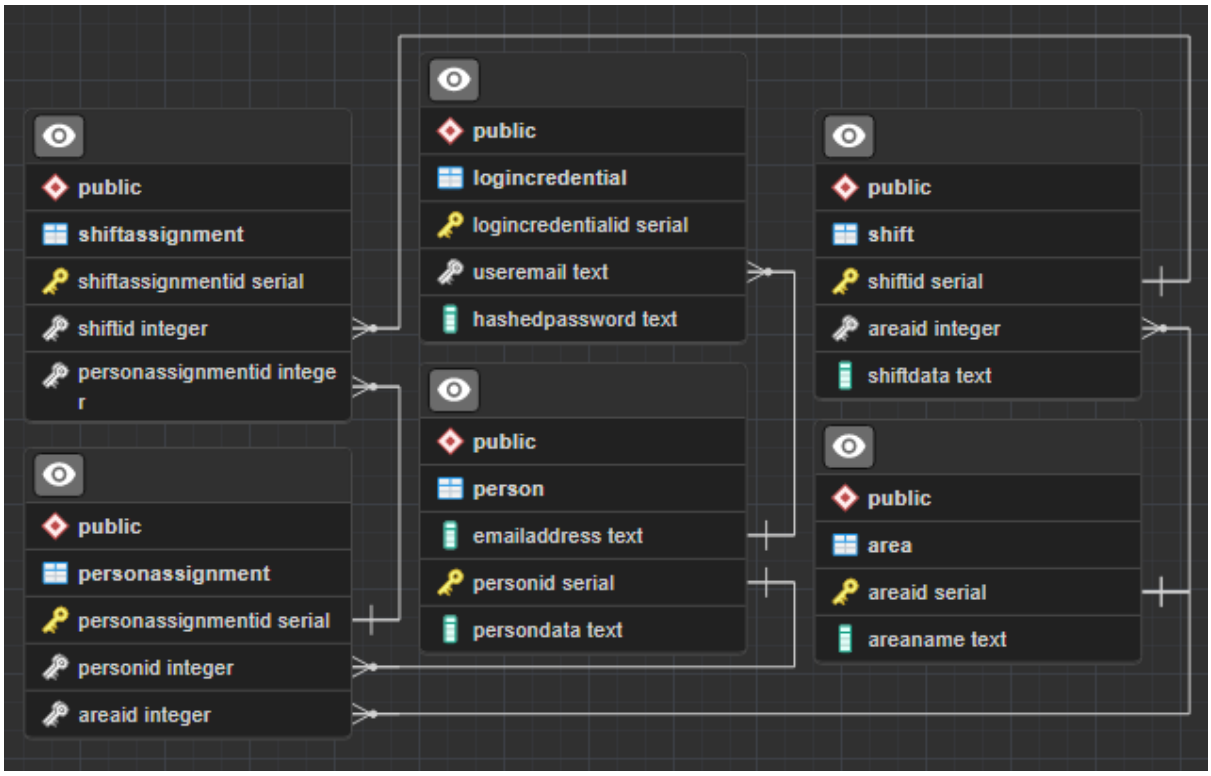



## Endelige af ERD

Endeligt udkast af ERD-diagram laves med henblik på at kortlægge den eksisterende database struktur. Der er undervejs blevet foretaget en del ændringer på databasen for bedre at passe til systemets krav, men overordnet lægger det sig relativt tæt op af det originale udkast. Den primære forskel er en direkte adgang til *areaid* på *shift* da vi ellers ville have problemer med at sortere på *shifts* da *area* ville kræve et join via. *shiftassignment*. Denne struktur var problematisk da *shifts* optimalt skal kunne eksistere uden *shiftassignments*.

Derudover har vi i sidste udkast af ERD-diagrammet samlet rækker uden direkte relevans til databasens struktur i komposit rækker. Her ses eksempelvis *persondata text* som reelt

indeholder flere rækker med persondata der anvendes på applikationsniveau, men som ikke har direkte relevans for databasens interne tilhørsforhold(Meier og Kaufmann, 2019: kap 2).



## 7.5. Use Cases view

Use cases er konstrueret på baggrund af de krav vi har fået overleveret fra styregruppen, hvor alle funktionelle krav er blevet formuleret som userstories. En komplet gennemgang af hele kravspecifikation kan findes i Projektinitieringsdokumentet(PID) under afsnit 1.4 - Krav og kravspecifikation.

### 7.5.1 User stories & Use cases

US 1:

*Som bruger skal jeg kunne logge ind i systemet, så bruger relevant data og handlinger er tilgængelig for mig.*

- UC1: Register user account

US 2:

*Som bruger skal jeg kunne lave CRUD på egne oplysninger, så brugerinfo altid er opdateret.*

- UC2: CRUD Person Data {user}

US 3:

*Som frivillig skal jeg kunne vælge hvilke vagter jeg kan tage udfra tilgængelige vagter, så jeg kan bidrage bedst muligt.*

- UC3: Select shift

US 4:

*Som frivillig skal jeg kunne se en oversigt over mine vagter, så jeg kan planlægge min tid nemt og overskueligt.*

- UC4: See all shifts

US 5:

*Som koordinator skal jeg kunne lave CRUD på vagter, så jeg løbende kan justere vagterne.*

- UC5: CRUD Shift Data {Coordinator}

US 6:

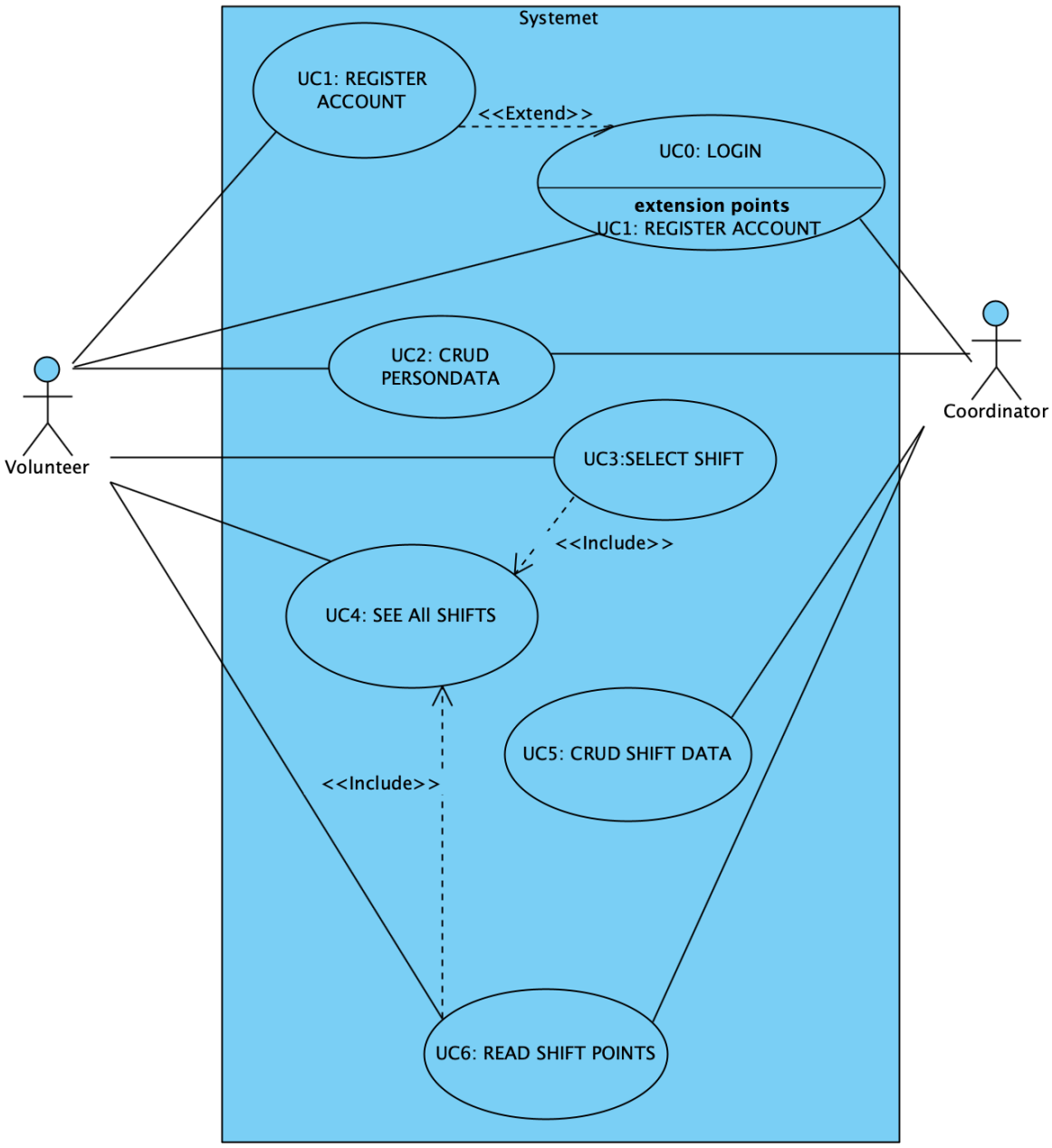
*Som koordinator skal jeg kunne markere prioriteringen af en vagt, så det er klart hvilke vagter er vigtigt at besætte.*

- UC5: CRUD Shift Data {Coordinator}
- UC6: Read Shift Points

| | |
|---|---|
| **UC0** | Login |
| **UC1** | Register user account |
| **UC2** | CRUD Person Data |
| **UC3** | Select Shift |
| **UC4** | See all shifts |
| **UC5** | CRUD Shift Data |
| **UC6** | Read Shift Points |

### 7.5.2 Use Case Diagram

Nedenstående Use Case diagram viser alle use cases i relation til de to aktører volunteer(frivillig) og koordinator. Diagrammet er løbende blevet opdateret i takt med at projektteamet har fået nye krav fra styregruppen(Sommer, 2020: kap 2).



### 7.5.2 Fuld use case beskrivelse

Vi har valgt at beskrive UC3: Select Shift i detaljen, da den har en relativt høj kompleksitet, og samtidig er en af de mest hyppige interaktioner mellem system og aktør. Derfor er en kort og mellemlang beskrivelse heller ikke tilstrækkelig til at afdække hele casen og der vil gå en del forståelse tabt når denne dokumentation læses af andre end projektteamet(Sommer, 2020: kap 2)..

| | |
|---|---|
| **Use Case Name:** | UC3: Select Shift |
| **Trigger Event:** | Den frivillige vil gerne vælge en vagt |
| ***Kort beskrivelse:*** | *Den frivillige skal vælge en vagt på FestivalCito* |
| ***Aktører*** | *Frivillig* |
| ***Related use cases*** *(includes og extends)* | UC0: Login<br>UC1: Register Data<br>UC4: See All Shifts<br>UC6: Read Shift Points |
| ***Interessenter*** | *Koordinator* |
| ***Præ-betingelse*** | *Aktøren skal være logget ind* |
| ***Post-betingelse*** | *Aktøren har valgt en vagt* |

<table>
<tr><td>Flow of events:</td><td>
<table>
<tr><td>Aktør:<br>1.Aktøren logger ind med mail & kode<br><br>2. Hvis aktøren ikke findes i systemet << extend>> UC1: Register Data<br><br>3. Aktøren vælger en vagt</td><td>System:<br>1.1 Systemet validere login oplysninger<br><br>1.2 Systemet viser aktørens side og tilgængelige vagter.<br><<include>> UC4: See All Shifts<br><br>1.3 Igennem UC4 beregner Systemet aktørens antal af point << include>> UC6: Read Shift Points<br><br>3.1 Systemet tildeler vagten til aktøren<br><br>3.2 vagten markeres rød og gemmer information i Database</td></tr>
</table>
</td></tr>
<tr><td>Exceptional flows:<br>Fejl el. ændring i interaktionen</td><td>3. Condition: Ingen tilgængelige vagter<br>- Aktøren har ikke mulighed for at vælge vagter<br><br>3. Condition: Aktøren er ikke tilknyttet et område af koordinator<br>- Aktøren har ikke mulighed for at vælger vagter</td></tr>
</table>

# 8. Fremtidig implementering

Ved fremtidig implementation ville det være fordelagtigt at fokusere på kravspecifikation IF12. Implementering af en logtabel forbundet med en trigger kan i en database sammenhæng være et nyttigt redskab. En logtabel kan genere oplysninger omkring hvem der har foretaget ændringen, hvornår ændringen er foretaget og hvilken type af ændring der er tale om. At skabe en logtabel kunne være med til at spore ændringer og aktiviteter, og det kan være fordelagtig i forbindelse med fejlfinding og problemløsning. Ligeledes kan en log hjælpe med at genskabe en gammel tilstand eller dokumentere ændringer i databasen eller bare sørge for at holde øje med om ukomplet eller fejlagtig data kommer ned i databasen.

Endnu et fremtidigt implementeringsforslag kunne være vores *ageMin* attribut. som kan være med til at sikre de frivillige der opretter sig og tager vagter er af en bestemt alder, for således at sikre at unge mennesker ikke står for udskænkning af alkohol eller andre ting der kræver en voksen frivillig.

Derudover kunne man vælge at implementere en overbooking beskyttelse, fordelen ved dette kunne være at sikre at der ikke er flere bookinger af vagter end der vagter til. Men henblik på at undgå kaos i den aktuelle situation når flere møder op til vagten, end der reelt er behov for.

# 9. Litteraturliste:

Mathiassen, Lars, et al., [n.d.]. *Object orinted analysis & design*. Forlaget Marko.

Meier, Andreas og Kaufmann, Michael, 2019. *SQL & NoSQL Databases: models, languages, consistency options and architectures for Big Data Management*. Wiesbaden [Heidelberg]: Springer Vieweg.

Seidl, Martina, et al., 2015. *UML @ Classroom: An Introduction to Object-Oriented Modeling*. 1st ed. 2015 udg. Undergraduate Topics in Computer Science. Cham: Springer International Publishing : Imprint: Springer.

Sommer, Hanne, 2020. *Objektorienteret analyse og design med UML og Unified Process*.