d. 17-02-2022

Gruppe 5.5

Medlemmer:
Josef Marc Segato | cph-jp325@cphbusiness.dk
Frederik Dinsen | cph-fd77@cphbusiness.dk
Sebastian Bentley | cph-sb287@cphbusiness.dk
Frederik Dahl | cph-fd76@cphbusiness.dk

Introduktion

Hospitaler i Danmark mangler et nyt system, hvor patienter kan vælge tid og sted efter patientens behov. Derudover skal patienten have mulighed for at aflyse deres aftale. Vi har designet en database, der kan indeholde disse aftaler. Hospitaler kan med denne database, se hvilke hospitaler, afdelinger og læger der har mest travlt, og hvilke typer patienter der oftest aflyser deres aftale.

Domænemodel

Vi har udarbejdet en domænemodel, ud fra den givne opgavebeskrivelse[1] med seks domæner. ***Hospital*** har en ***Department***, som har et ***staff***. Derudover har hospitalerne bedt om et ***System***, der har til ansvar at styre ***Patients*** og deres ***Appointments***.

Hvert hospital skal have deres eget navn og adresse, så de kan identificeres. Et hospital består af afdelinger, som har et navn, og under-afdeling som er styret af medarbejdere.

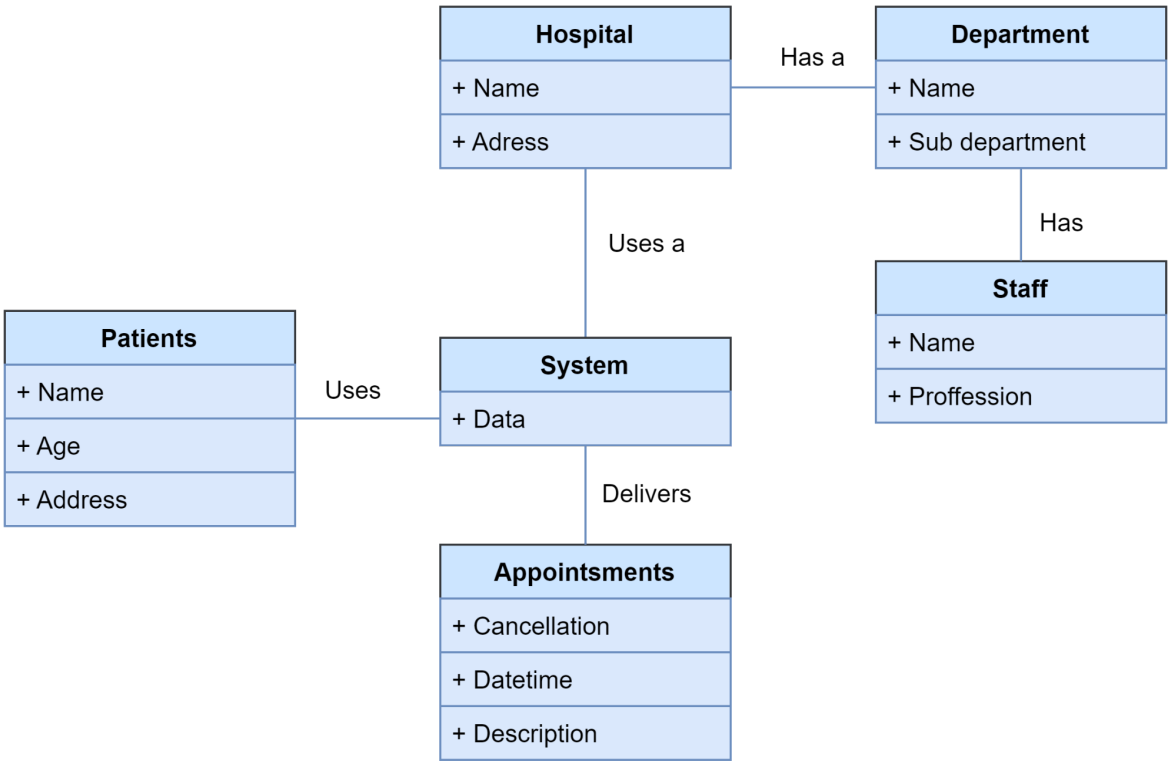
Eksempel*:* Herlev Hosptial, har afdeling Intensiv, med underafdelingen akutbehandling, som har fem medarbjedere.

Derudover har et hospital et system, hvorpå patienter kan bestille og afbestille tid, med en specifik dato og beskrivelse.

Eksempel*:* Patient Sven, går på nettet og benytter systemet, til at vælge at få en tid på Herlev Hospital. Sven har nu muligheden, for at vælge afdeling, og nærmest ledige tid. Denne information er nu gemt i databasen, og tiden er ikke ledig for andre patienter, medmindre Sven aflyser.

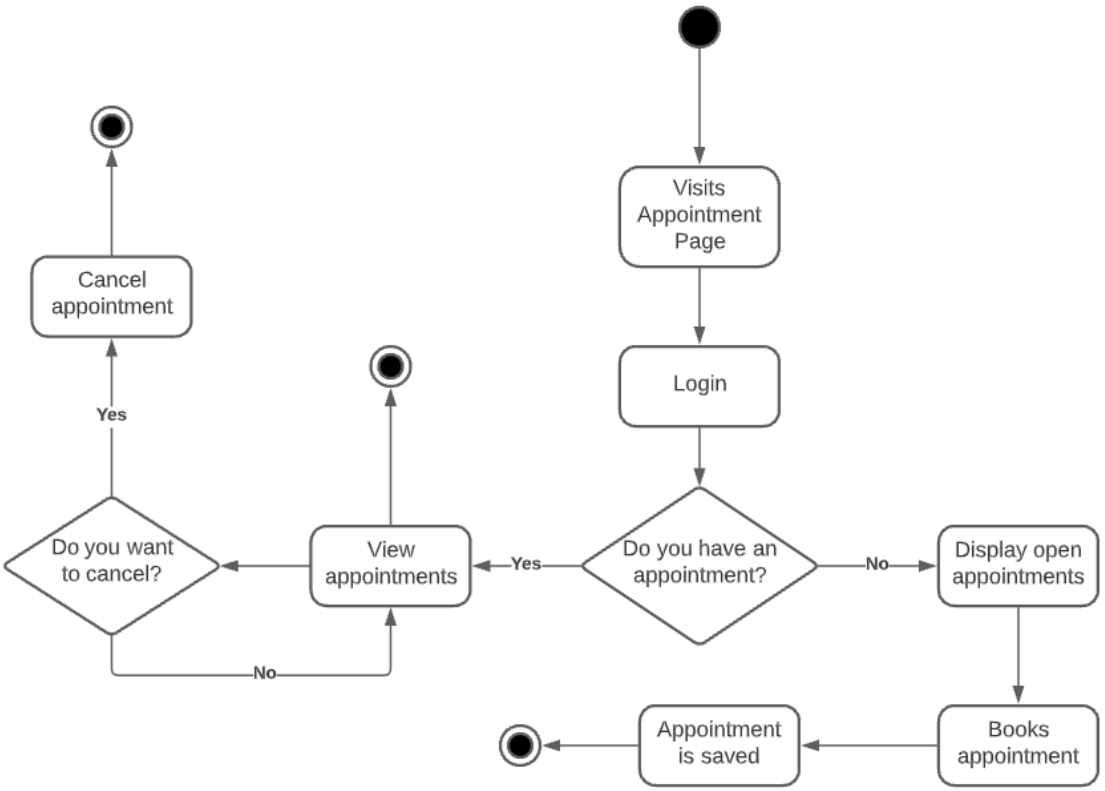
[1] https://docs.google.com/document/d/1_9RvvgK0FL9oktOKgvL1-rLQ4UQHQjKZqZj_TDp63s0/edit

d. 17-02-2022



Figur 1: Domænemodel

Flowchart for bestilling af tider - Josef

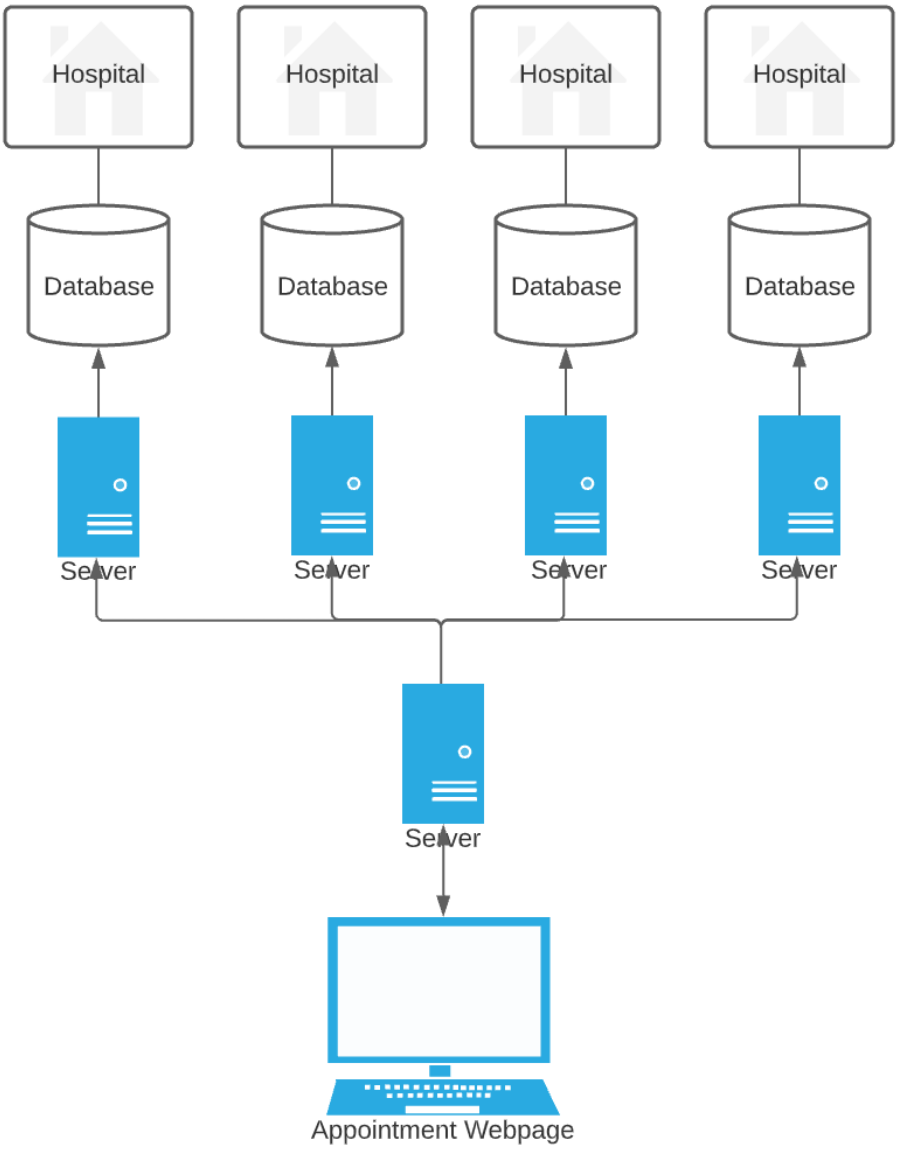


Figur 2: Flowchart

Ovenstående figur er et flowchart som vi har udarbejdet for at give et grafisk visuelt overblik over de beslutninger der skal træffes og hvilke processer der afhænger heraf. Den sorte prik er startpunktet for brugeren og pege pilene er forløbets gang. Diamant boksene repræsenterer beslutninger i form af ja / nej og de cirklede bokse er den følgende proces som sker, den hvide cirkel med sort i midten er slutpunktet for et flow. Vi har taget udgangspunkt i dette flowchart på en bruger som besøger Appointment siden. Det første skridt for brugeren er at Login, dernæst har brugeren to valgmuligheder, enten at tilgå Open appointments her kan brugeren se åbne tider og derfra bestille en ny tid. Hvis brugeren allerede har en appointment kan der tilgås View appointments herfra opstår der endnu to valgmuligheder for brugeren om enten at fjerne sin tid, eller forblive på siden.

d. 17-02-2022

Diagram over distribuering - Freddi



Vi har udarbejdet følgende model for distribuering af systemet. Tanken er at hvert hospital har deres egen database med deres egne tidsreservationer og patientinformationer. Formålet ved dette er at sprede dataen ud på de institutioner hvor den skal bruges, samt at uddele trafik byrden til flere servere.

Derudover var tanken at have en (eller flere) servere til at servere selve websiden som brugeren reservere tider på. Når brugeren så har valgt det hospital de vil bestille tid på, forbinder siden til det rigtige hospitals server.

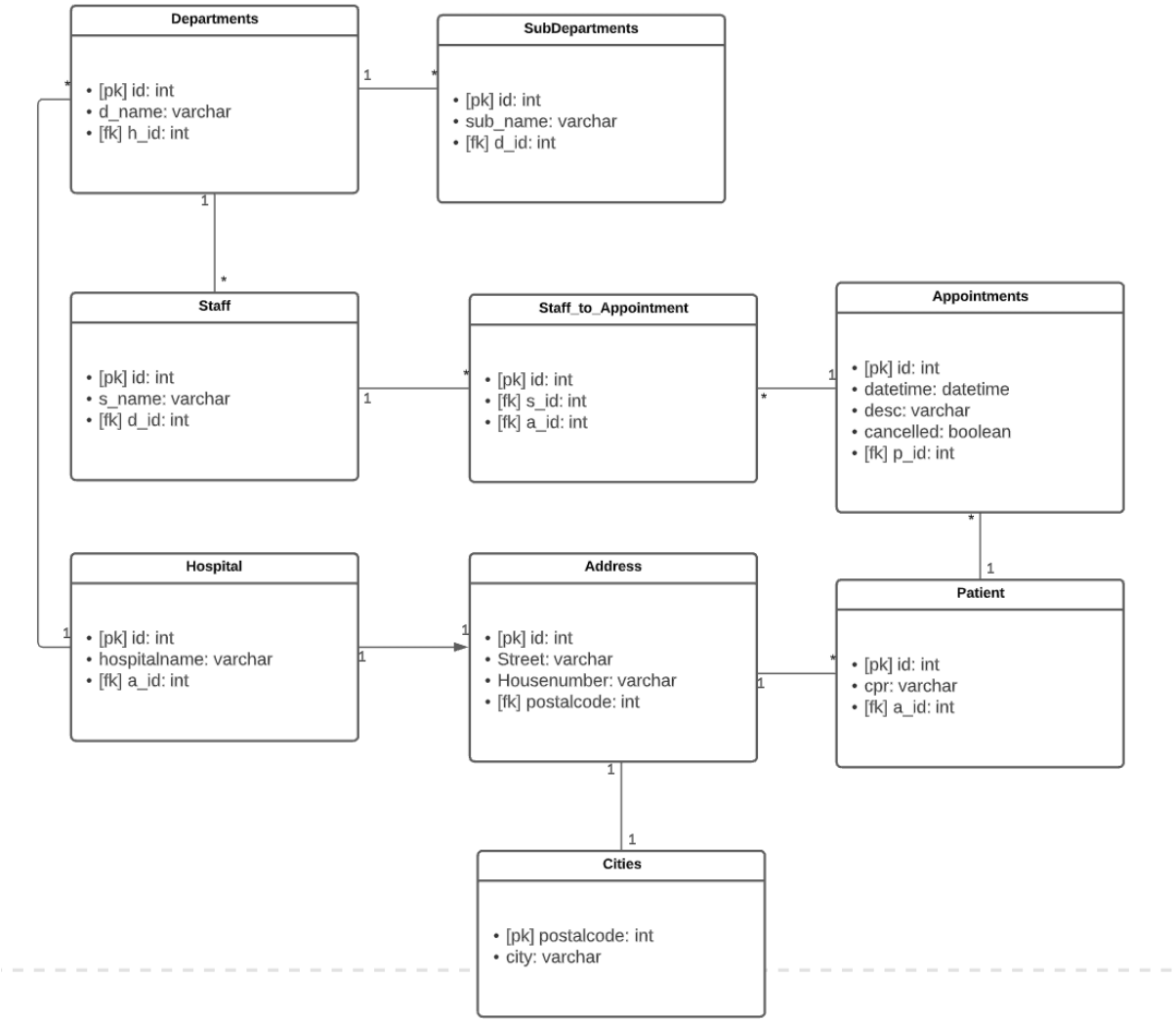
For at sikre ensartet opsætning på alle disse servere/databaser vil vi bruge Docker images til at sætte hvert miljø op.

Vi har gjort overvejelser om nødvendigheden af flytning af information mellem hospitaler, og kom til konklusionen at vi ikke ser nødvendigheden af det. Dette fordi at systemet kun holder bestilte tider, og det giver derfor ikke mening at flytte en tid til et andet hospital, da de nok ikke har en åben tid der. I stedet burde man slette sin originale tid, og oprette en ny på det andet hospital, som man ville gøre det på coronaprøver eller lægevejen.

Til gengæld ville det give mening at samle information om fx aflyste tider fra alle hospitalerne. Tanken var at dette kunne gøres fra en central server som forbinder til alle hospitalerne, og anmoder om alle aflyste tider.

d. 17-02-2022

Database Diagram



Ovenstående database diagram er en illustration af hvorledes vi ville strukturere databasen. Idet vi benytter en relationel database er der relationer mellem tabellerne. Disse relationer benyttes til at opbevare data på tværs af tabellerne i stedet for at opbevare alt data i en enkelt tabel. Dette gøres for at gøre tabellerne mindre og mere overskuelige.
I tabellen fremgår det at Departments-tabellen har en 1-* relation til SubDepartments-tabellen. De to tabeller er struktureret på samme måde og indeholder begge et id, navn og et id som refererer til enten et hospital / en afdeling. Hospitals-tabellen har en 1-* relation med Departments-tabellen, men udover det er strukturen som førnævnte tabeller.
I midten af diagrammet ses Address-tabellen, denn her en 1-1 relation med Hospital og Cities-tabellen. Sidstnævnte benyttes til at opbevare bynavn til et postnummer. Address-tabellen har også en 1-* relation til Patient-tabellen. Dette skyldes at en adresse kan have mange patienter, fx en familie som bor på adressen.

Appointments-tabellen indeholder et id, en dato, beskrivelse, om den er blevet aflyst og et patient ID. Patient ID er en fremmednøgle fra Patient-tabellen hvor til Appointments har en *-1 relation.

Tilknyttet en aftale personalet på afdelingen og der er oprettet en mellem tabel, Staff_to_Appointment, som indeholder et id, et id til hvilket personale som er tilknyttet aftalen samt et aftale id. Ved at benytte en mellem tabel kan man let og overskueligt undersøge hvilke aftaler en fra personalet er tilknyttet eller omvendt med patienter.
Mellem tabellen har en *-1 relation til de tabellerne den forbinder.

Overvejelserne bag dette design af databasen har været at holde antallet af tabeller med irrelevant data til et minimum og i stedet kunne hente yderligere data ved hjælp af transaktioner i databasen. Det vil altså sige at de tabeller som denne database indeholder kun må indeholde relevant data som passer til tabellens navn. Fx vil Hospital-tabellen kun indeholde information om et hospital, som fx id, navn og adresse id. Den fulde adresse vil være opbevaret i Address-tabellen, men da vi har adresse id'et er det hurtigt at finde hele adressen med den information der er tilgængelig om et givent hospital.

Dette betyder at vi overholder tredje normalform idet den overholder forskrifterne for dette som er:

- Ingen kolonne må gentages en andens værdi (atomare værdier)
- Hvis en tabel har en sammensat nøgle skal alle felter der ikke indgår i nøglen afhænge af den samlede nøgle (ingen partiel funktionelt afhængighed)
- Ingen felter uden for primærnøglen der er indbyrdes afhængige (ingen transitive funktionelle afhængigheder mellem non-prime attributter).

Postgresql to create all tables:

```
CREATE TABLE if not exists cities (
  postal_code int primary key ,
  city varchar(100)
);

create table if not exists adress (
        id SERIAL primary key,
        street varchar(100),
        house_number varchar(20),
        postal_code int,
        constraint postal_code
        foreign key (postal_code)
                references cities(postal_code)
);

create table if not exists patient (
        id SERIAL primary key,
        cpr varchar(11),
        a_id int,
        constraint a_id
        foreign key (a_id)
                references adress(id)
```

```
);

create table if not exists hospital (
	id SERIAL primary key,
	hospitalname varchar(255),
	a_id int,
	constraint a_id
	foreign key (a_id)
		references adress(id)
);

create table if not exists departments (
	id SERIAL primary key,
	d_name varchar(255),
	h_id int,
	constraint h_id
	foreign key (h_id)
		references hospital(id)
);

create table if not exists subdepartments (
	id SERIAL primary key,
	sub_name varchar(255),
	d_id int,
	constraint d_id
	foreign key (d_id)
		references departments(id)
);

create table if not exists staff (
	id SERIAL primary key,
	s_name varchar(255),
	d_id int,
	constraint d_id_two
	foreign key (d_id)
		references departments(id)
);

create table if not exists appointments (
	id SERIAL primary key,
	datetime timestamp,
	description varchar(255),
	cancelled boolean,
	p_id int,
	constraint p_id
	foreign key (p_id)
		references patient(id)
);
```

```
create table if not exists staff_to_appointment (
        id SERIAL primary key,
        s_id int,
        a_id int,
        constraint s_a_relation
        foreign key (s_id)
                references staff(id),
        foreign key (a_id)
                references appointments(id)
);
```