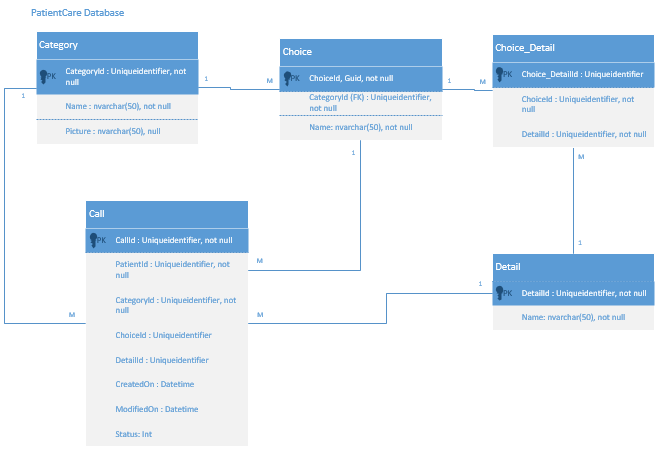
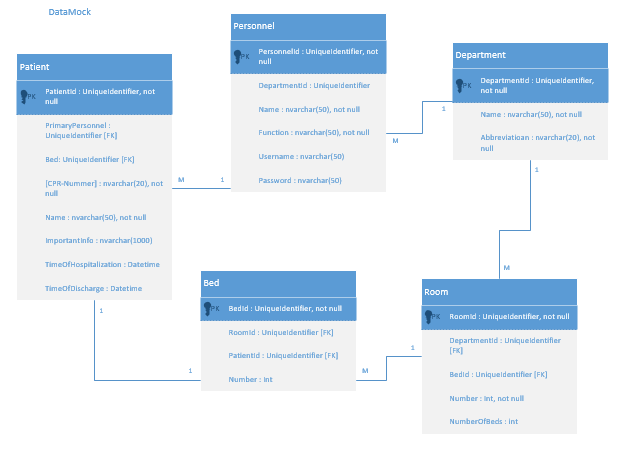
Til systemet skal der bruges en database der kan holde styr på det data der udveksles mellem systemets apps.

# Datastruktur

Datastrukturen for det data vi selv genererer i PatientCare er vist herunder. Denne datastruktur er lavet ud fra den relationelle tankegang.



Dertil er der lavet en datastruktur for datamock. Datamocken er lavet ud fra de data som vi ønsker at gøre brug af fra et andet eksisterende system som Cetrea har udviklet.



Her gives et eksempel på datastrukturen i PatientCare:  
Den data der bliver sendt når en patient opretter et kald via PatientApp indeholder IKKE alle de parametre der er krævet for visningen af et kald på personalesiden. Der er derfor behov for at hente data et andet sted fra i databasen og knytte det til kaldet, inden det via webAPI’et sendes videre til personalet. Det drejer sig blandt andet om informationer som; hvilket personale der er tilknyttet patienten og hvor kaldet kommer fra (hvilken seng/stue/afdeling patienten er på/i).

Valg af database

Det blev besluttet at bruge en relationel database og lagre data i tabeller med kolonner og rækker. Beslutningen om at gøre brug af en relationel database blev taget ud fra det faktum at alle i gruppen har berørt det i IT-fag i løbet af uddannelsen og fordi datastrukturen til applikationerne i PatientCare passer fint til den relationelle datastruktur.

Her gives et eksempel på datastrukturen i PatientCare:  
Den data der bliver sendt når en patient opretter et kald via PatientApp indeholder IKKE alle de parametre der er krævet for visningen af et kaldet på personalesiden. Der er derfor behov for at hente data et andet sted fra i databasen og knytte det til kaldet, inden det via webAPI’et sendes videre til personalet. Det drejer sig blandt andet om informationer som; hvilket personale der er tilknyttet patienten og hvor kaldet kommer fra (hvilken seng/stue/afdeling patienten er på/i).

Da det som nævnt er oplagt at benytte en relationel database til dette besluttede gruppen at gøre brug af en sådan. Projektgruppen fik en studielicens til Microsoft Azure som blandt andet tilbyder brug af MySQL databaser på en database server. Her oprettede gruppen tabeller i overensstemmelse med den datastruktur som var fastlagt for PatientCare. På Azure platformen fik gruppen tildelt 650 danske kroner per måned som ’credits’.

Efter ca. 2 uger efter oprettelsen af studiekontoen lukkede alt på Azure ned fordi alle ’credits’ var opbrugt. Det viste sig at alle credits var opbrugt fordi vi fra start af havde kommet til at foretage et forkert valg med hensyn til valg af databasetype. Under import af databasen fra localdb til Azure, har vi ikke været opmærksomme på at den databasetype som blev valgt, koster en masse penge og har en masse ressourcer som vi ikke havde behov for, men kom til at betale for. Vi havde kontakt til Microsoft og kunne konstatere at subscription var udløbet og databaserne med alle vores data var blevet slettet.

På det tidspunkt skulle der træffes et hurtigt valg om hvordan vi kunne løse problemet. Gruppen snakkede med IHA-vejleder om at få en ny Microsoft Azure konto, men blev enige om at det kun kunne afhjælpe problemet midlertidigt, eftersom vi ville støde ind i samme problem igen. Derfor valgte gruppen at kigge andre veje.

Vi valgte at benytte os af en anden database, nemlig MongoDB. Tankegangen med datastrukturen er ikke helt den samme i MongoDB som for relationelle databaser. Med MongoDB tænker man ikke normalisering men i stedet gemmes hele objekter ned i collections. MongoDB er en NOSQL database som godt kan bruges som relationel database, men er som udgangspunkt ikke beregnet til at hente data ind fra forskellige collections(tabeller), da MongoDB ikke er tænkt på den måde. Derudover var MongoDB på det tidspunkt ukendt for alle i gruppen, men på grund af nedbryd af Azure og at der skulle træffes en hastig beslutning valgte vi denne vej.

På efterkant kan man diskutere om MongoDB var det rigtige valg for PatientCare. Efter at have erfaret hvad MongoDB er, og hvordan det bruges kan vi konkludere, at hvis vi havde tænkt at finde en anden løsning med mySQL database, udenom Azure kontoen, havde det nok været et klogere valg, af den simple årsag at tankegangen med datastrukturen matcher PatientCares behov bedre.

# Overblik over systemet



I Web API’et benytter vi os af et Repository pattern der gør at vi forholdsvis nemt kan udskifte vores database hvis der skulle blive behov for det. En anden ting er at det gør det nemt at teste systemets controllers da de ikke tilgår databasen direkte, men tilgår et repository som kan mockes ud i en unittest.

Databasen forbindes til resten af systemet gennem Web API’et og da Web API’et er et webinterface til resten af systemet spiller det ingen rolle for vores apps hvilken database der ligger bag ved Web API’et, dog vil man på disse devices opleve at der kan opstå fejl hvis data ikke kan gemmes. Web API’et modtager og sender data i json format til og fra vores apps. Det har ingen betydning for databasen, da omdannelsen af json format til det format som databasen bruger, sker i web API’et.   
Om det er mySQL, Bson eller et tredje sprog har ingen betydning for brugerne af systemet.  
I forbindelse med skiftet af databasen er det derfor ’kun’ en ny implementation i Web API’ets repository pattern der gør at vi nu kan gemme data i en anden database.