

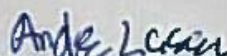


PatientCare

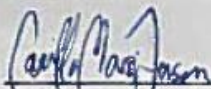
Patientkald med tilknyttet årsag
Bachelorprojekt
Efterår 2015



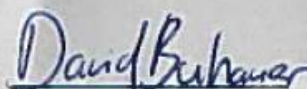
AARHUS
UNIVERSITET
INGENIØRHØJSKOLEN



Informations- og kommunikationsteknologi
Anders Junker Lassen
201270933



Sundhedsteknologi
Camilla Marsi Jensen
201271010



Informations- og kommunikationsteknologi
David Buhauer
201270749



Sundhedsteknologi
Minna Thomsen
201270690

1 Abstract

Introduction

More than half a million people was hospitalized in Denmark in 2014. Most people are in need of care during hospitalization and contacts nurses using the call bell located beside the hospital bed. The call bell triggers an alarm that indicates a patient call. For many years, patients have used the call bell to communicate with nurses. The bell symbolizes comfort and safety for a patient during the period of hospitalization.

Aim

It is not possible to associate a cause with the patient call by using the call bell today. The aim of this bachelor project is to develop a prototype that makes it possible to associate a cause with a patient call, as well as investigate the need for such a system. The bachelor project clarifies in what manner the nurses can reduce the number of steps and save time on each patient call by making use of PatientCare. The importance of a patient's ability to specify a need for help is also examined.

Materials and methods

During the development process of the prototype the bachelor team performed iterative development methodologies. As the need of a system like PatientCare was investigated the functional requirements was identified. A requirement specification and an acceptance test has been produced during the development period and simultaneously a mini version of a health technology assessment (Mini-HTA) which is designed to examine the preconditions and consequences of implementing the PatientCare system in a chosen department at the Regional Hospital of Randers. Healthcare professionals have been involved through the progress to clarify the needs.

Results

The project has resulted in developing a prototype for the PatientCare system involving three modules including a WebAPI which provides communication in between. One is a solution to patients in the form of an App that can be downloaded on the patients' own smartphones, from which they can choose between some predefined options and send a patient call to the staff with a associated cause. The second is a solution for the staff in the form of an App from which staff can receive patient call, see the cause for the call as well as from whom when and how the patient is positioned on the ward. The third is an administrative module where you can customize the predefined options for patients as they are department-specific and thus more likely to hit the needs of patients.

Results from the Mini-MTV study shows that the staff is better prepared to perform the task that comes with the patient call and reduce half as much time on a patient call and reduce the number of steps significantly. This is because they do not have to walk back and forth to find out the cause of the call as they are compelled today with the existing call system.

Conclusion

The need for a system that associates a cause with a patient call is highly requested at the chosen department at the Regional Hospital of Randers.

The team has developed a prototype for PatientCare that makes it possible to associate a cause with a patient call

1 Resume

Indledning

I 2014 blev over en halv million danskere indlagt på de danske hospitaler. De fleste patienter har behov for pleje igennem indlæggelsesforløbet og tager kontakt til personalet ved at trække i kaldesnoren som de fleste steder er placeret ved patientens sengeplads. På den måde udløses en alarm som udtryk for et patientkald. Kaldesnoren har fungeret som et kommunikationsredskab mellem plejepersonale og patienter i mange år. Den har til formål at være bindeled mellem sygeplejersker og patienter og virker som et symbol på tryghed for en patient under indlæggelse.

Formål

Som det er i dag er det ikke muligt at knytte en årsag til patientkaldet gennem kaldesnoren. Formålet med dette bachelorprojekt er både, at udvikle en prototype af et system kaldet PatientCare, der gør det muligt at knytte en årsag til patientkald, men også at undersøge behovet for et sådant system. Bachelorprojektet belyser blandt andet i hvilken grad personalet kan spare unødvendige skridt og tidsforbrug på patientkald ved at gøre brug af PatientCare systemet. Det bliver også undersøgt hvilken betydning det har for patienter at de kan specificere deres behov for hjælp.

Materialer og metoder

For at gennemføre udviklingen af prototypen har bachelorteamet arbejdet med en iterativ udviklingsproces i takt med at behovet for systemet blev undersøgt og de funktionelle krav identificeret. Der er udarbejdet en kravspecifikation, accepttest og et designdokument i forbindelse med udviklingen af systemet. Samtidig er der udarbejdet en Mini-MTV, der har til formål at undersøge forudsætningerne for og konsekvenserne af at indføre PatientCare systemet, på en udvalgt afdeling på Regionshospitalet i Randers. Det har været en brugerdriven proces, hvor sundhedsprofessionelle har været involveret igennem store dele af forløbet.

Resultater

Projektet har resulteret i udvikling af en prototype af PatientCare systemet bestående af tre moduler samt et WebAPI der sørger for kommunikationen mellem dem. Det ene er en løsning til patienter i form af en app, der kan downloades på patienternes egne smartphones, hvorfra de kan vælge mellem nogle foruddefinerede valgmuligheder og sende et patientkald afsted til personalet med tilknyttet årsag. Det andet er en løsning til personalet også i form af en app, hvorfra personalet kan modtage patientkald, se årsagen til kaldet samt hvilken patient der har sendt det hvornår og hvor patienten er placeret på afdelingen. Det tredje er et administrativt modul hvorfra man kan tilpasse de foruddefinerede valgmuligheder til patienterne, så de bliver afdelingsspecifikke og dermed med større sandsynlighed rammer patienternes behov.

Resultater fra Mini-MTV-analysen viser at personalet kan forberede sig bedre på at udføre den opgave der følger med patientkaldet og sparre halvt så meget tid på et patientkald samt reducere antallet af skridt markant. Dette skyldes at de ikke behøver at gå ind til patienten først for at få information om årsagen til kaldet og ofte gå tilbage igen for at hente det efterspurgte, som de er nødsaget til i dag med det eksisterende kaldesystem.

Konklusion

Behovet for et system der kan knytte en årsag til patientkald er stor på den udvalgte afdeling på Regionshospitalet i Randers

Projektgruppen har udviklet en prototype for PatientCare, som viser de centrale funktionaliteter for at knytte en årsag til et patientkald.

Indholdsfortegnelse

2	Indledning	5
	Formål	5
	Baggrund	5
	Projektbeskrivelse	6
	Vision	7
3	Afgrænsning	8
	Funktionalitet	8
	Afdeling	9
	Integrationer	10
4	Systembeskrivelse	12
	Aktørbeskrivelse	14
5	Materialer og metoder	15
	Udviklingsproces	15
	Udviklingsdokumentation	15
	Tidsplan	15
	Agil systemudvikling	16
	Mini-MTV	17
	Projektstyring	19
	Møder og eksternt samarbejde	20
	Samarbejde med Systematic	20
	Samarbejde med sundhedsfaglige	21
6	Krav	22
7	Systemarkitektur	26
	Kommunikation mellem moduler	26
	Data	26
	Lokal data persistering	28
	Trelagsarkitektur	28
	PatientApp	29
	PersonaleApp	31
	AdminApp	31
	WebAPI	33
8	Design	35
	PatientApp	35

PersonaleApp.....	38
AdminApp.....	42
WebAPI.....	44
Database.....	44
9 Resultater	47
Resultater fra Mini-MTV.....	47
Status på prototype	48
PatientApp.....	48
PersonaleApp.....	49
AdminApp	50
WebAPI	51
10 Diskussion	52
Mini-MTV	52
Databaseskift	52
Android Studio og GitHub.....	53
JSON objekter vs. tekststreng.....	53
Xamarin.....	54
Hvorfor er et system som dette ikke lavet før?.....	55
12 Konklusion	56
13 Perspektivering	57
14 Referencer	59

2 Indledning

I Danmark har vi 54 offentlige sygehuse og på disse arbejder der rundt regnet 100.000 fuldtidsansatte (1). På sengeafdelingerne er det blandt andet sygeplejerskerne som dagligt tager sig af plejen af patienterne og de administrative opgaver der følger med. Sygeplejersker har generelt travlt med mange forskellige opgaver og bliver ofte afbrudt i løbet af deres arbejdsdag. Afbrydelserne skyldes blandt andet henvendelser fra andre sygeplejersker, alarmer, patientkald, telefonopkald og stuegang (2).

Når man er indlagt som patient på et hospital og vil have fat i personalet på distancen, foregår det i dag ved at patienten trækker i kaldesnoren som er placeret ved patientens sengeplads. På den måde udløses et patientkald. I 2014 blev over en halv million (3) danskere indlagt og der er derfor mange der benytter sig af kaldesnoren som findes på mange sengeafsnit på de danske hospitaler. Kaldesnoren har fungeret som et kommunikationsredskab gennem mange år. Den har til formål at være bindeled mellem sygeplejersker og patienter. Kaldesnoren virker som et symbol for tryghed og sikkerhed for en patient under indlæggelse. Når patienten trækker i snoren udløser det en alarm til afdelingens personalet, som derefter tager sig af patientens behov. Når personalet får besked om alarmen, vides grunden til at patienten trækkede i snoren ikke, før personalet har været inde ved patienten for at finde ud af hvad årsagen var. Kaldet kan skyldes alt fra, at patienten befinder sig i en livstruende tilstand til, at patienten beder om en serviceydelse som toiletbesøg eller et glas vand. I enkelte tilfælde er det muligt for personalet at have en ide om hvad der har været grund til patientkaldet, men som udgangspunkt kan de aldrig vide hvad der er i vente. I tilfælde hvor årsagen ikke er livstruende, bruger sygeplejerskerne nødvendige skridt på at gå ind på stuen for at få information fra patienten om kaldets årsag og skal ofte ud af stuen for at hente det efterspurgte og tilbage til patienten på stuen igen. Personalet kan opleve at skulle gå fra den ene ende af sengeafsnittet til det andet og sommetider er personalet nødt til at gå hele vejen tilbage for at hente noget der opfylder patientens behov. I løbet af en arbejdsdag kan det hurtigt blive til mange skridt for personalet som kunne undgås, hvilket der med dette bachelorprojekt forsøges at gøre noget ved.

Formål

Formålet med projektrapporten er at give et overordnet indblik i bachelorprojektet som helhed og belyse processen og resultaterne samt de overvejelser der har været gennem projektforløbet.

Baggrund

I den tidlige opstartsfase før bachelorprojektets begyndelse faldt en af projektgruppens medlemmer over en artikel (4) på nettet omhandlende **innovativ sporbarheds-it på vej til danske hospitaler, hvor en aftale mellem Region Midtjylland og Systematic lige var blevet indgået**. Ifølge artiklen bruger en medarbejder på et dansk hospital i gennemsnit 12 minutter pr. vagt på at lede efter kollegaer og udstyr på hospitalet. Ved at indføre teknologi, der kan spore og identificere mennesker og udstyr, er der potentiale for at spare tusinder af timer hver dag på de danske hospitaler. I artiklen stod der også at løsningen i første omgang skulle implementeres på Det Nye Universitetshospital i Skejby i sommeren 2015. Der var potentiale for at indgå i et projekt, hvor kompetencerne fra både IKT-studerende og sundhedsteknologistuderende kunne udnyttes. Gruppemedlemmet tog kontakt til de tre andre og der blev arrangeret et møde med Systematic for at høre nærmere om projektet. På mødet blev projektgruppen introduceret til den **samlede service logistik-løsning** fra Systematic kaldt C. Den består af en række **individuelle løsninger** til opgaveløsning, søgning efter personer eller udstyr, sengelogistik og vognlogistik. Hver løsning kan anvendes hver for sig, men er også **integreret med andre kliniske systemer**. Til mødet blev det sagt at det projekt som vi havde læst om i artiklen om-

kring sporbarhed allerede var et pilotprojekt i implementering, hvorfor det ville være mere relevant for os at arbejde med et nyt projekt der både krævede behovsundersøgelse og udvikling af software. Systematic havde en række cases liggende i form af inputs fra sundhedsprofessionelle som de ønsker at få løst. Deriblandt et ønske fra plejepersonalet om at reducere antallet af skridt ved at kende årsagen til patientkaldet når patienterne kalder på personalet. Denne case valgte projektgruppen at arbejde videre med. Opgavesystemet, *Columna Service Logistics*, håndterer i dag serviceopgaver som portørerne kan tage sig af som fx transport af patienter på tværs af afdelinger. Systemet har gjort at portørernes arbejdsdag er blevet effektiviseret ved at gøre det lettere for portørerne at koordinere og planlægge driftsopgaver. Opgavesystemet gav inspiration til en idé om at *Columna Service Logistics* også skulle kunne håndtere opgaver rettet mod sygeplejerskerne, således at patientkald blev til en opgave der kunne løses gennem opgavesystemet, i stedet for en alarm uden årsag som det er med kaldesnoren i dag. Derfor tænkte projektgruppen at forbedre patientkaldet ved at udvikle en løsning hvor patienten har mulighed for at knytte en årsag til kaldet, som sendes til *Columna Service Logistics*, hvor plejepersonalet kan håndtere det, og på den måde på forhånd vide hvad kaldet indebærer med henblik på at spare skridt. Løsningen til patienterne skulle bestå i en applikation, som patienterne kan downloade ned på deres egen smartphone. For at tilknytte årsagen til patientkaldet skal patienten vælge årsagen ud fra foruddefinerede årsager.

Ideen med at sende patientkald ind i *Columna Service Logistics* blev drøftet på endnu et møde med Systematic. På mødet blev det gjort klart at integrationen til Systematic ville kræve en udvidelse af opgavesystemet, da det ikke er udviklet til at kunne håndtere plejeopgaver på afdelingsniveau, men derimod større logistikopgaver på tværs af sygehuset. Da plejeopgaverne omkring patienterne ikke skal tage lang tid at planlægge og udføre blev det på mødet besluttet at projektgruppen skulle udvikle et bud på en løsning der kan varetage denne form for opgaver. På længere sigt ville Systematic kunne bruge projektgruppens arbejde som inspiration til at udvide *Columna Service Logistics* til fremtidig håndtering af de plejeopgaver der følger med et patientkald. Det gav anledning til at projektgruppen kunne udforme en projektbeskrivelse.

Projektbeskrivelse

I dette projekt arbejdes der med konceptet *patientkald med tilknyttet årsag*. Et patientkald med en tilknyttet årsag betyder at en patient har mulighed for at tilføje en årsag til sit patientkald når patienten har behov for en serviceydelse, som ikke er akut. Det giver personalet mulighed for at medbringe det patienten forespørger og personalet kan dermed spare tid og skridt.

For at løse dette er der udviklet et patientkaldssystem med fire forskellige moduler, hvor tre af dem er applikationer kaldet PatientApp, PersonaleApp og AdminApp og det sidste er et WebAPI.

- PatientApp er den smartphoneapplikation som indlagte patienter kan downloade ned på deres egen smartphone. Patienterne kan med denne app sende et patientkald og tilknytte en årsag til det ud fra nogle foruddefinerede valgmuligheder
- PersonaleApp er en smartphoneapplikation hvor personalet kan modtage patientkald og se den tilknyttede årsag. Personalet kan efterfølgende vælge at udføre patientkaldet eller lade det vente, hvis det vurderes at noget andet skal færdiggøres først. Personalet har mulighed for at se afventende kald og en historik over de kald der er udført

- AdminApp er en webapplikation som gør det muligt for en hospitalsafdeling at tilpasse patienternes forudbestemte valgmuligheder fx muligheden for hjælp til amning på familieafsnittet, hvilket ikke vil give mening på akutafdelingen
- WebAPI fungerer som kommunikationscentral mellem ovennævnte applikationer, da der skal udveksles data mellem disse

Derudover benyttes en fælles database som tilgås via WebAPI og persisterer al det data der bruges imellem modulerne. Blandt andet patientens foruddefinerede valgmuligheder fra AdminApp og det data der hører med til et patientkald når det oprettes af en patient eller udføres af et personale. Systemet kaldes samlet set PatientCare og er dermed en fællesbetegnelse for alle modulerne tilsammen.

Vision

Visionen med PatientCare er at gøre patientkald mere informative ved at knytte en årsag til kaldet. Dette kan skabe overblik for plejepersonalet og giver mulighed at patienterne kan få den rette service hurtigere. Plejepersonalet har mange opgaver der skal udføres i løbet af en arbejdsdag, såvel administrative som praktiske. Med et værktøj der giver mulighed for koordinering og planlægning af opgaver kan plejepersonalet på forhånd forberede sig på årsagen til at patienten har kaldt.

Tabel 1 viser de værdier som PatientCare forventes at have i praksis for patienter og personale på sygehusafdelinger.

Værdier	
Patient	Plejepersonale
<ul style="list-style-type: none">• Bedre patientinddragelse• Bedre service• Bedre patientoplevelse• Ingen tvivl om at begrundelsen for at kalde er god nok i ikke-akutte tilfælde• Synlig status på patientkald	<ul style="list-style-type: none">• Mere informativ end kaldesnoren• Bedre overblik over plejeopgaver• Giver mulighed for at koordinere og planlægge plejeopgaver• Sparrer skridt i form af gåturen frem og tilbage for at få informationen om patientkaldet• Effektiviserer arbejdsgangen• Fordeling af plejeopgaver• Færre forstyrrelser fra andres patienter• Bedre arbejdsmiljø• Dokumentation af plejeopgaver

Tabel 1 Værdier for patient og plejepersonale

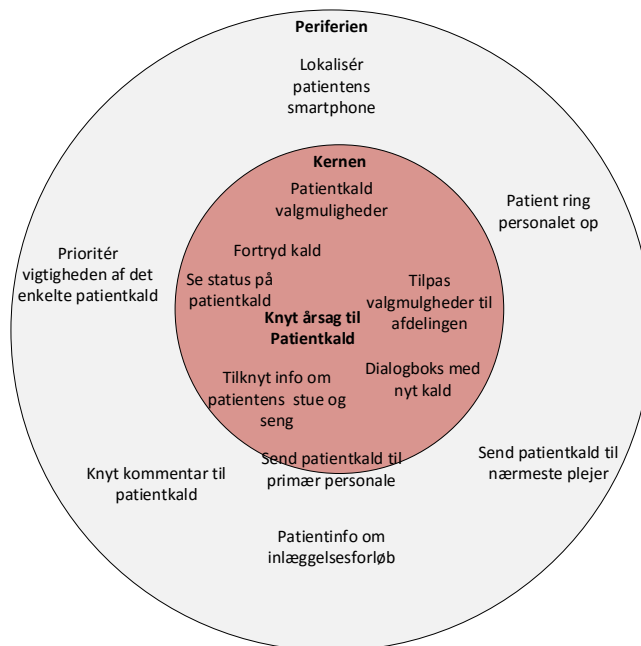
3 Afgrænsning

I dette afsnit beskrives hvilke beslutninger projektgruppen har valgt at tage, for at afgrænse projektet. Først afgrænses funktionerne som prototypen for systemet skal have. Da afdelinger er meget forskellige og derfor har forskellige måder at udføre plejeopgaver omkring patienterne på, afgrænses projektet til at tage udgangspunkt i en afdeling med en bestemt organisationsstruktur. Til sidst afgrænses de integrationer som projektgruppen finder relevante for PatientCare at integrere til.

Funktionalitet

I dette afsnit afgrænses ønskerne om funktionaliteten til PatientCare, hvis vigtigste opgave er at knytte en årsag til patientkald. Ønskerne bygger på udtalelser fra sygeplejersker og fagfolk som projektgruppen har været i dialog med (5). Prototypen af systemet bærer præg af at **koncept og funktion er prioriteret højere end performance og sikkerhed**, da det er afgørende at konceptet bliver vist før performance og sikkerhed har en betydning.

Kerneopgaverne der ligger inden for den røde cirkel i figur 1 er prioriteret vigtigst i første omgang og underbygger kravene for en prototype. De opgaver der ligger i periferien kan genovervejes til en fremtidig udvidelse af systemet.



Figur 1 viser kerneopgaver i midten og resterende opgaver i periferien

Send patientkald til primær personale ligger på grænsen mellem kernen og periferien fordi det er en funktion som er det næst vigtigste for slutbrugerne, **for at opgaverne fordeles og det øvrige personale ikke forstyrres.**

Projektgruppen er undervejs stødt ind i nogle problemstillinger som ikke er væsentlige at løse før systemet kan fungere i praksis. Et eksempel på dette er **lokalisering af patientens smartphone**. Patienten får pludselig mulighed for at oprette et patientkald alle steder fra, hvilket ikke har været muligt førhen, da kaldesnoren er stationær. Der vil derfor opstå et problem i at patienten kan sende et patientkald uafhængig af hvor pa-

tienten befinder sig. Det er især et problem hvis patienten ikke er indlagt på et sengeafsnit. Derfor antages det i afgrænsningen at patienten altid befinder sig i nærheden af den seng på den stue patienten er indlagt.

Afdeling

Der er ligeledes taget et valg om at afgrænse målgruppen for projektet eftersom hver afdeling på hvert hospital er indrettet forskelligt og har forskellige formål og forskellige måder at udføre de daglige opgaver omkring patienterne på. Afdelinger som PatientCare kunne være relevant for er undersøgt i projektets Mini-MTV og der er efterfølgende taget udgangspunkt i en udvalgt afdeling som har været villige til at indgå i et samarbejde omkring projektet. Denne afdeling er Gynækologisk - Obstetrisk afdeling på Regionhospital i Randers. Fremover betegnes afdelingen som gyn-obs. På afdelingen er der:

- 11 senge fordelt på 5 stuer
- Plejepersonalet består af både sygeplejersker og jordmødre
- Tildelt patientpleje
- Indlagte patienter som er i stand til at foretage nogle valg når de har behov for serviceydelser
- Patientkald hvor størstedelen drejer sig om serviceydelser og ikke akutte nødhjælp

Tildelt patientpleje

På gyn-obs har de tildelt patientpleje for at undgå at patienterne møder alt for meget forskelligt personale under indlæggelsen. Det betyder at der er én person der kender patientens historie og indlæggelsesforløb godt og dermed nemt kan se hvis patientens tilstand forværres eller forbedres. På den måde sikres den rigtige behandling. På gyn-obs har de som udgangspunkt én der er primært personale på den enkelte patient. På andre afdelinger har man både primær og sekundær personale på en patient og andre afdelinger igen har et helt team af fagfolk omkring patienten, hvis patientens sygdom fx kræver inddragelse af mange faggrupper (6).

Patienter

Patients forudsætninger for at kunne bruge en app afhænger blandt andet af deres sygdom og alder. På gyn-obs er der patienter i alle aldersgrupper fra unge til ældre kvinder. Det er en afdeling hvor der er mange indlagte kvinder efter fødsel, hvorfor mange af patienterne er fra en generation som har forudsætningerne for at bruge en smartphone. Ligeledes er patienterne ofte ikke så berørte af deres sygdom, at de ikke er i stand til at tage stilling til hvad de har behov for ud fra nogle foruddefinerede valgmuligheder.

Serviceydelser

Årsager til patientudløste kald som systemet fx skal kunne håndtere er typisk serviceydelser, hvor personalet kan forberede sig inden mødet med patienten. Eksempler på disse, er listet herunder og stammer fra en behovsundersøgelse hvor projektgruppen har været i dialog med sygeplejersker fra afdelingen (7):

- *Forplejning*: Behov for mad/drikke som personalet kan tage med til patienten på forhånd
- *Hygiejne*: Behov for et bad, blive vasket eller komme på toilettet, hvor personalet kan tage de nødvendige remedier med på forhånd
- *Tålelige smerter*: Behov for smertestillende medicin til at dulme smerter yderligere efter fx en operation eller hovedpine, hvor personalet kender deres patient i forvejen og ved hvad for noget smertestillende medicin patienten skal have, som de kan tage med på forhånd sammen med et glas vand

- *Mobilisering*: Behov for at blive vendt i sengen for at undgå liggesår og tryksår, hvor personalet kan tage remedier med til at vende patienten eller få assistance af en kollega hvis det er nødvendigt, inden mødet med patienten.

Det er vigtigt at være opmærksom på at systemet ikke skal håndtere de kald som patienten foretager i en akut situation. PatientCare er derfor ikke en erstatning af kaldesnoren men et supplement.

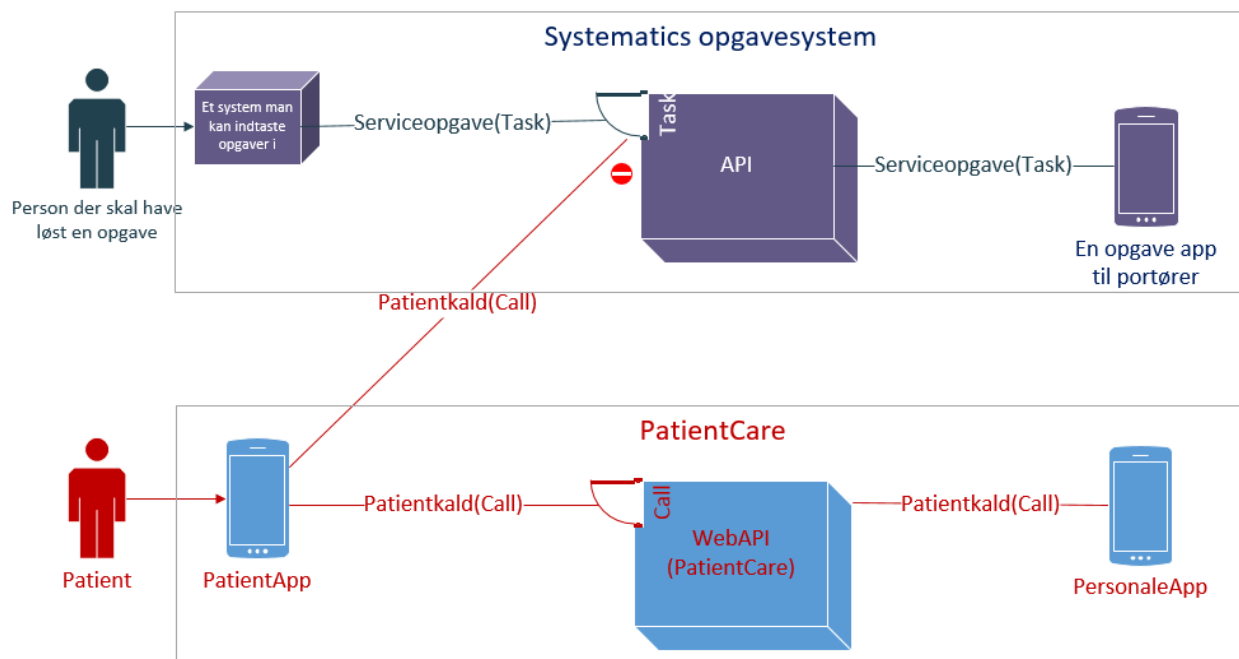
Integrationer

Projektgruppen har gjort sig nogle overvejelser om hvilke systemer der kunne være relevante for PatientCare at integrere til. Disse beskrives her.

Systematics Columna Service Logistics

Columna Service Logistics håndterer, som tidligere nævnt, serviceopgaver der udføres af portører og omhandler transport af patienter på tværs af afdelinger. Opgavesystemet har gjort at portørernes arbejdsdag bliver effektiviseret ved at gøre det lettere for portørerne at koordinere og planlægge driftsopgaver (8). Det ville være optimalt hvis PatientCare kunne sende patientkald til dette opgavesystem og sygeplejerskerne kunne benytte sig af den applikation som allerede er udviklet til portørerne.

Columna Service Logistics er dog ikke udviklet til at håndtere patientkald og de opgaver som skal løses som følge af et patientkald kræver rent teknisk nogle andre parametre end større serviceopgaver gør. Dette vides fordi projektgruppen har fået adgang til Systematics dokumentation for *Columna Service Logistics*. Der er fx mange steps en serviceopgave skal igennem før den er udført, blandt andet kan portørerne booke sig ind på opgaver og udføre dem løbende. Det ville ikke give mening for et patientkald at skulle igennem disse steps fordi de opgaver der følger med et patientkald er nogle der løses indenfor kort tid og det skal ikke tage længere tid at klikke sig igennem opgaven end at udføre selve opgaven. I stedet for at forsøge at tilpasse patientkald til et system der håndterer omfattende serviceopgaver og dermed forsøge at putte opgaver på afdelingsniveau ind i et system der håndterer opgaver på tværs af afdelinger valgte bachelorgruppen i overensstemmelse med Systematic at lave en ny løsning. Hvis systemerne skal kunne snakke sammen skal der derfor foretages en udvidelse af *Columna Service Logistics*. Projektgruppen har derfor ikke haft mulighed for at integrere til *Columna Service Logistics*, men udvekslingen af data mellem PatientApp og PersonaleApp gennem WebAPI'et i PatientCare er løst med samme teknologi som det kræver at kommunikere til grænsefladen for *Columna Service Logistics*. Dette foregår med http-protokollen. Der er derfor taget højde for en fremtidig integration.



Figur 2 Integration til Systematic

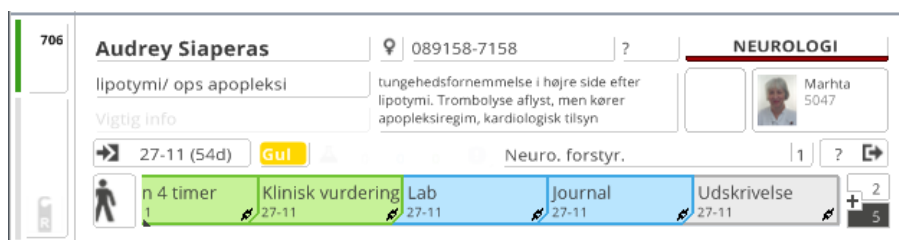
Figur 2 illustrerer at Systematics opgavesystem ikke kan modtage patientkald, men kun serviceopgaver som portørerne udfører.

Cetreas Klinisk Logistik

PatientCare er udviklet med henblik på at trække data fra et system som Cetreas kliniske logistik hvor brugbare oplysninger i forvejen registreres. Dette er gjort med henblik på at udnytte data fra et system hvor de i forvejen er registreret og dermed undgå dobbeltregistrering. For at PatientCare kan blive en realitet i praksis er det derfor nødvendigt at være understøttet af et system som dette, der leverer oplysninger om blandt andet patientens navn, stue, sengeplads og indlæggelsestidspunkt. På gyn-obs bruger de Cetreas klinisk Logistik hver dag og det er derfor oplagt at kunne trække data herfra.

Det har ikke været muligt at samarbejde med Cetrea fordi de har haft travlt med andre projekter. Derfor har projektgruppen opsat et data mock for hver patient som skal illustrere et sådant system.

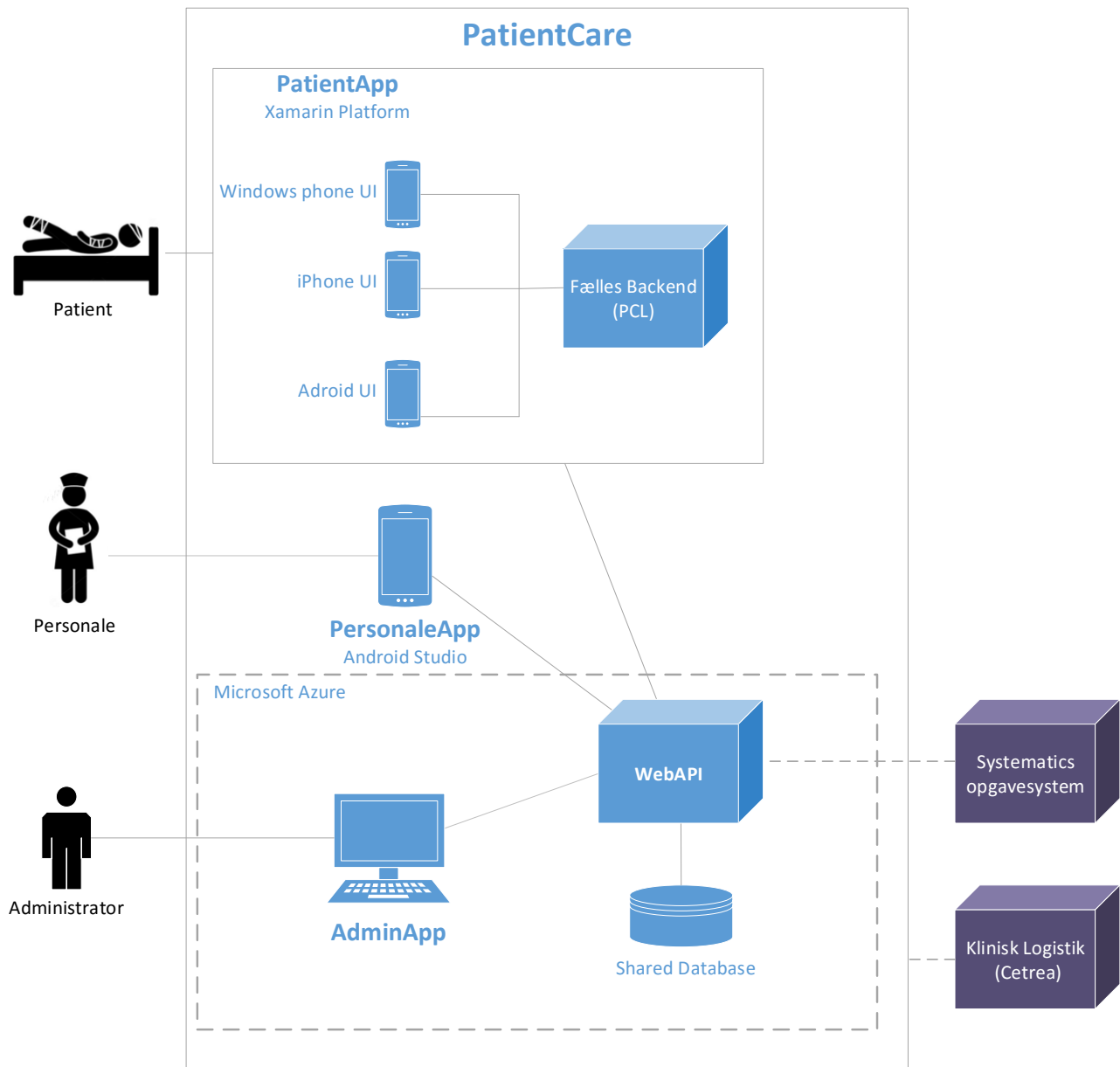
På figur 3 ses et eksempel på en patientkomponent på Cetreas kliniske logistik skærme, hvor patienten Audrey har fået tildelt primær personalet Marhta. Man kan samtidig se at Audrey ligger på stue 706 og er blevet indlagt d. 27/11.



Figur 3 Patientkomponent Cetreas klinisk logistik

4 Systembeskrivelse

På figur 4 ses en oversigt over systemets moduler og de aktører der interagerer med systemet. De moduler som projektgruppen selv har stået for udviklingen af, er angivet med farven blå. De lilla moduler illustrerer de sundheds-IT systemer som der i projektet er gjort overvejelser om, at PatientCare med fordel kan integrere til i fremtiden.



Figur 4 Systemoversigt

PatientApp

Prototypen af PatientApp er udviklet som en cross-platform løsning med Xamarin med henblik på at kunne understøtte både iOS, Android og Xamarin, for at favne bredt i forhold til patienternes egne devices.

PersonaleApp

Prototypen af PersonaleApp er udviklet til Android som ifølge statistikker fra markedsandelen for smart-phone operativsystemer er det mest udbredte smartphonestyresystem (9). Til forskel fra PatientApp, der skal kunne understøtte flere forskellige platforme, behøver PersonaleApp kun at understøtte en platform.

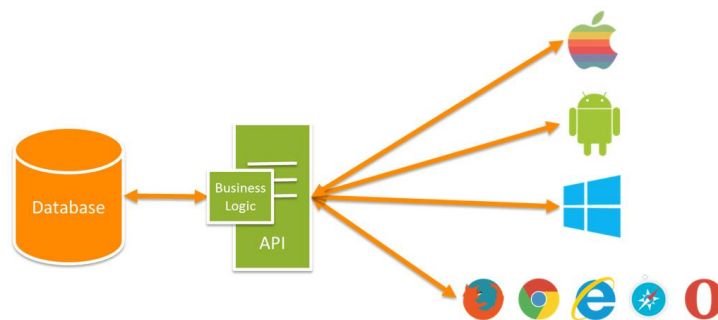
AdminApp

AdminApp er lavet på asp.net MVC 5 og denne applikation er hostet på Azure.

WebAPI

Hele PatientCare er bygget op omkring et WebAPI, som fungerer som kommandocentral og sørger for kommunikationen og dermed transport af data mellem de forskellige moduler i PatientCare (PatientApp, PersonaleApp og AdminApp). Kommunikationen foregår ved at data hentes fra eller leveres til en webside som WebAPI udstiller.

Figur 5 viser princippet i et WebAPI, som projektgruppen har udnyttet.

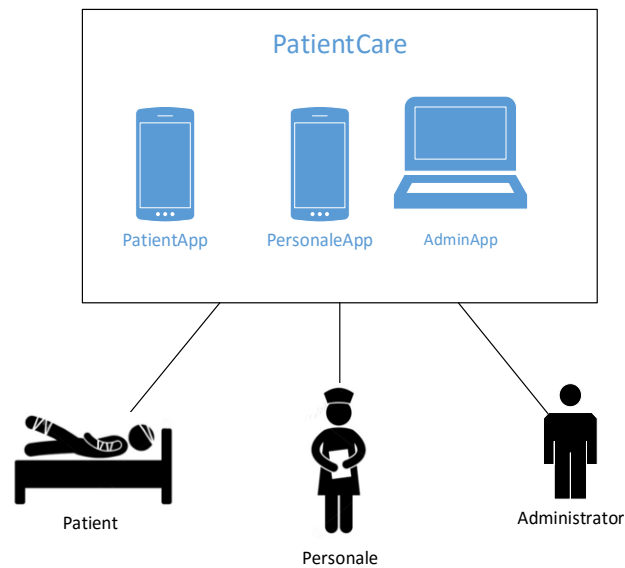


Figur 5 WebAPI'et har forbindelse til resten af modulerne (10)

Fordelen ved at bygge systemet op omkring et WebAPI, er at modulerne gøres uafhængige af hinanden, så der opnås lav kobling, hvilket kan lede til bedre performance, vedligeholdelse og udvidelse.

Aktørbeskrivelse

Her beskrives de aktører som interagerer med systemet, disse ses på figur 6.



Figur 6 aktør-kontekst-diagram

Patient

Patienten repræsenterer indlagte patienter, typisk unge til midaldrende, som har behov for hjælp af det sundhedsfaglige personale. De kan sende patientkald afsted med deres egen smartphone via smartphone applikationen PatientApp, som er udviklet på en Xamarin platform med henblik på at kunne understøtte både iOS, Android og Windows.

Personale

Personalet repræsenterer det plejepersonale der på en hospitalsafdeling har ansvar for pleje af patienterne og fungerer som kontaktperson når de har behov for hjælp. De modtager patientkald gennem PatientCare systemet på deres arbejdstelefon (smartphone), via smartphone applikationen PersonaleApp, som understøtter Android.

Administrator

Administrator repræsenterer en sundhedsfaglig person, typisk en sygeplejerske, som står for de administrative opgaver på en hospitalsafdeling. Administratoren sørger for at tilpasse systemets opsætning så den passer til en pågældende afdeling. Tilpasningen indebærer blandt andet oprettelse af foruddefinerede valgmuligheder, som patienterne på afdelingen kan vælge imellem, når de har behov for det. Administratoren sætter altså rammerne for hvilke patientkald patienterne kan sende og dermed hvad personalet modtager. Tilpasningen sker gennem webapplikationen AdminApp som tilgås via en webbrowser på en computer.

5 Materialer og metoder

Formålet med dette afsnit er at beskrive faserne i projektarbejdet, samt hvilke materialer og metoder der er anvendt for at drive bachelorprojektet i mål.

Udviklingsproces

Udviklingen af PatientCare har været en brugerdreven proces, hvor de sundhedsprofessionelle har været involveret igennem store dele af forløbet. I starten har de sundhedsprofessionelle været med til at identificere problemstillinger ved deres arbejde, som teknologi evt. kunne afhjælpe.

Udviklingsdokumentation

Under projektforsløbet blev der udarbejdet følgende dokumenter som afspejler udviklingsfasen af prototypen:

Kravspecifikation

I starten af projektet blev der udarbejdet en kravspecifikation som indeholder alle tænkelige krav, som stilles til det fuldstændte system. Kravene til prototypen er inddelt i funktionelle og ikke funktionelle krav og for funktionelle krav er der opstillet fully dressed use cases. Disse use cases lægger direkte op til hvordan systemet skal designes og hvilke opgaver der skal løses rent teknisk, under udvikling af prototypen i designfasen.

Accepttest

Der er udarbejdet en accepttest med det formål at have dokumentation for status for prototypen. Accepttesten indeholder en punktlig gennemgang af de tekniske krav fra kravspecifikationen, det forventede resultat og det faktiske resultat testet på prototypen.

Designdokument

Der er udarbejdet et designdokument med det formål at give en detaljeret beskrivelse af hvordan de tekniske krav til systemet er løst. Designdokumentet henvender sig til udviklere af systemet og er delt op i to dele: *Systemarkitektur* og *Design*. I systemarkitekturen beskrives forbindelserne mellem systemets moduler og opbygningen af hvert modul. Efter systemarkitekturen for PatientCare var blevet bestemt delte projektgruppen sig op for at arbejde parallelt med design og implementering af hvert modul. I design beskrives implementeringen af de funktionelle krav der blev specificeret i kravspecifikationen og hvordan de enkelte moduler udføre operationer.

Tidsplan

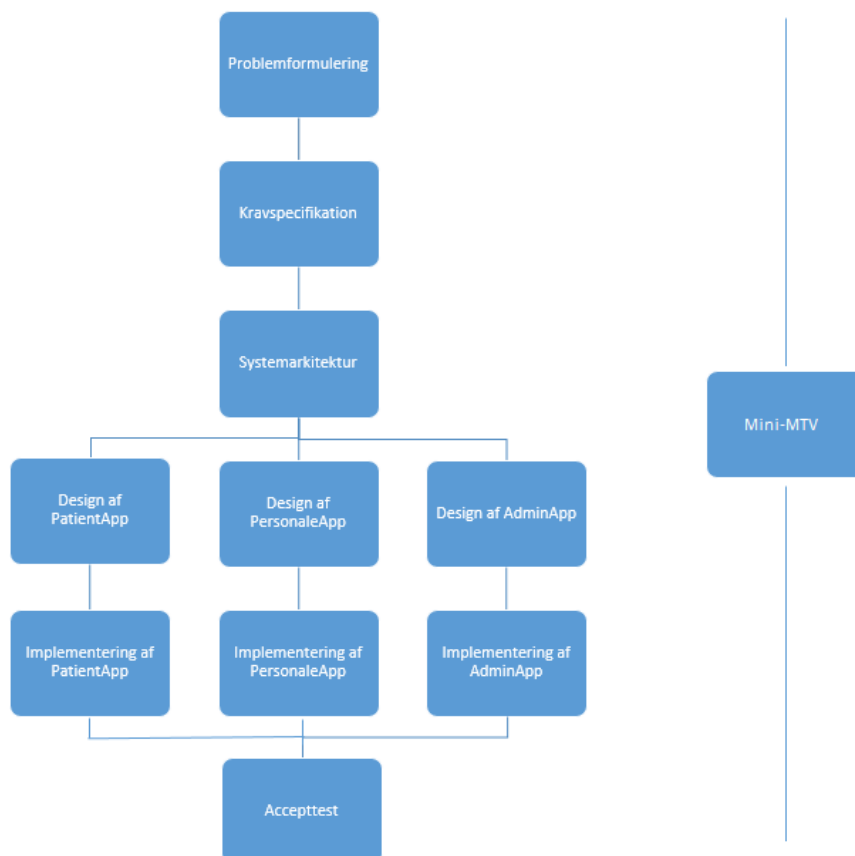
Tidsplanen blev brugt som en oversigt over hvilke faser af projektet der ville være størst i bestemte perioder af den tid der var stillet til rådighed. Tidsplanen er opsat som et Gantt-skema som vist i tabel 2, der illustrerer ugerne som et resumé af projektforsløbet. Skemaet viser samtidig strukturen af projektet, hvor faserne er overlappende.

Uge	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
Behovsanalyse																	
Mini-MTV																	
Kravspecifikation																	
Designdokument																	
Udvikling																	
Accepttest																	
Afprøvning																	
Rapport																	

Tabel 2 Gantt-skema for tidsplan

Agil systemudvikling

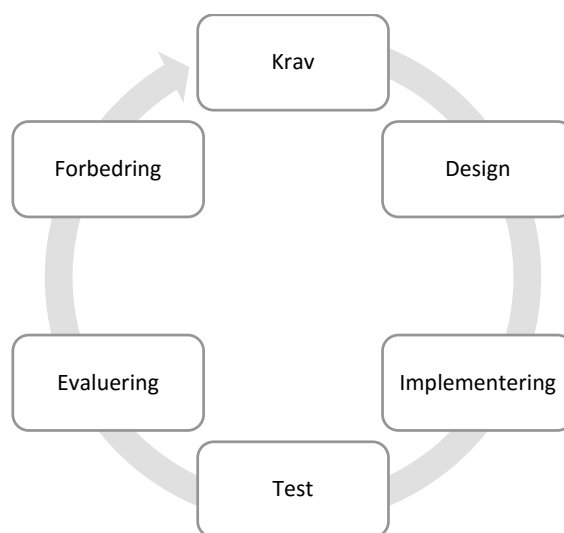
Figur 7 viser de udviklingsfaser som projektgruppen i store træk har været igennem i udviklingsprocessen. Hver fase afsluttes ikke før den næste påbegyndes, der arbejdes altså agilt med faserne. Mini-MTV er udarbejdet sideløbende med systemudviklingen hvor brugerinddragelsen har været i fokus.



Figur 7 Udviklingsproces

Iterativ proces

Projektgruppen har fulgt en iterativ udviklingsproces som vist på figur 8, hvilket giver et fleksibelt og tilpassningsdygtigt system. Da PatientCare er komplekst og lægger op til mange udfordringer, er det nødvendigt at dele systemet op i mindre dele, så det gøres simpelt og overskueligt og kan løses i iterationer. Ved løbende at designe, implementere og teste kan eventuelle fejl opdages tidligt og det er nemt at foretage ændringer hvis krav revurderes.



Figur 8 Iterativ proces

De opgaver der følger med kravene i use casene er løst i iterationer. I hver iteration designs, implementeres og testes en opgave der skal løses. Herefter evalueres iterationen og er opgaven løst kan en ny iteration med den næste opgave påbegyndes, men er opgaven ikke løst gentages iterationen til opgaven er løst. Denne proces gentages flere gange for hvert krav.

Scrum elementer

Der er gjort brug af elementer fra *Scrum* blandt andet *Scrum boardet*, både online og på papirform. Opgaverne blev opdelt i *To do, In progress, To test og Done*. Dette gav et overblik over hvilke opgaver der skulle løses og hvilke opgaver der var igangværende, klar til test og færdige. I løbet af udviklingsfasen blev der dagligt afholdt *scrummøder* for at drøfte dagens opgaver og eventuelle forhindringer.

Mini-MTV

Projektets Mini-MTV er udarbejdet sideløbende med udviklingen af systemet og skal afspejle et beslutningsværktøj der bygger på MTV-tankegangen. Metoden er i dette projekt anvendt til at undersøge forudsætningerne for og konsekvenserne af at indføre et system som PatientCare på en udvalgt afdeling. Undersøgelsen bygger på et teknologisk, organisatorisk, patientmæssigt og økonomisk perspektiv.

I det følgende beskrives de materialer og metoder der er anvendt i Mini-MTV'en til at besvare spørgsmål som: *hvad kan PatientCare bidrage med som det nuværende kaldeanlæg ikke kan?* Læs mere i dokumentet Mini-MTV.

Interessentanalyse

For at undersøge hvilke interessenter der berøres af den nye teknologi er der lavet en interessentanalyse.

Interviews og møder med slutbrugere

Der er lavet interviews og afholdt møder med sygeplejersker i forbindelse med undersøgelse af behovet for PatientCare.

Møder med eksterne fagfolk

Derudover er der afholdt møder med eksterne personer såsom IT-konsulenter på sygehuse og IT-arkitekter fra Region Midt for at udveksle erfaring på området.

Litteratursøgning

Der findes ingen **evidens for at kaldesystemer**, hvor indlagte patienter kan knytte en årsag til kaldet har positiv effekt på afdelingsniveau på hospitaler. Det skyldes at der simpelthen ikke er fundet eksempler på teknologier hvor dette er muligt. Derfor har det været svært at søge evidensbaseret litteratur. Litteratursøgningen har i stedet foregået i forbindelse med at sammenligne PatientCare systemet med andre eksisterende kaldeanlæg.

Observationsstudie

Der er foretaget observationer, hvor personalets arbejdsgang på en udvalgt afdeling på Regionshospitalet Randers er blevet observeret. Formålet med observationen er at observere brugen af en nuværende teknologi på en udvalgt afdeling, spørge ind til hvilke problemstillinger de oplever og de rutiner de ansatte har med det kaldeanlæg de bruger i dag.

Spørgeskemaundersøgelse

Der er udarbejdet en spørgeskemaundersøgelse for at danne et overblik over den generelle holdning til idéen med PatientCare og oplevelsen af at være indlagt. Spørgeskemaundersøgelsen er en kvantitativ metode og har derfor kunnet give målbare resultater. Spørgeskemaet er sendt ud på facebook målrettet den almindelige borger der har været indlagt på et hospital i Danmark og eller har en holdning til sagen.

Tænk højt studie

Prototypen af PatientCare er blevet afprøvet af personalet fra den udvalgte afdeling i forbindelse med et tænk højt studie hvor konceptet for PatientCare blev gennemgået på et møde. Formålet var at få feedback fra plejepersonalet på brugeroplevelsen og funktionaliteten af PatientCare.

Tidsstudie

Da det ikke har været muligt at afprøve prototypen af PatientCare af i klinisk praksis foretog projektgruppen selv et tidsstudie. I forbindelse med tidsstudiet blev der målt tid og antal skridt ved at gennemgå et scenarie, hvor en patient opretter et kald gennem PatientCare, hvor personalet kender årsagen til kaldet inden mødet med patienten. Dette blev sammenlignet med tid og antal skridt for et kald hvor årsagen var ukendt. Tidsstudiet er foretaget for at give en indikation på om løsningen kan spare personalet for tid og nedsætte antallet af skridt de går i forbindelse med et patientkald.

Projektstyring

Versionsstyring

Projektgruppen har benyttet sig af versionsstyring af kode for at kunne udveksle arbejdet med hinanden eller gå tilbage til en tidligere version. Da projektgruppen består af fire medlemmer er dette en fordel for at alle medlemmer hele tiden har haft mulighed for at få den senest opdaterede version af projektstoffet at arbejde videre med. Det gælder både software og dokumentation.

Team foundation server

For modulerne PatientApp, AdminApp og WebAPI er der brugt Team Foundation Server (TFS) til versionsstyring af koden.

Udover versionsstyring er TFS også brugt som projektstyringsværktøj hvor opgaverne der skulle løses har ligget digitalt så de kunne tilgås alle steder fra.

GitHub

Til versionsstyring af koden for PersonaleApp er der anvendt GitHub. GitHub er et versionsstyringsværktøj ligger tilgængelig på nettet. Projektgruppen har haft en brugerkonto som har gjort at koden for PersonaleApp ikke har været tilgængelig for andre.

TortoiseSVN

TortoiseSVN (subversion) giver mulighed for at se hvilke ændringer og hvilke dokumenter der sidst er blevet tilføjet til det fælles SVN-drev. Dette har projektgruppen gjort brug af i forbindelse med dokumentation af arbejdet.

Udviklingsværktøjer

Projektgruppen har benyttet sig af følgende programmer:

- Microsoft Visual Studio er anvendt til udvikling af PatientApp, AdminApp og Web API
- Microsoft Word er anvendt til rapport- og dokumentering
- Microsoft Visio er anvendt til at lave diagrammer og figurer
- Android Studio er anvendt til at udvikle PersonaleApp
- XCode er anvendt for at kunne bygge og debugge iOS projekter på Xamarin platformen

Ansvarsområde

Projektgruppen som har lavet dette bachelorprojekt består af fire studerende fra to forskellige diplomingeniøruddannelser: Informations- og kommunikationsteknologi (IKT) og sundhedsteknologi (ST).

Samarbejdet har båret præg af et teamsamarbejde, hvor alle har mødtes hver dag for at diskutere, tage beslutninger, løse problemer, planlægge og fokusere på et fælles mål med bachelorprojektet. Der har ikke været en udnævnt teamleder, men det har naturligt ligget til nogle af teammedlemmerne at tage styringen når der har været behov for det.

For at nå målet har der været en ansvarsfordeling i overensstemmelse med den enkeltes kompetencer og uddannelse.

Anders (IKT-studerende)

Har haft ansvar for design og implementering af AdminApp og WebAPI foruden at have haft hovedansvaret

for design og implementeringen af fællesdatabasen. Anders har også haft hovedansvaret for oprettelse og brugen af TFS.

David (IKT-studerende)

Har haft ansvar for design og implementering af PatientApp, og de overvejelser der er blevet gjort i forhold til brugen Xamarin platform. David har også været medvirkende til design og implementering af PersonaleApp i Android. Bl.a. design og implementering af en service, som har været essentiel for udtræk af data fra WebAPI til PatientApp.

Minna (ST-studerende)

Har haft ansvar for kontakten med slutbrugerne for at undersøge behovet for et system hvor der tilknyttes en årsag til patientkaldet. Minna haft ansvar for udarbejdelsen af Mini-MTV ud fra den teknologiske og patientmæssige perspektiv og har også stået for at udvikle PersonaleApp i Android Studio i samarbejde med Camilla.

Camilla (ST-studerende)

Har haft ansvar for at undersøge slutbrugernes behov for et system hvor man kan knytte en årsag til patientkaldet i samarbejde med Minna. Camilla har haft ansvar for store dele af udarbejdelse af Mini-MTV ud fra et organisatorisk perspektiv og ressourcemæssigt perspektiv. I forhold til det tekniske har Camilla stået for store dele af implementeringen af PersonaleApp i Android Studio, systemarkitektur og design af app'en der er henvendt til personalet på en hospitalsafdeling.

Faciliteter

Projektgruppen har gjort brug af et bachelorlokale som har gjort det muligt at arbejde sammen som et team. Det har været den primære arbejdsplads for alle fire studerende i gruppen, foruden de dage hvor der har været møder ud af huset med eksterne kontaktpersoner.

Bachelorgruppen har haft en fælles e-mailadresse som har været nyttig når gruppen som en enhed har skullet kommunikere på skrift med vejlederen fra Ingeniørhøjskolen og eksterne personer. Ligeledes har den gjort at alle i gruppen har haft adgang til e-mailkorrespondancerne. Udover e-mailadressen er der gjort brug af en fælles kalender som har givet projektgruppen mulighed for at skrive fælles og individuelle aftaler ind. Det har givet et godt samarbejde at alle i projektgruppen havde et overblik over aftalerne.

Møder og eksternt samarbejde

Foruden scrummøder mellem projektgruppens medlemmer er der også ugentligt afholdt vejledermøde med vejleder fra Ingeniørhøjskolen Jesper Rosholm Tørresø for at drøfte status undervejs i projektet og blive vejledt i en fornuftig retning. Derudover har projektgruppen afholdt møder med eksterne kontaktpersoner og reviewmøder med en anden bachelorgruppe.

Samarbejde med Systematic

Systematic har været den primære årsag til at projektet blev sat i verden. Foruden det er de tekniske muligheder blevet diskuteret med Systematics IT-medarbejdere. Systematic har ikke fungeret som kunde på projektet men har fra start vist stor interesse og set potentiale i projektet. Flere medarbejdere har været behjælpelige med at svare på spørgsmål undervejs, udlevere udviklingsdokumentation omkring *Columna Service Logistics* og videregive kontaktoplysninger på sundhedsfaglige og IT-folk i regionen.

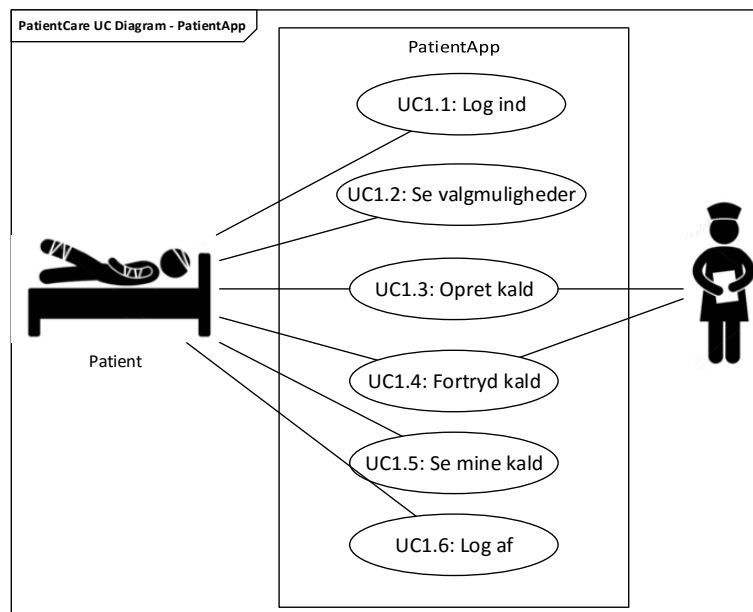
Samarbejde med sundhedsfaglige

I gennem Systematic har projektgruppen fået kontakt til en innovationskonsulent og projektleder fra Regionshospitalet i Randers, der skabte kontakt til gyn-obs for at indgå i projektet som testafdeling. Samarbejdet med afdelingen har været uundværligt i behovsundersøgelsen, hvor idéen med projektet blev drøftet med slutbrugerne, herunder sygeplejersker og jordemødre. Afdelingen kom med inputs til funktionalitet og italesatte behovet for et system, hvor man kan knytte en årsag til patientkaldet. Afdelingen er senere anvendt til at afprøve prototypen af systemet.

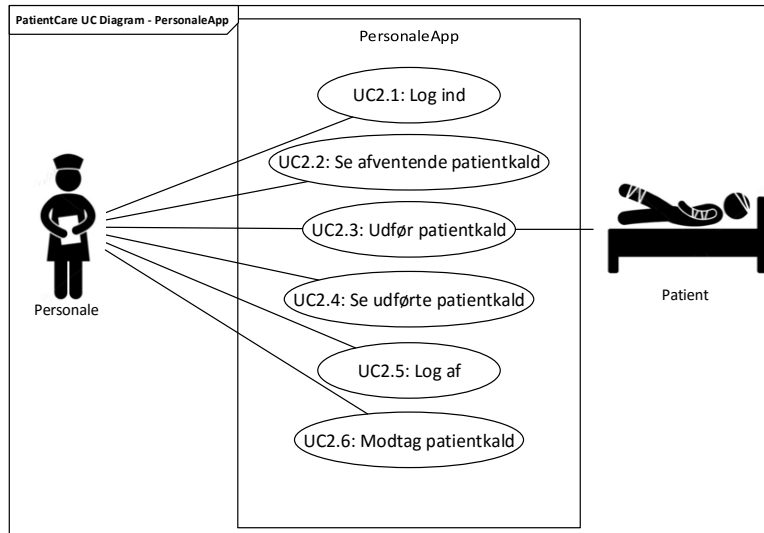
6 Krav

For at specificere kravene der skal være grundlag for udviklingen af prototypen for PatientCare, blev der udarbejdet en kravspecifikation. Disse krav afspejler de behov der er blevet identificeret i forbindelse med en behovsundersøgelse hvor slutbrugerne har været inddraget (5).

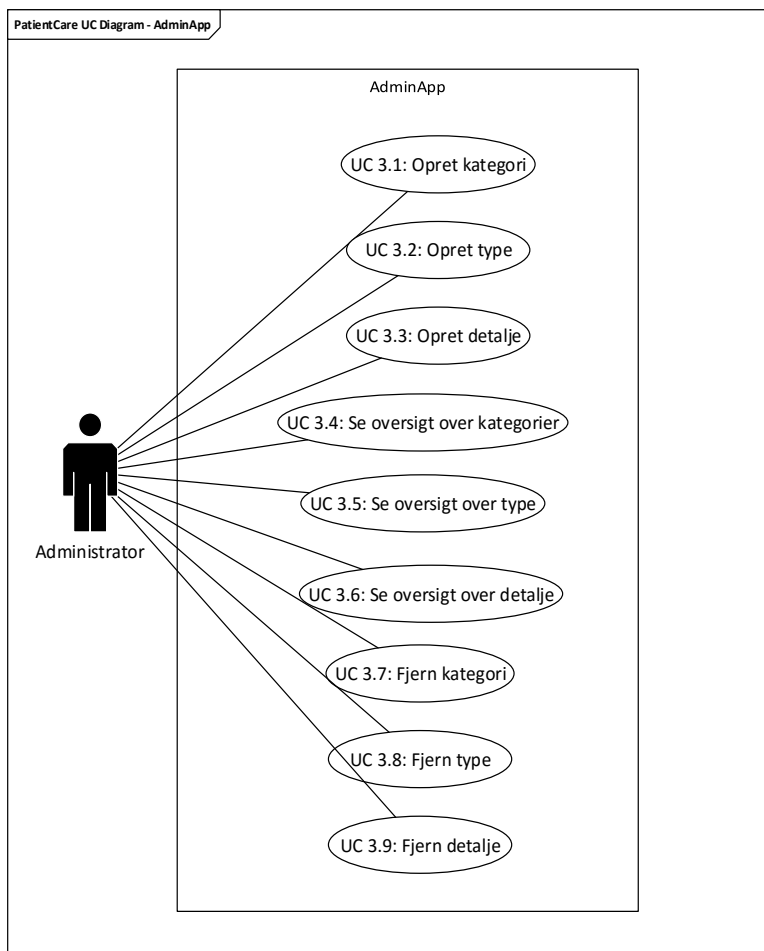
Alle kravene blev i kravspecifikationen opstillet og prioriteret i en MoSCoW-prioritering som *Must have*, *Should have*, *Could have* og *Wont have*. *Must have* er minimumskravene til systemet og dermed de krav som projektgruppen har udviklet i en prototype af systemet. Der er opstillet use case diagrammer for de tre moduler PatientApp (figur 9), PersonaleApp (figur 10) og AdminApp (figur 11). Ud fra disse er der lavet fully dressed use cases med henblik på at kunne omsætte kravene til en systemarkitektur og design over funktionaliteten over de enkelte moduler.



Figur 9 Use case diagram for PatientApp



Figur 10 Use case diagram for PersonaleApp



Figur 11 Use case diagram for AdminApp

Når en patient skal oprette et patientkald bliver patienten præsenteret for nogle valgmuligheder der er blevet foruddefineret gennem AdminApp. Disse valgmuligheder kan specificeres på tre niveauer: Kategori, type og detalje som vist på figur 12.



Figur 12 Valgmuligheders tre niveauer

Disse niveauer er beskrevet herunder.

Kategori:

Den højstrangerede valgmulighed: En kategori kan godt eksistere uden en type og en detalje. Eksempel på kategori: Drikke eller toilet (Se figur 13).



Figur 13 Eksempel på kategori

Type:

Den mellemrangerede valgmulighed. En type kan ikke eksistere uden en kategori, da typer er en underkategori for kategorien. En type kan have igen, en eller flere detaljer. Eksempel på type: Et glas vand (Se figur 14).



Figur 14 Eksempel på kategori med type

Detalje:

Den lavest rangerede valgmulighed. En detalje kan ikke eksistere uden en kategori og en type, da detaljer er en underkategori af typer som er en underkategori af kategorier. Eksempel på detalje: med mælk (Se figur 15) eller graden af smerter fx moderate (Se figur 16).

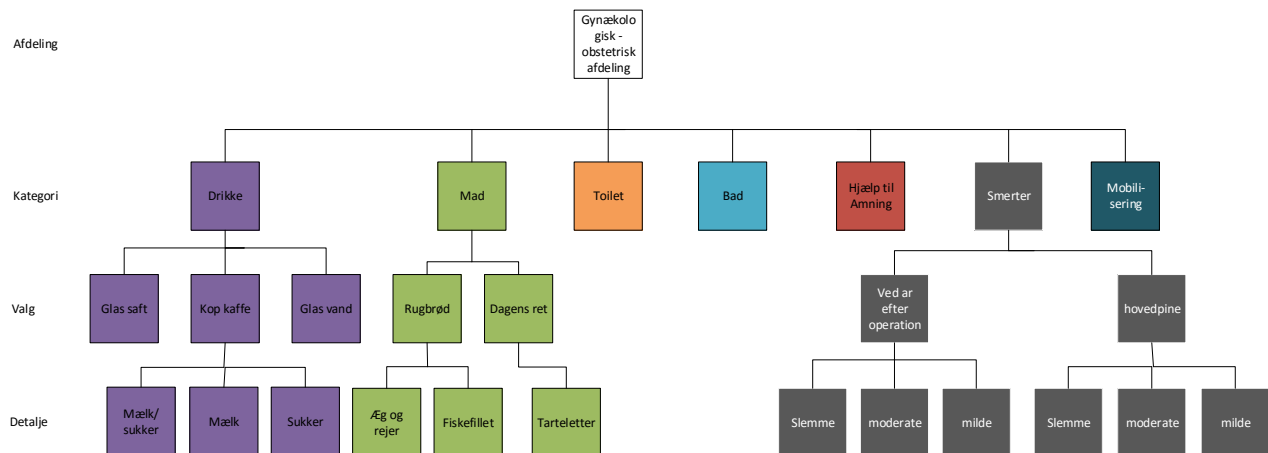


Figur 15 Eksempel på kategori med type og detalje



Figur 16 Et andet eksempel på kategori med type og detalje

Figur 17 viser hvilke valgmuligheder der er blevet talt med sygeplejerskerne fra gyn-obs om at patienterne for eksempel kunne have på deres afdeling.



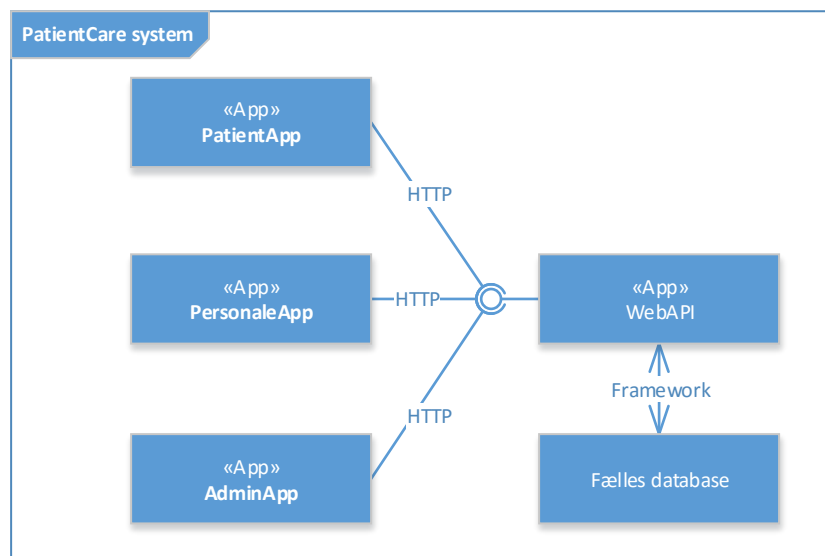
Figur 17 Hieraki af valgmuligheder på Gyn-obs i Randers

7 Systemarkitektur

Systemarkitekturen beskriver forbindelserne mellem systemets moduler og opbygningen af det samlede system, da der er behov for at specificere hvordan de enkelte moduler skal kommunikere sammen inden det designes.

Kommunikation mellem moduler

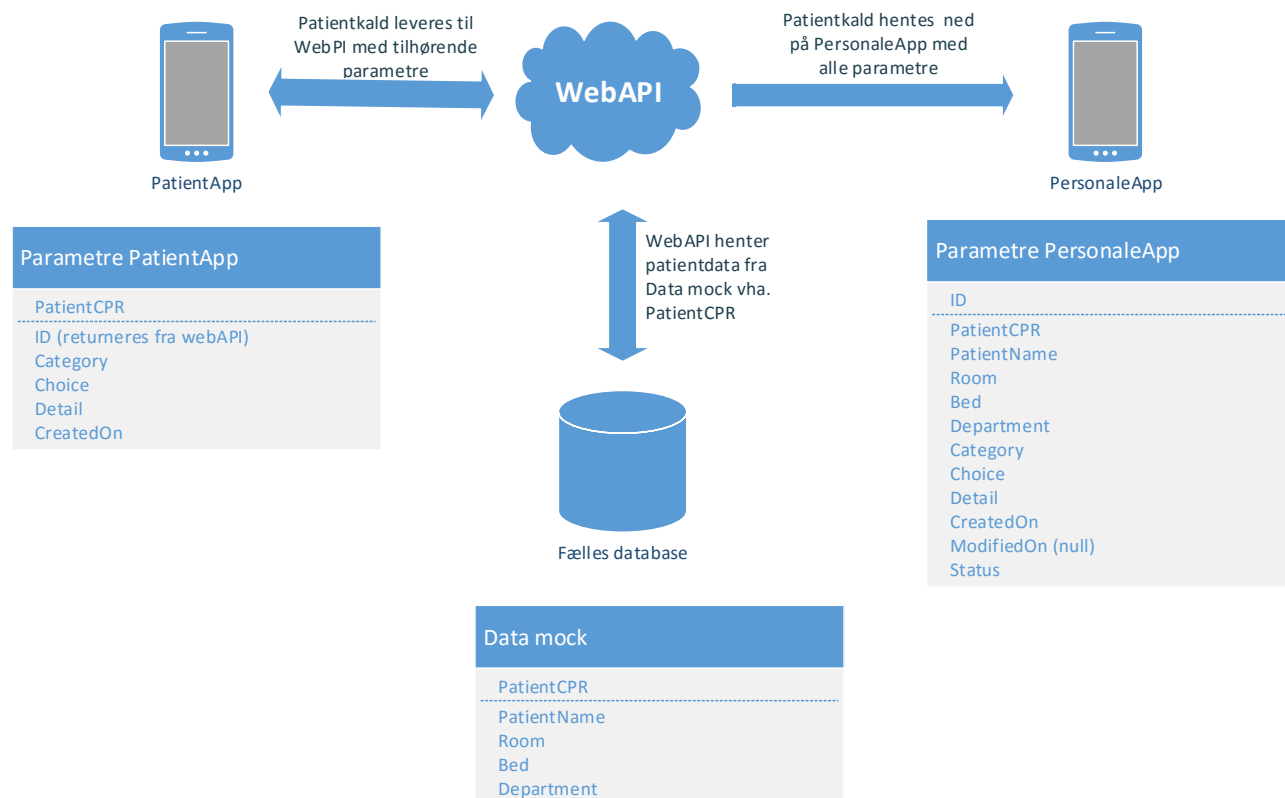
På figur 18 ses det at WebAPI er bindeled mellem modulerne. WebAPI benytter http-protokol til udveksling af data mellem modulerne og formatet af data er JSON. Dette er en anerkendt måde at kommunikere mellem moduler på og også den måde kommunikationen til Systemetics opgavesystem foregår på.



Figur 18 Kommunikationen mellem modulerne

Data

Figuren viser at WebAPI har forbindelse til en fællesdatabase. Der er behov for at kunne persistere data i en fællesdatabase for at alle moduler kan tilgå det. Det data der skal persisteres i fællesdatabasen består af data fra to forskellige steder: data som PatientCare systemet selv genererer i PatientApp (se parametre for PatientApp figur 19) og data mock af data af Cetreas kliniske logistik (Data mock figur 19).



Figur 19 viser en oversigt over hvor de forskellige parametre kommer fra.

Figuren viser også at PersonaleApp skal modtage det samlede data (se parametre for PersonaleApp figur 19).

Projektgruppens medlemmer blev enige at datamodellen skal se ud som ovenstående figurer viser. Dette er besluttet ud fra hvilke data en patient skal kunne generere ud fra de valgmuligheder patienterne kan vælge imellem når der er behov for hjælp. Ligeledes er datamodellen valgt ud fra hvilke oplysninger personalet har brug for omkring patienten og årsagen til kaldet når de modtager det. På et kald er der parameteren *status* som afgør hvilket stadie kaldet er i. Hvis status er 0 er kaldet *afventende*, hvis status er 1 er kaldet *udført* og hvis status er 2 er kaldet *fortrudt*. Denne datamodel ligger til baggrund for de modeller WebAPI modtager og returnerer.

Lokal data persistering

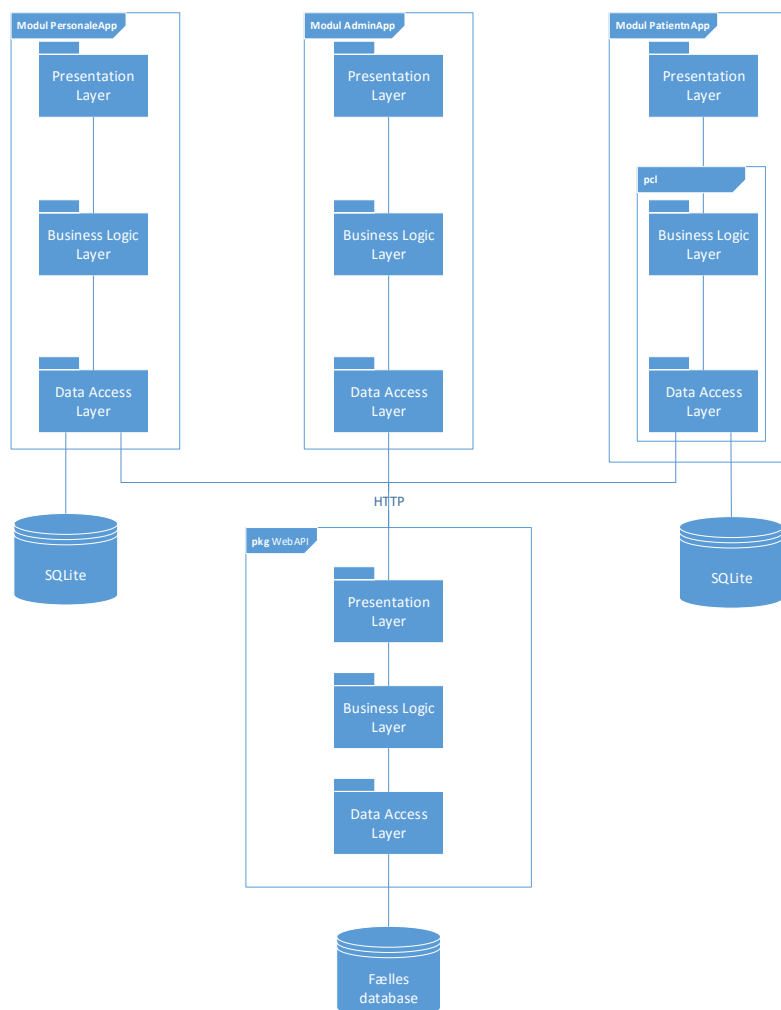
Ud over en fælles database er der også behov for at kunne persistere data lokalt på PatientApp og PersonaleApp så brugerne af app'en kan se det seneste data der er blevet udvekslet mellem modulerne i tilfælde hvor der fx ikke er adgang til internettet. Det er hensigtsmæssigt for brugervenligheden og det opretholder gennemsigtigheden, dvs. at app'en stadig kan tages i brug men at funktionaliteten er begrænset.

Trelagsarkitektur

Arkitekturen i systemets moduler er lavet ud fra trelagsmodel-tankegangen, hvor de tre lag er: *presentation layer*, *business logic layer* og *data access layer*. Dette gør det muligt for udvikleren at fokusere på et område ad gangen, når kode skal implementeres, da de tre lag så vidt muligt holdes adskilt.

Presentation layer er det lag brugeren interagerer med og håndterer modtagelse og præsentation af data. *Business logic layer* er det lag der håndterer udvekslingen af data mellem *presentation layer* og *data access layer*. Når brugeren interagerer med app'en kaldes der funktioner i *business logic layer*. F.eks. når en patient indtaster CPR-nummer i PatientApp vil CPR-valideringen ligge i dette lag.

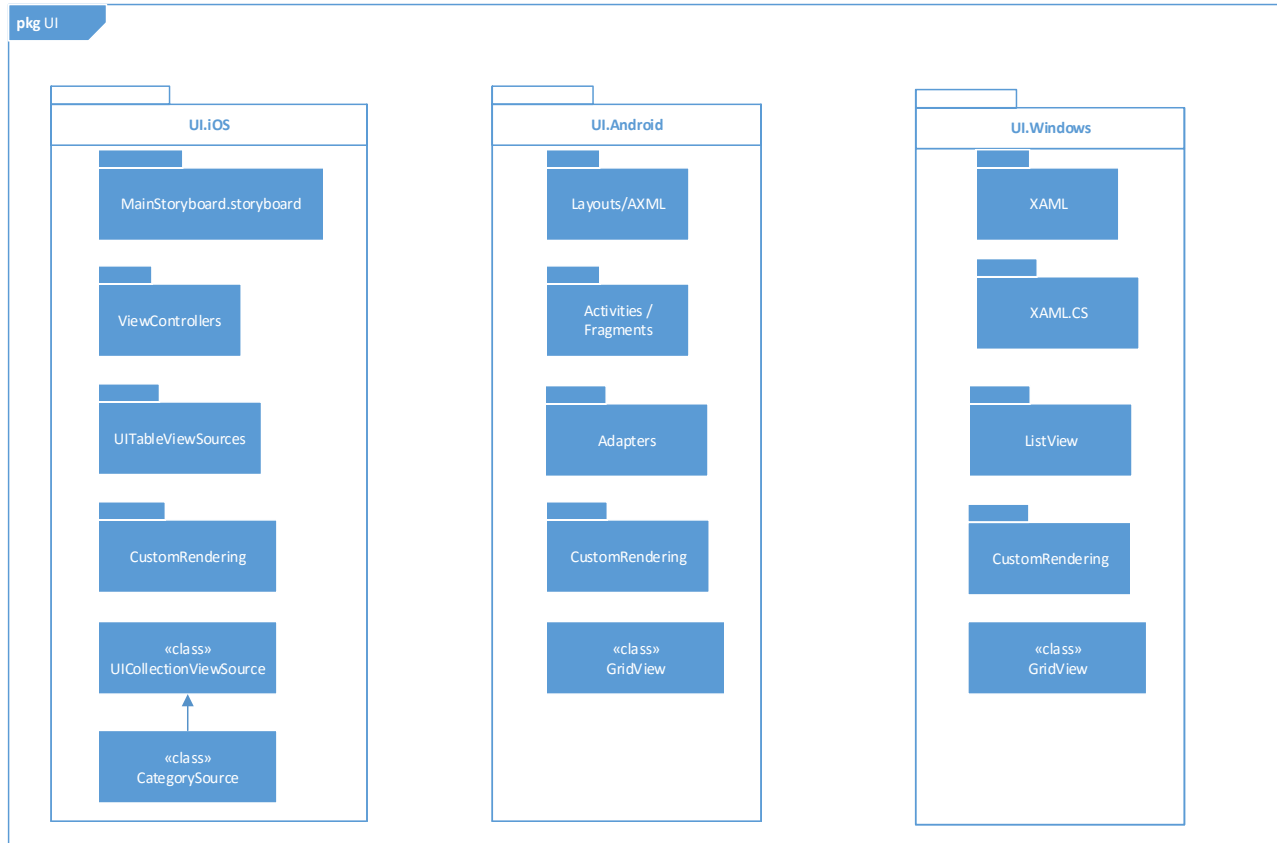
Data access layer håndterer udvekslingen af data til det sted data lagres. Trelagsarkitekturen for hvert modul ses på figur 20.



Figur 20 Trelagsarkitekturen for hvert modul

PatientApp

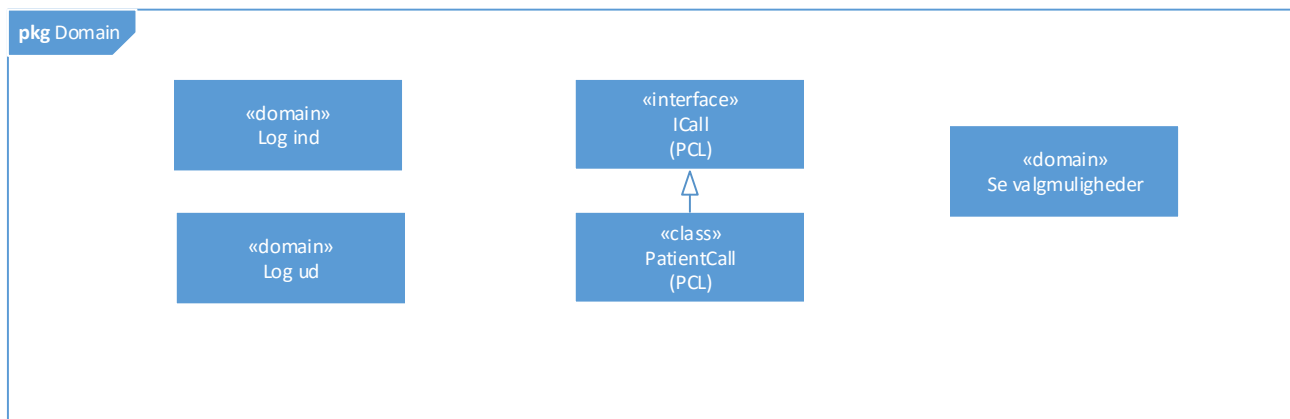
Presentation layer i PatientApp har nogle forskellige pakker for hver platform (se figur 21). Disse pakker indeholder en række forskellige controllers og andet logik der tilsammen danner en brugergrænseflade, der kan interageres med.



Figur 21 Pakker i presentation layer for hver platform af PatientApp

Hver platform har sin egne API'er og biblioteker til fx at vise et view. På iOS bruges *Storyboards*. På Android bruges *Layouts/AXML* og på Windows bruges *XAML*. Logikken bag hvert view, ligger i controllers. På iOS bruges ViewControllers, på Android bruges Activities og/eller Fragments og på Windows bruges Code-Behind c-sharp filer. Derudover er der andre *custom renderinger* dvs. forskellige måder at personalisere en menuknap eller en tabel.

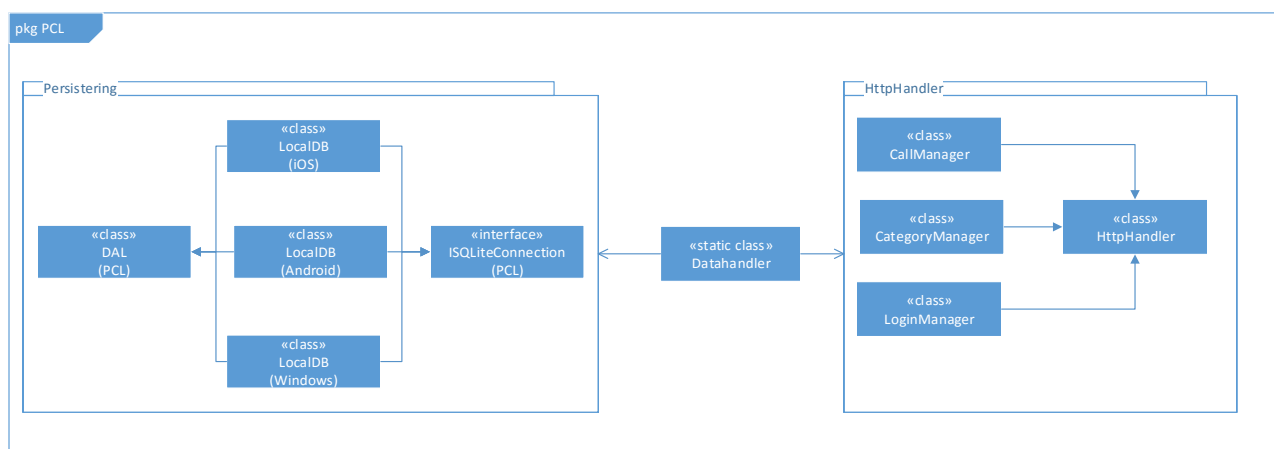
Ligesom *Presentation layer* har pakker, der indeholder andre pakker, er det på samme vis i *business logic layer*.



Figur 22 Business logic layer for PatientApp

Figur 22 tager udgangspunkt i domænet, hvor patienten opretter et kald. Det ses at *ICall* er et interface og at *PatientCall* er klassen som implementerer dette interface. Årsagen til at *ICall* er lavet som et interface, er fordi, at det er tænkt som et kald både i form af et portør kald fra *Columna Service Logistics* men også i form af et patientkald. Det opstiller nogle definitioner for en gruppe af relaterede funktionaliteter som en klasse skal implementere.

Data acces layer for PatientApp består af to pakker: En pakke der håndterer persistering af data og en pakke der står for kommunikationen til omverdenen. Hver af disse er pakket ud til klasser som ses på figur 23.



Figur 23 Data acces layer for PatientApp

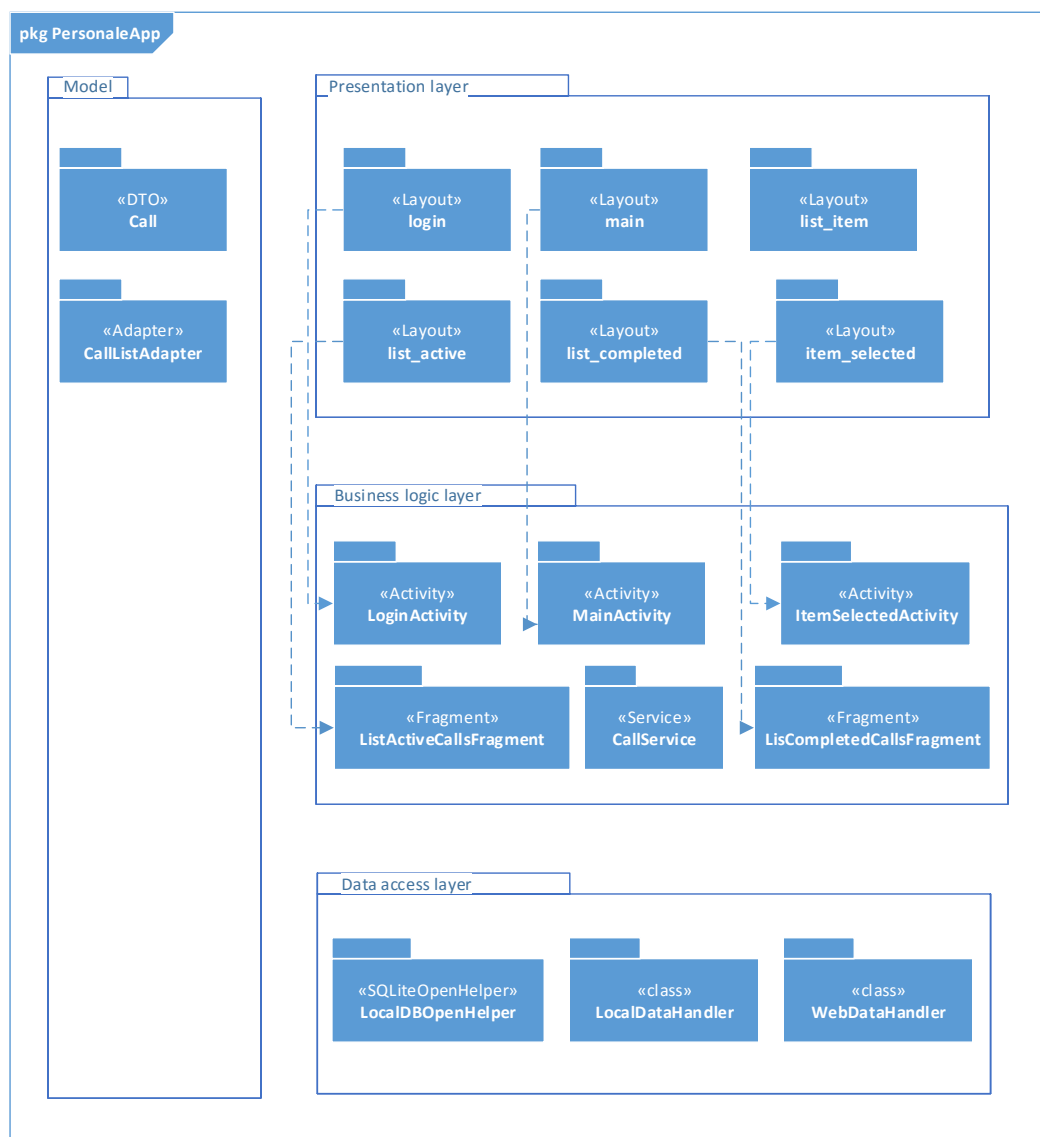
Metoderne til at oprette tabeller, indlæse og gemme data kan alt sammen tilgås via den samme klasse, *LocalDB* (se figur 23). Dette kan lade sig gøre fordi, der findes et API til SQLite, der gør muligt at gemme data i SQLite3 databaser med C# og kan kompileres til at virke på alle platforme (11).

Pakken der hedder *HttpHandler* sørger for at lave http-requests ud til et WebAPI, der sender respons tilbage i JSON-format, der derefter vil blive deserialiseret til konkrete objekter i *Data acces layer*.

Alt dataudtræk fra WebAPI og lokaldatabasen holdes styr på i en klasse kaldt *Datahandler*. Derved når man som udvikler har behov for enten at indlæse patientkald fra lokaldatabasen eller sende et kald til WebAPI, kan udvikleren finde den rette metode i denne klasse.

PersonaleApp

Figur 24 viser pakkediagrammet for PersonaleApp opdelt efter trelagsarkitekturprincippet. *Presentation layer* består af layouts som skal definere den visuelle struktur af brugergrænsefladen. *Business logic layer* består af activities og fragments som skal håndtere logikken bag layouts og dermed de operationer der aktiveres gennem *Presentation layer*. I dette lag skal der også køre en service, som håndterer modtagelse af patientkald uforstyrret af brugerens interaktion med app'en. *Data access layer* består af klasser der skal sørge for adgang til udveksling af data, lokalt på personalets smartphone og med resten af systemet via WebAPI. Ud over de tre lag er der også modeller der kan operere frit mellem lagene. Dette gælder en DTO for Call der skal transportere data for et kald samlet og sikkert gennem app'ens lag og en adapter der skal bruges til at indsætte den rette data i et bestemt view for hvert kald i *Presentation layer*.

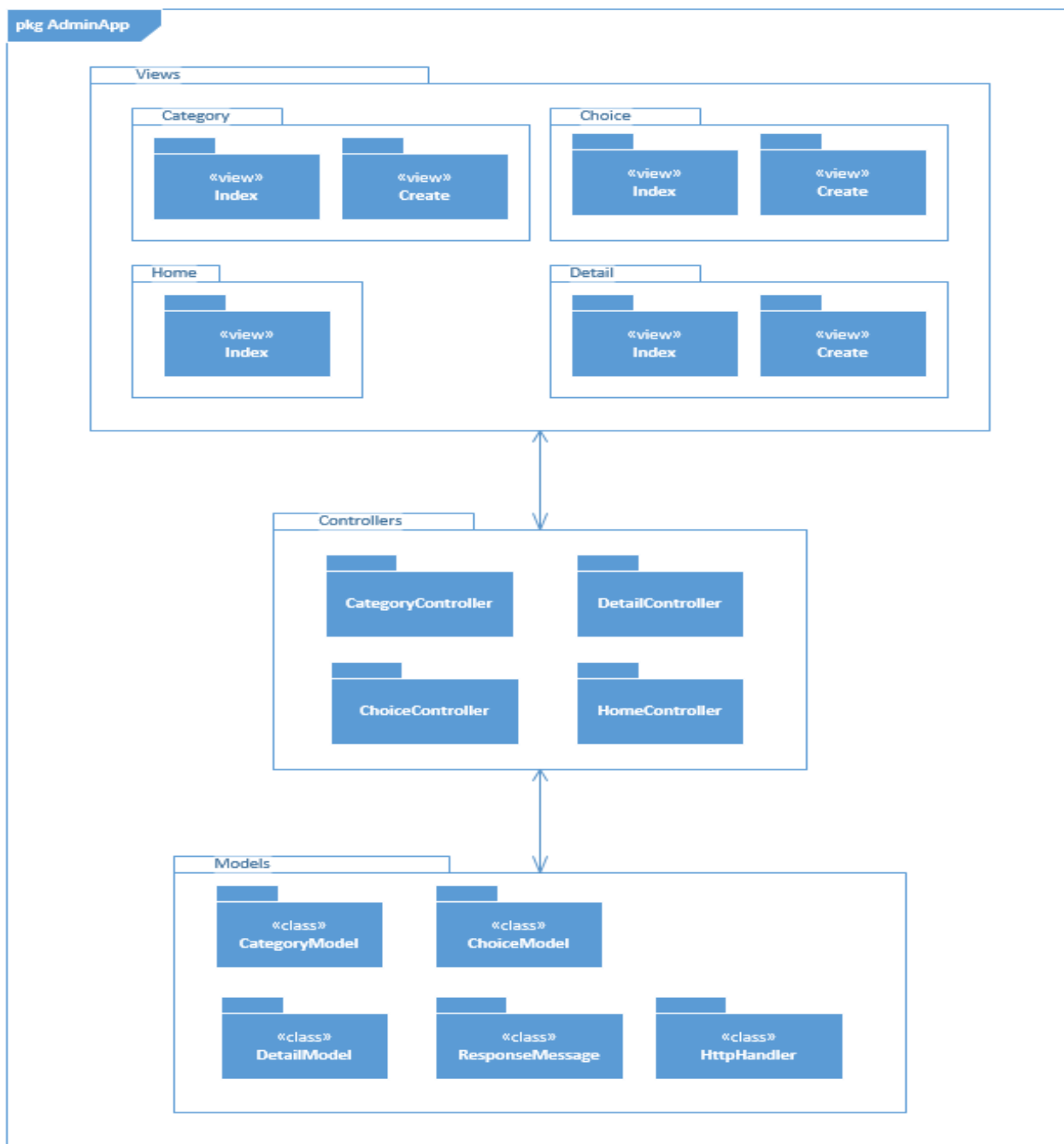


Figur 24 package diagram PersonaleApp

AdminApp

Figur 25 viser trelagsarkitekturen for AdminApp som bygger på MVC strukturen. MVC strukturen består af *views*, som er det brugeren bliver præsenteret for og kan interagere med, *controllers*, som er den logik der

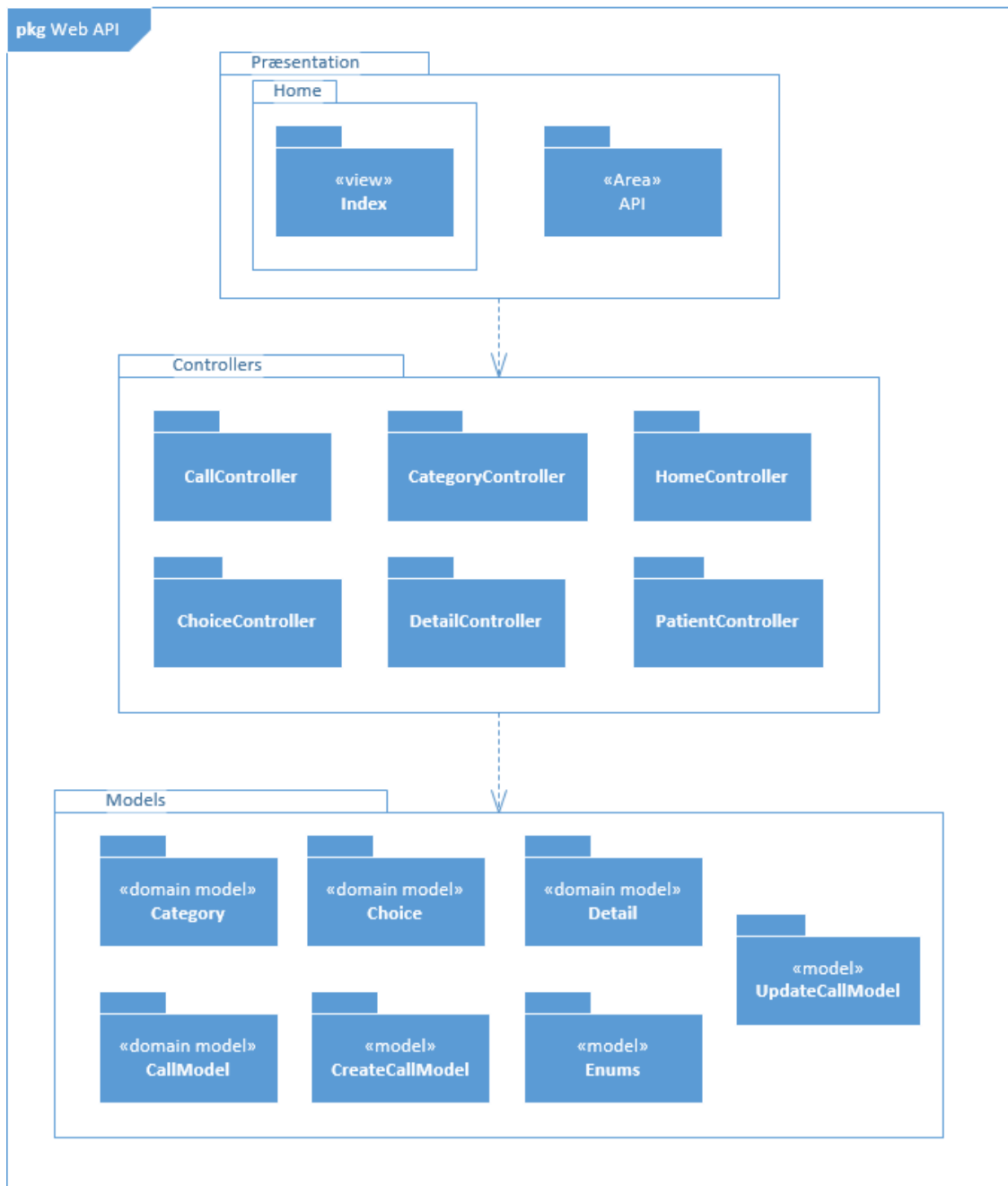
tager sig af brugerens interaktion med præsentations lagets views. Det kan fx være form data der bliver sendt til controlleren. MVC strukturen bygger på én controller for hver view-pakke. *Models*, er de modeller som indeholder controller benytter sig af til at vise data til viewet eller gemme data sendt fra viewet til controlleren. *Models*, består også af evt. hjælpeklasser til fx at sende data til WebAPI eller hente data ind fra WebAPI.



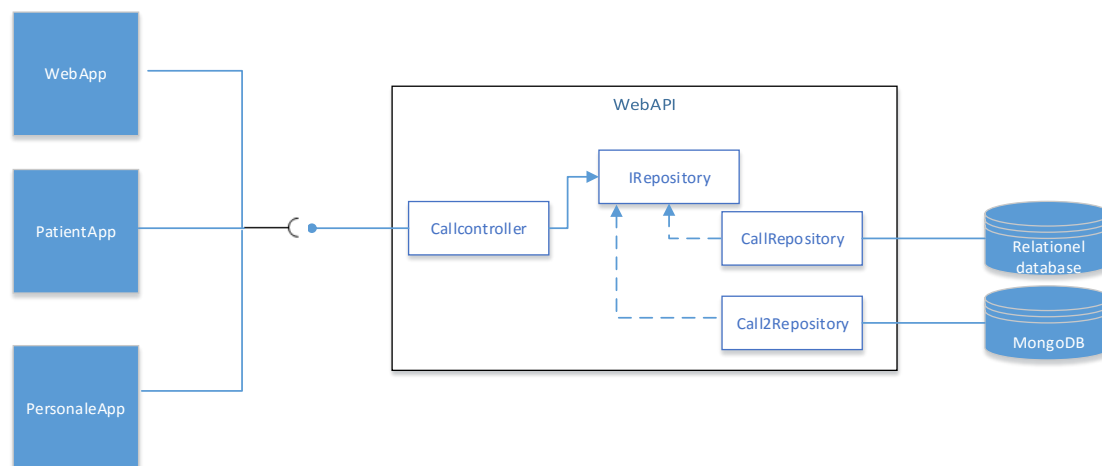
Figur 25 Package diagram AdminApp

WebAPI

Figur 26 viser trelagsarkitekturen for WebAPI som bygger på MVC strukturen. MVC strukturen består af views, som i denne figur er beskrevet som præsentation. Det er gjort fordi API'et ikke benytter sig af views på samme måde som fx MVC strukturen for AdminApp. Præsentation ses her som værende WebAPI'ets dokumentations side, som beskriver kommunikationen med WebAPI. Controllers, som tager sig af http requests til WebAPI. Models, som er WebAPI'ets datamodeller som bliver modtaget fra PatientApp eller PersonaleApp, eller de modeller som bliver persisteret i databasen.



Figur 26 Package diagram for WebAPI



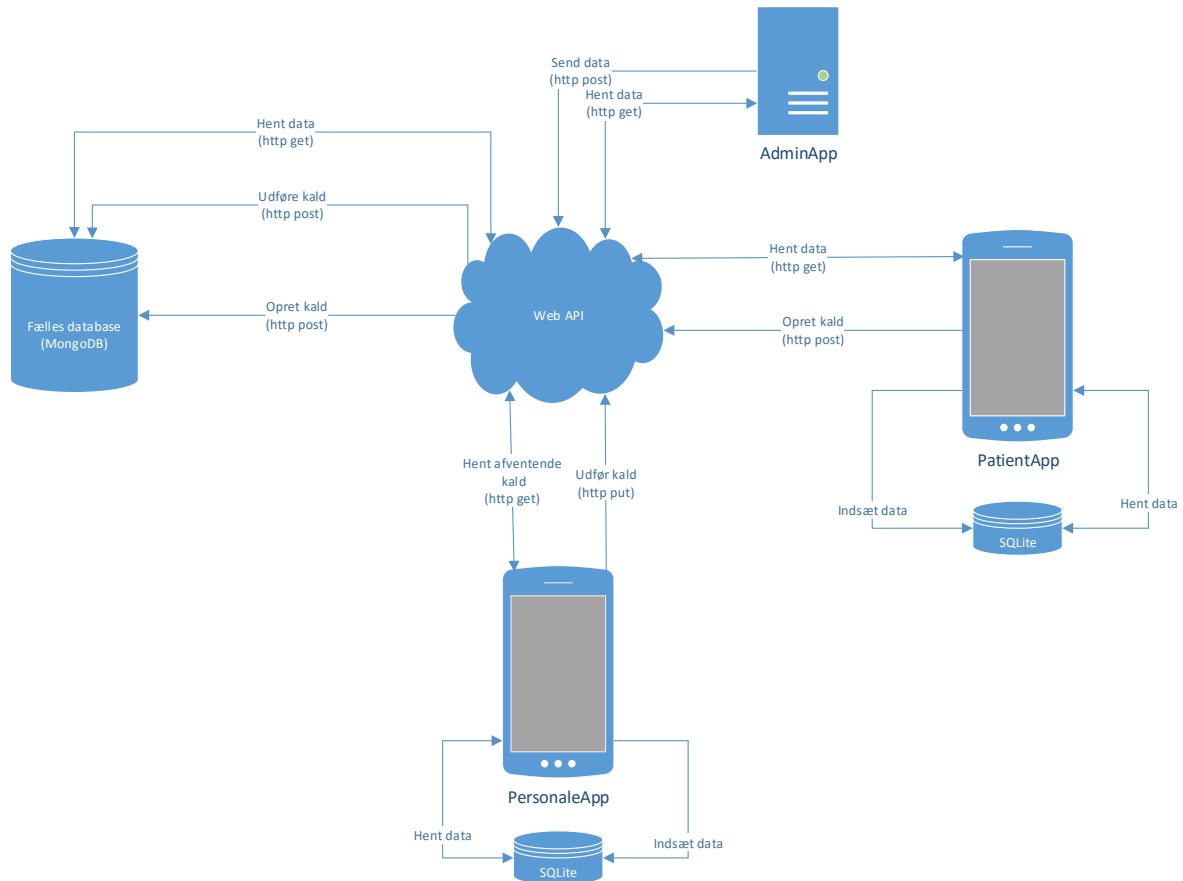
Figur 27 Repository pattern

WebAPI benytter et *repository pattern* som vist på figur 27, hvilket betyder at der opnås lav kobling mellem databasen og resten af WebAPI'ets business logik. Ideelt set betyder det at databasen kan udskiftes uden det har betydning for resten af systemet, såfremt datastrukturen bevares. Ved et eventuelt databaseskift, vil man ikke skulle ændre noget i den eksisterende kode men blot lave en ny implementering af sit *repository pattern* så systemet kan tilgå den nye database.

8 Design

Som følge af systemarkitekturen er næste fase i udviklingsprocessen design af PatientCare. Dette afsnit beskriver flowet mellem modulerne og designet af hvert modul.

I figur 28 ses flowet af data mellem systemets moduler.



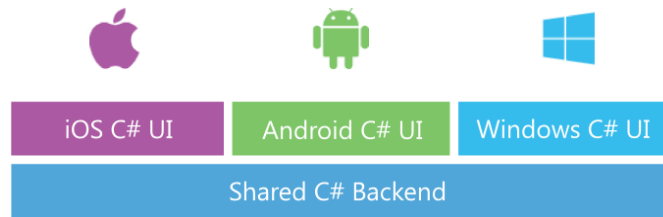
Figur 28 flow af data mellem systemets moduler

Det ses at hvert modul kommunikerer med hinanden via WebAPI, som fungerer som systemets grænseflade ud til modulerne. Fx når der i PatientApp oprettes et kald, sendes det til WebAPI, der skubber kaldet videre til fællesdatabasen. Det oprettede kald vil ligge i fællesdatabasen, som WebAPI tilgår når PersonaleApp efterspørger afventende kald.

PatientApp

Cross platform

Løsningen som PatientApp benytter sig af, er den traditionelle Xamarin fremgangsmåde. I denne fremgangsmåde er hvert operativsystems brugergrænseflade(UI) udviklet til at have en *fælles backend* som vist på figur 29. Det medfører at alle operativsystemer kan bruge samme logik og der undgås kodedublering. Det er også i den *fælles backend* data bliver sendt og hentet via WebAPI.



Figur 29 Xamarin, fælles backend

PatientApp er delt op efter trelagsarkitekturen som beskrevet under systemarkitektur. *Presentation Layer* består af pakker af forskellige API'er og biblioteker der sammen med tilhørende logik udgør en brugergrænseflade, som patienten kan se. Dette lag vil blive vist forskelligt afhængig af hvilken platform PatientApp er installeret på. *Business Logic Layer* indeholder logikken for PatientApp med udgangspunkt i use casene. Patienten skal fx oprette et kald og den logik der sørger for dette ligger i dette lag. *Business Logic Layer* og *Data Acces Layer* er de lag som bliver brugt af de forskellige platforme og kaldes tilsammen et *Portable Class Library* (PCL).

PCL funktioner:

- Indlæs/gem data herunder valgmuligheder og mine kald
- HTTP requests til og fra WebAPI herunder *Object Relationel Mapping*(ORM)
- CPR validering via WebAPI
- Andet logik (klasse til at pakke valgmuligheder til et kald, der senere skal persisteres)

Hver platform skal implementere:

- Egen lokal database
- Filstien til hvor hver database lokalt skal ligge.
- Oprettelse til databasen
- Brugergrænseflade (menuknapper, dialogbokse osv.)

De designmæssige valg, der er blevet taget i forbindelse med persistering kommer frem til en teknologi som SQLite gør brug af. Her vil det være muligt at mappe data fra lokaldatabaserne direkte til klasser. Hvorefter disse klasser kan traversere rundt på tværs af iOS, Android og Windows via PCL og benyttes i logikken og vises på brugergrænsefladen på PatientApp.

PCL og SQLite

For PatientApp har projektgruppen valgt at persistere data lokalt i en SQLite database. Da logikken for PatientApp skal bygges på en PCL, skal persistering af data dermed også være af samme fremgangsmåde og bygges i samme PCL. Til dette er der valgt en SQLite-Extension¹, som er en meget simpel ORM der tilbyder alle de relationer *databasemodellen* har brug for, af et SQLite-net² bibliotek. Det er open source, der tillader .NET og monoapplikationer at gemme data i SQLite3 databaser. Med SQLite - Net Extension udvides SQLite3's funktionalitet og hjælper bedre udvikleren til at håndtere relationer mellem SQLite-net entiteter.

¹ <https://bitbucket.org/twincoders/sqlite-net-extensions>

² <https://github.com/praeclarum/sqlite-net>

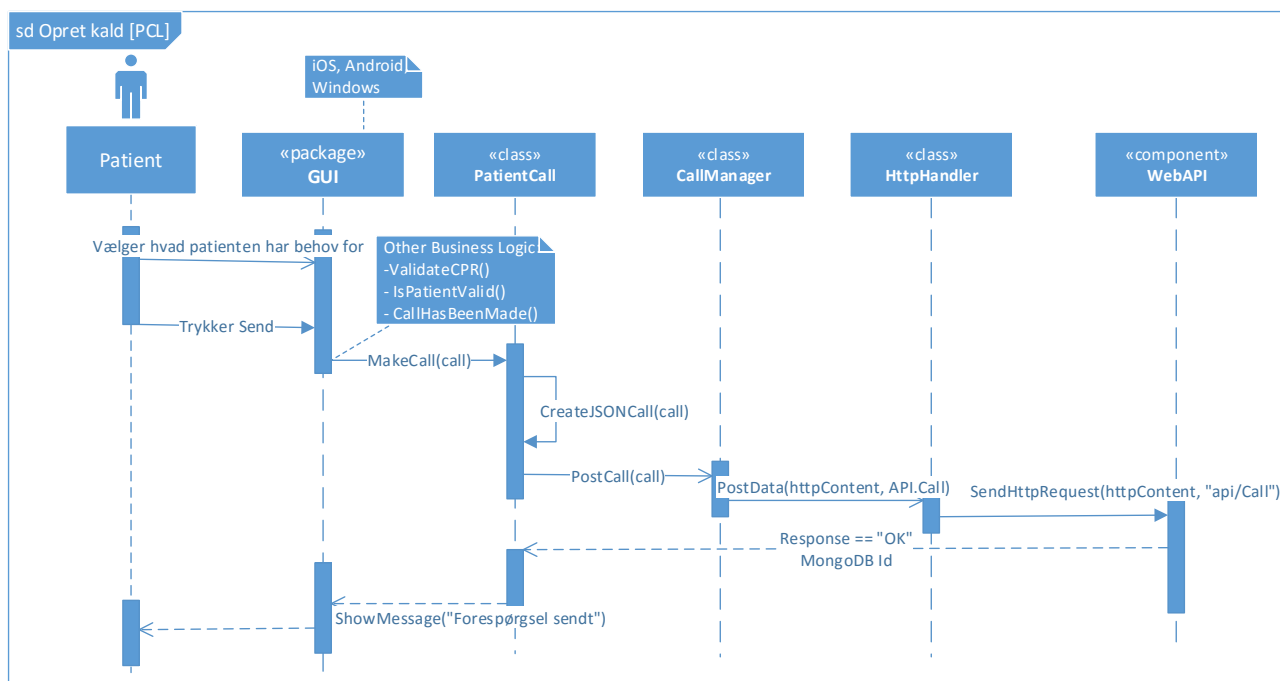
Med SQLite – Net Extension kan man via attributter på sine properties angive relationerne. Man skal ikke oprette tabeller eller kolonner i databasen og derfor har man fuld kontrol over databasens skema til at persistere entiteterne. SQLite-Net Extensions kræver kun, at man angiver fremmednøgler, der anvendes til at håndtere relationer og resten finder SQLite3 den selv ud af.

Et eksempel på brugen af SQLite-Net Extension er metoden *GetAllWithChildren<Category>*. Metoden kigger på alle de relationer der er specificeret i databasemodellen, finder eventuelle fremmednøgler og automatisk fylder properties på entiteten, der er i dette tilfælde er *Category*. Man slipper altså for at skrive queries, som man normalt ville med SQLite3.

Hver platform implementere en metode til at returnere filstien for hvor SQLite databasen lokalt skal ligge. Samtidig skal hver platform implementere en metode til at oprette forbindelse til SQLite databasen. Dette kræver et interface som implementeres af tre platformsspecifikke klasser med hver af deres SQLiteConnection. I *Data Access Layer* laget ligger der en klasse, der har en række metoder som skal bruges ved indlæsning og persistering af data fra og til SQLite databaserne. Denne klasse skal også bruges i de tre platformsspecifikke klasser.

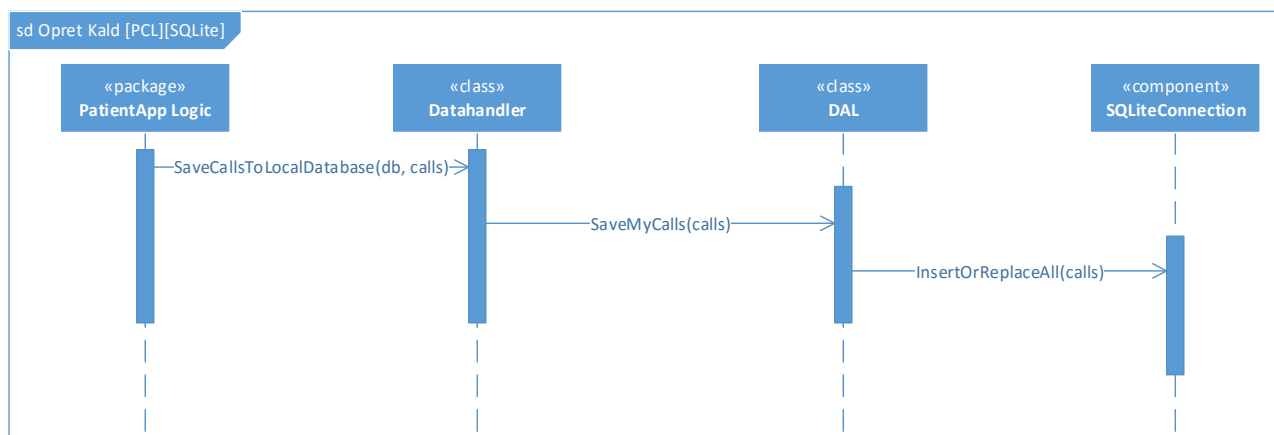
Sekvensdiagrammer (opret kald)

For at illustrere hvordan de forskellige lag snakker sammen i PatientApp er der taget udgangspunkt i use case 1.3: Opret kald. Kommunikation mellem disse tre lag vises med et sekvensdiagram på figur 30 der beskriver sekvensen fra en patient opretter et kald til patienten kan se kaldet er oprettet. Brugergrænsefladen (GUI) er en pakke af forskellige komponenter på hver af de forskellige platforme som initialiserer logikken i PCL. Komponent, WebAPI, er ikke en del af PCL, men er taget med i diagrammet for at vise forbindelsen til resten af systemet.



Figur 30 sekvensdiagram over UC 1.3 Opret kald

På figur 31 vises sekvensen for hvordan kaldet persisteres i lokaldatabase efter det er blevet oprettet. Komponenten, *SQLiteConnection*, er et API til SQLite.Net der er inkluderet i PCL.



Figur 31 Sekvensdiagram for hvordan kaldet persisteres i lokaldatabase

PersonaleApp

I systemarkitekturen blev systemets pakker identificeret og det blev beskrevet hvad de skulle bruges til. I dette afsnit beskrives designet af PersonaleApp og hvordan funktionerne skal implementeres i den fastsatte struktur.

Layouts, activities og fragments

Layoutsne i *Presentation layer* er designet i XML-filer, én fil for hvert layout der er behov for. Det har resulteret i syv xml-filer. Den tilhørende logik i *Business logic layer* er designet som activities, der som udgangspunkt håndterer logikken for hvert layout, men der er også anvendt fragments. Fragments lever i en aktivitet men har egen livscyklus. Dermed behøver man kun at udskifte det der er forskelligt på layoutsene og på den måde undgås kodeduplikering. Fragments er anvendt i tilfælde hvor der skal tilgås samme funktioner på flere layouts. Dette er tilfældet for layoutsne for *afventende*- og *udførte* patientkald hvorfra man skal kunne tilgås samme menu. Disse kan med fordel leve i hver deres fragment, men i samme aktivitet, så menuen kan tilgås uagtet af hvilket layout brugeren ser.

SQLite Database

For PersonaleApp har projektgruppen valgt at persistere data lokalt i en SQLite database. SQLite databasen har en relationel databasestruktur med tabeller og kolonner som er anvendt fordi der er behov for at gemme flere parametre for et kald lokalt (se hvilke parametre der gemmes på PersonaleApp i under afsnittet *Systemarkitektur*). Disse parametre kan nemt tilgås ved kendskab til et id, da der er lavet en tabel kun for kald.

Data acces laget

Data acces layer håndterer kommunikationen med den lokale SQLite database som instansieres i klassen *LocalDBOpenHelper*. Alt dataudtræk fra SQLite databasen holdes der styr på i en klasse kaldt *LocalDataHandler*. Laget håndterer også adgangen til udveksling af data til og fra WebAPI i klassen *WebDataHandler*.

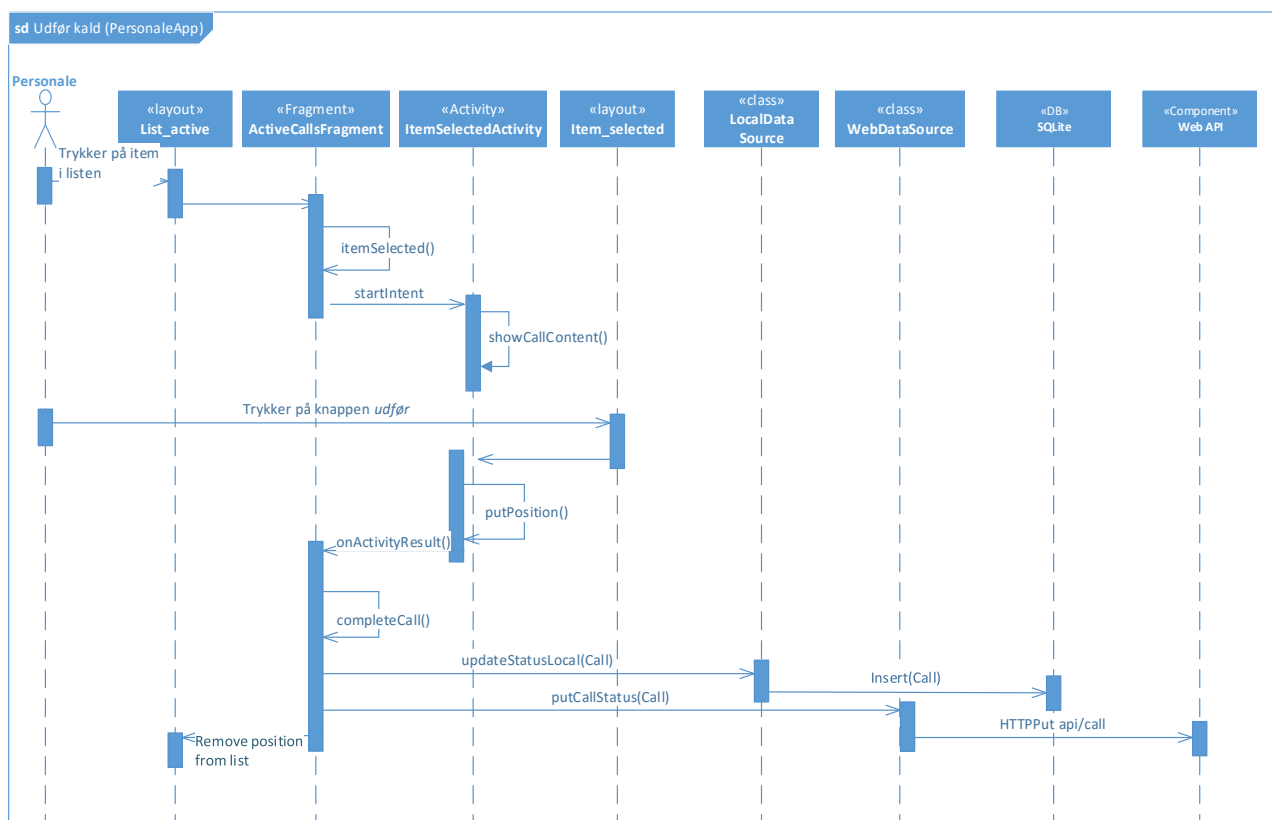
Design af funktioner

Nogle af de vigtigste funktioner PersonaleApp skal tage sig af er at modtage patientkald og gøre det muligt for personalet at udføre dem.

Udfør kald

For PersonaleApp har designprocessen taget udgangspunkt i kravspecifikationen hvilket har givet anledning til en række sekvensdiagrammer. De opgaver som er forbundet med et patientkald skal modtages på PersonaleApp og udføres af personalet. Med udgangspunkt i UC 2.3: Udfør kald laves et sekvensdiagram der beskriver sekvensen for at et kald udføres i PersonaleApp.

Figur 32 viser et sekvensdiagram for når et kald udføres. Når patienten har sendt et patientkald og personalet har modtaget det via WebAPI, kan personalet vælge at udføre det på layoutet *item_selected*.



Figur 32 Sekvensdiagram for UC 2.3 Udfør kald

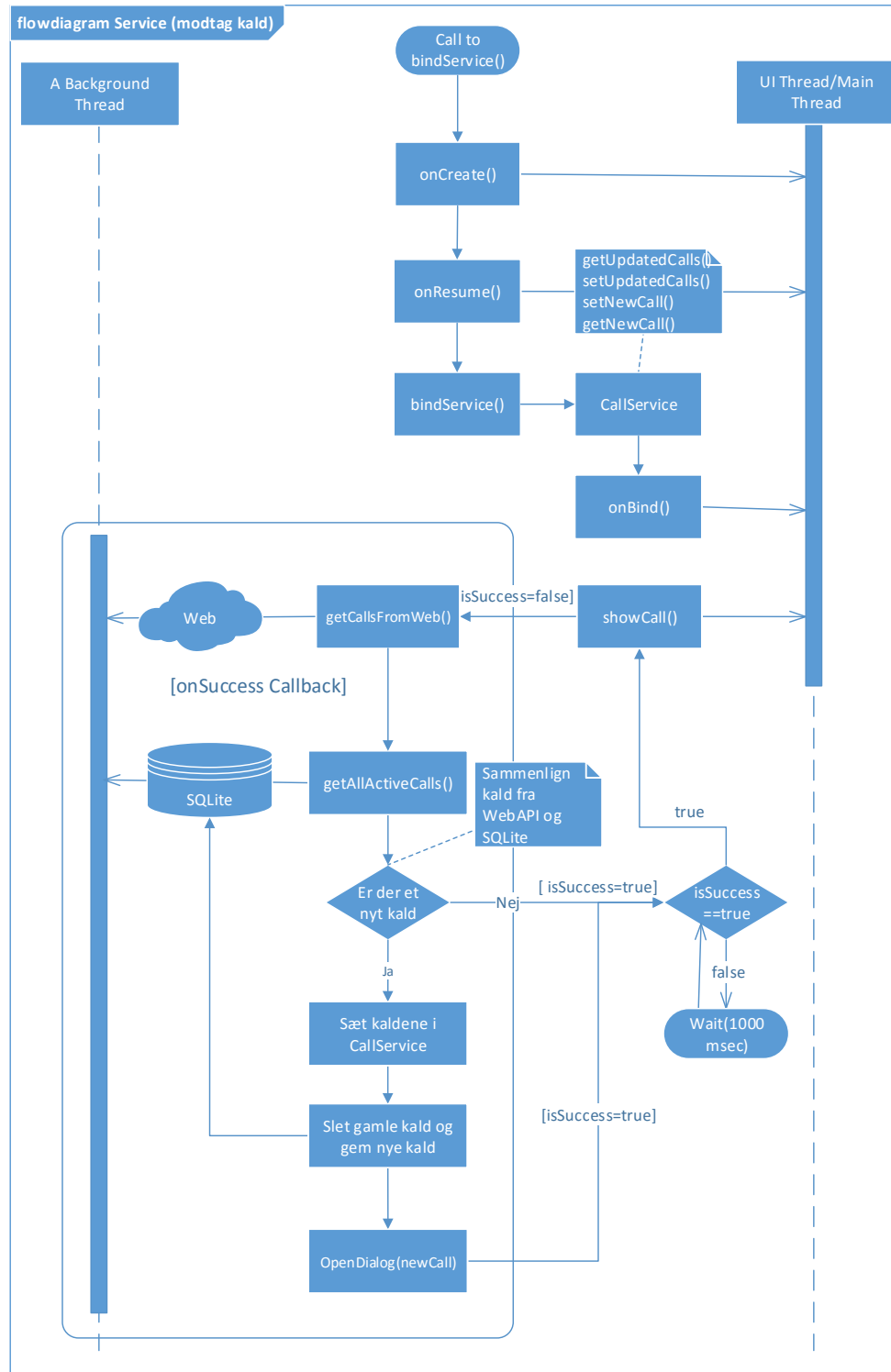
Modtag kald (Bound Service)

Der er brug for en service der kan opdatere listen med afventende kald og notificere personalet når der er kommet et nyt kald. Da det er prioriteret at vise konceptet frem for performance, er der valgt en pull-funktion til at indlæse nye kald fra systemet. Denne pull-funktion kan laves ved at personalet gør noget aktivt eller det kan håndteres med en service. Da en service kan håndtere baggrundsopgaver, vil der ikke være behov for at personalet manuelt skal interagere med app'en for at holde sig opdateret med de indkomne kald. Der er valgt at gøre brug af en service for at gøre brugeroplevelsen god. En service i Android Studio kommer i to udgaver: *Started Service* og *Bound Service*. En *Started Service* startes af en aktivitet ved at kalde *startService()* hvorefter servicen uafbrudt vil udføre én operation i baggrunden uden at returnere et resultat til brugeren. I en *Bound Service* bindes en aktivitet eller en komponent til servicen, der tillader et

client-server interface, hvilket gør det muligt at interagere med servicen, sende requests og modtage resultater på tværs af processer. I PersonaleApp er der behov for det sidstnævnte, da der løbende skal hentes patientkald fra en webserver via WebAPI. Derfor er der valgt en *Bound Service*.

Servicen starter i *MainActivity*, som er den aktivitet der starter når brugeren af PersonaleApp er logget ind. Servicen håndterer opgaven at hente alle afventede patientkald fra WebAPI'et via et HTTP request til en URL. WebAPI'et returnerer en string af JSON objekter som deserialiseres til konkrete objekter.

I figur 33 ses det hvordan servicen bliver oprettet i PatientApp. Til højre i diagrammet ses tidslinjen for aktiviteter og andet logik der kører i hovedtråden (*MainActivity*). Til venstre ses tidslinjen for en separat tråd hvor de aktiviteter servicen udfører kører på. Servicen kører på en anden tråd end hovedtråden så den ikke laver blokerende operationer på hovedtråden.

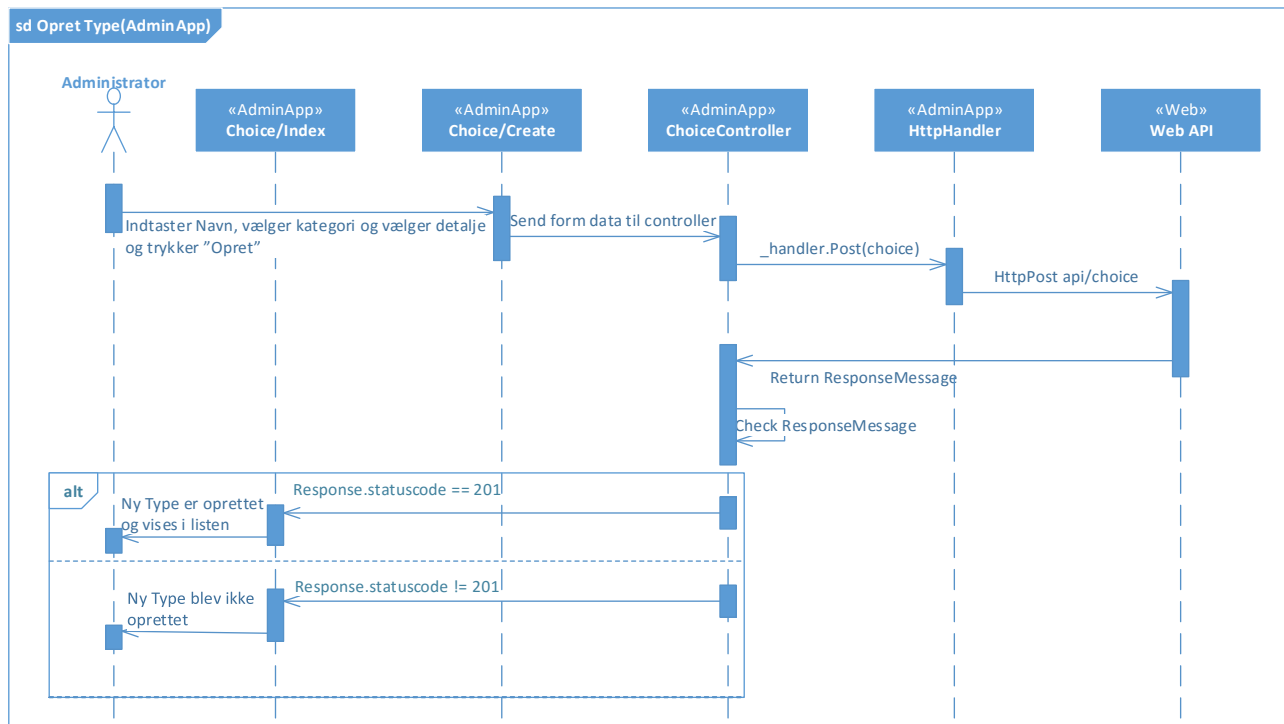


Figur 33 Flowdiagram Service (Modtag kald)

Requests til webserveren sker asynkront grundet det framework der benyttes til det og det kan tage noget tid at få respons tilbage, derfor må baggrundstråden vente på at operationerne er færdige, før den kan starte næste iteration. Den venter et sekund på at dette er sket hvilket illustreres i figur 33 ved at der tjekkes på en bool variable *isSuccess*.

AdminApp

For AdminApp har designprocessen taget udgangspunkt i kravspecifikationen hvilket har givet anledning til en række sekvensdiagrammer. Sekvensdiagrammerne er lavet ud fra tankegangen om at AdminApp skulle være en web applikation og at applikationen skulle kommunikere med et WebAPI.



Figur 34 Sekvensdiagram for UC 3.2 Opret kald

I figur 34 ses sekvensen for UC 3.2 Opret type. Abstraktionsniveauet i sekvensdiagrammerne for AdminApp, er generelt sat forholdsvis højt og giver ikke en direkte mapning til implementeringen. Formålet med denne fremgangsmåde er at forsøge at gøre diagrammerne fleksible i forhold til implementeringen. Med andre ord, hvis en use case ændres så meget at use casens sekvens udførelse også bliver ændret, vil det kræve ændring i diagrammet også. Det ses fx i kaldet fra *choice/create viewet* hvor form data bliver sendt til en *choicecontroller*. Hvad der sker inde i *choicecontrolleren* bliver der ikke taget stilling til i sekvensdiagrammet. Det eneste sted der i diagrammet er taget stilling til hvilket funktionskald der sker, er i klassen *HttpHandler* hvor kommunikationen ud af systemet sker. AdminApp kommunikerer med WebAPI og måden denne kommunikation sker på, ændres ikke.

Views, Controllers og *Models* i AdminApp, følger MVC standarden hvor *views* er opdelt efter hvilken del af applikationen de står for. Fx ligger alle *views* som har noget med *category* at gøre, i en mappe der hedder *Views/category*. Dette giver en god mappe struktur og det er nemt for en andre, der er bekendte med MVC sturkturen, at sætte sig ind i strukturen af applikationen.

Ved oprettelse af kategorier skal der bruges et navn for kategorien og en URL til det billede som skal vises på PatientApp. Kategorier kan have mange typer under sig og en type kan have mange detaljer. I design afsnittet for databasen, kan det ses at der det er tænkt at have en mange til mange tabel til at stå for denne kobling mellem typer og detaljer. Alternativet til dette er at oprette en detaljer for hver type, selv om detal-

jen hedder det samme. Fx kan kategorien drikke, have typerne kaffe og te, hvor det kunne være muligt at få sukker til kaffe eller te. Uden denne mange til mange relation vil det kræve at kaffe og te hver har deres sukker relateret til sig. I AdminApp laves denne relation mellem typer og detaljer ved oprettelse af en type. En type hører ind under en kategori og en type kan have ingen eller mange detaljer tilknyttet. På denne måde undgås det at have flere detaljer for hver type.

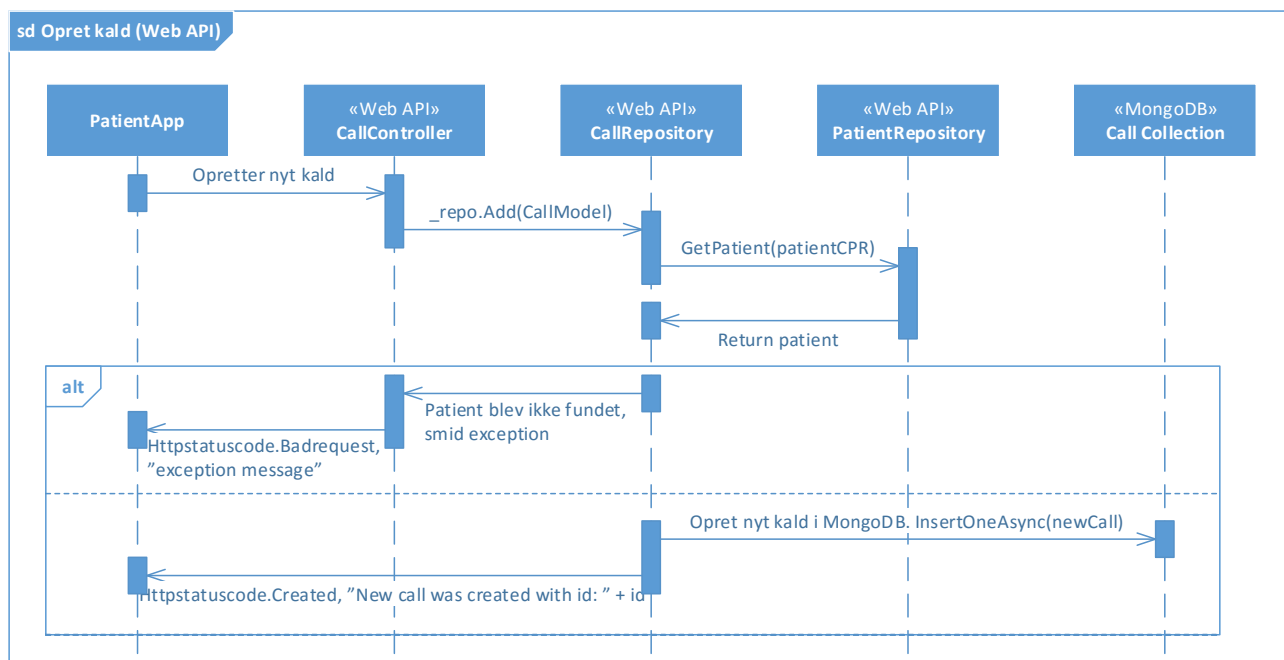
```
{
  "ChoiceId": "564c3bac4ca8e951dcb1ac66",
  "Name": "Kop kaffe",
  "CategoryId": "564ae9224ca8e912c4b39268",
  "Details": [
    {
      "DetailId": "564ae93e4ca8e912c4b39269",
      "Name": "Mælk"
    },
    {
      "DetailId": "564b24ae4ca8e917f8472c0d",
      "Name": "Sukker"
    },
    {
      "DetailId": "564c3b1a4ca8e951dcb1ac60",
      "Name": "Mælk og sukker"
    }
  ]
},
```

Figur 35 - Viser et eksempel på en type fra MongoDB

I figur 35 ses et eksempel på en oprettet type. Det ses at typen har en liste af detaljer og det er denne liste som fungerer som mange til mange relationen.

WebAPI

For WebAPI'et har designprocessen taget udgangspunkt i alle use cases i kravspecifikationen hvor et device har behov for at kommunikere ud af eget system. Det er fx ved oprettelse af et kald fra PatientApp.



Figur 36 Sekvensdiagram for Opret kald set fra WebAPI'ets synspunkt

Sekvensdiagrammet i figur 36 beskriver sekvensen konceptuelt og er lavet ud fra den tankegang om at udvikleren af systemet ikke skal være bundet op af hvordan diagrammet ser ud, men at diagrammet skal være en hjælp til sekvens implementering. Det at diagrammet er konceptuelt og ikke viser hver enkelt funktionskald er at diagrammet skal være åben for ændringer. I ovenstående diagram ses PatientApp som en black box og viser derfor kun hvad WebAPI'et gør når et nyt kald bliver sendt.

ASP.NET WebAPI udstiller et bibliotek til at autogenerere en hjælpeside på runtime, for dokumentation. Her vil man fx kunne se hvilke parametre der forventes, når der skal kommunikeres med WebAPI.

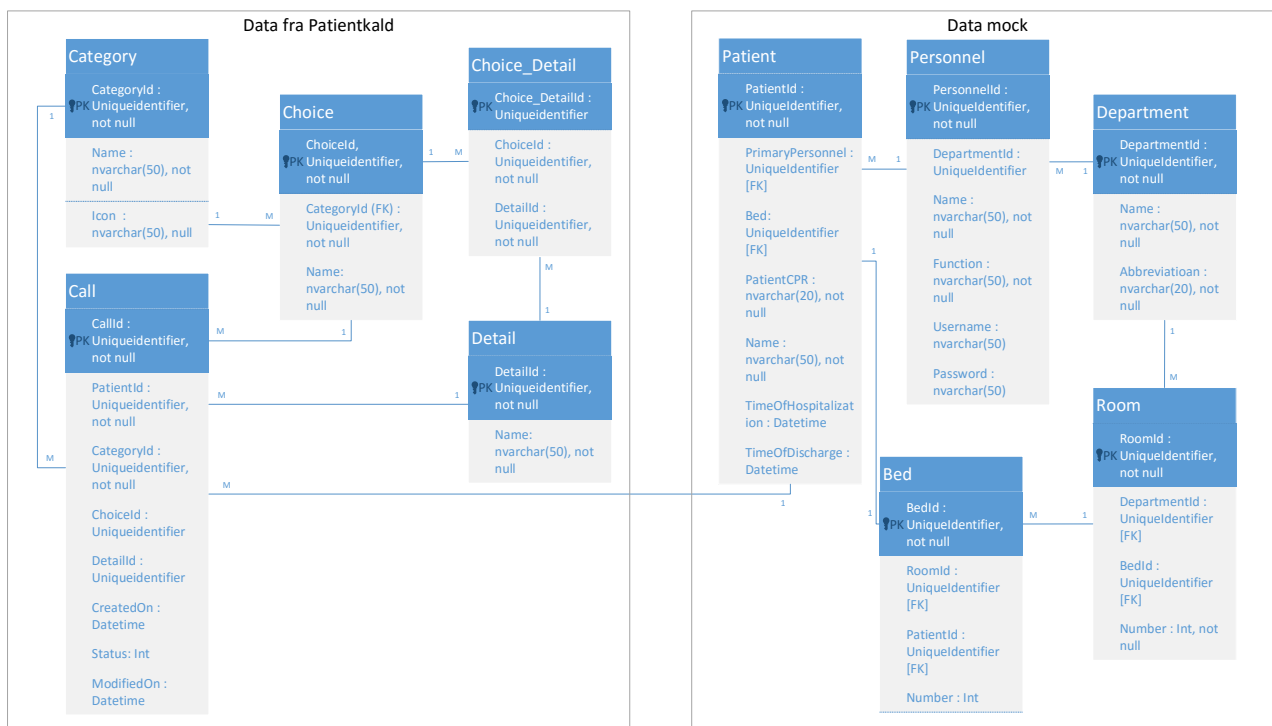
Database

I systemarkitekturen blev der gjort rede for hvad for noget data der skal persisteres i systemet. Der blev identificeret to datakilder, hvor den ene er mock data og den anden er data fra patientkald som genereres når et patientkald oprettes på PatientApp. Hvordan datamodellen skulle se ud blev besluttet i designfasen.

Databasestruktur

Projektgruppen havde fra starten af struktureret data efter relationer, som med fordel kunne lagres i en relationel database bestående af tabeller, kolonner og rækker. Derfor blev der valgt en relationel database, hvor der ikke umiddelbart bruges relationer.

Figur 37 er lavet ud fra den oprindelige tankegang om relationer mellem data.



Figur 37 Relational databasestruktur

Midt i udviklingsfasen blev der skiftet databasestruktur fra en relationel database til en NoSQL database, hvilket har givet anledning til at tænke databasestrukturen på en helt anden måde. Denne databasestruktur er beskrevet i følgende afsnit om MongoDB (Læs mere om databaseskiftet under afsnittet *Diskussion*).

MongoDB

Data persisteres på nuværende tidspunkt i en fællesdatabase MongoDB af typen NoSQL. Denne form for database er en cross-platform dokumentorienteret database der består af *collections* og *documents*, hvilket afviger fra tankegangen med en relationel database.

Patientkald genereret fra PatientApp og data mock af data fra Cetrea bliver gemt i hver deres database. Begge på mongolab.com. I databasen med patientkald fra PatientApp gemmes en collection af *Calls*, *Categories*, *Choices* og *Details*. Mens der i databasen for data mock gemmes en collection af *Departments*, *Patients*, *Personnel*, *Rooms* og *Beds*.

På figur 38 ses de data som et kald indeholder når det bliver sendt fra PatientApp'en til WebAPI'et, ved oprettelse af et nyt kald.

```
{
  "PatientCPR": "1111111118",
  "Category": "Smerter",
  "Choice": "Efter operation",
  "Detail": "Moderate",
  "CreatedOn": "12:37 PM"
}
```

Figur 38 Data der genereres ved oprettelse af et nyt patientkald på PatientApp

På figur 39 ses de data et dokument for en patient indeholder i *Patient Collection* i databasen for mock data. Det ses også at dette dokument har et id. Dette genereres i MongoDB.

```
{
  "_id": {
    "$oid": "562e5073e4b06ba894cee07b"
  },
  "PatientCPR": "111111118",
  "PatientName": "Louise Andersen",
  "Department": "Gyn-obs",
  "Room": "Stue 2",
  "Bed": 1,
  "ImportantInfo": "Acne"
}
```

Figur 39 Data om en patient fra mock data

Mock data skal nu kobles sammen med Patientkaldet før det kan bruges af personalet.

Den relation der bruges til at koble mock data sammen med data fra et nyt patientkald er CPR-nummeret, som er unikt. Når WebAPI'et har modtaget et patientkald fra PatientApp går WebAPI'et ned i databasen for data mock, finder *Patient Collectionen* og kigger efter PatientCPR der matcher det CPR-nummer som blev sendt med patientkaldet. Hvis CPR-nummeret ikke findes bliver der smidt en exception og fejl besked samt 404 statuscode bliver sendt tilbage. Hvis en Patient med det sendte CPR-nummer findes i databasen, bliver data mock af data for patienten med dette CPR-nummer tilføjet til objektet og gemt i databasen, *Call Collection*.

Figur 40 viser det færdige objekt som bliver gemt i *Call Collection* i databasen. Det består af attributter fra det modtagne patientkald samt data mock for den pågældende patient. Objektet består af al den information som er nødvendigt for personalet.

```
{
  "_id": "564c62e34ca8e93380157bdf",
  "PatientCPR": "111111118",
  "PatientName": "Louise Andersen",
  "Room": "Stue 2",
  "Bed": "1",
  "Department": "Gyn-obs",
  "Category": "Smerter",
  "Choice": "Efter operation",
  "Detail": "Moderate",
  "CreatedOn": "12:37 PM",
  "ModifiedOn": "Wednesday, November 18, 2015 11:38:14 AM",
  "Status": 1
}
```

Figur 40 det samlede data der skal bruges på PersonaleApp

9 Resultater

I dette afsnit beskrives de resultater der er kommet ud af Mini-MTV'en og status på udviklingen af prototypen.

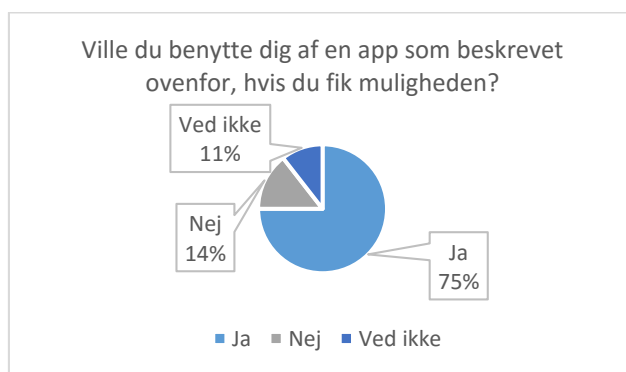
Resultater fra Mini-MTV

Der er indikatorer for at et system som PatientCare vil have **forskellig indvirkning på organisationsniveau, fordi afdelinger har forskellige strukturer og personalet på afdelingerne har forskellige arbejdsgange.**

Sandsynligheden for at der er behov for et system der kan knytte en årsag til patientkald er stor på Gynækologisk – Obstetrisk Afdeling på Regionshospitalet i Randers. Det viser resultater fra Mini-MTV-analysen.

Personalet vurderer at det mere informative patientkald vil gøre dem mere velforberedte til mødet med patienten og at de vil få mulighed for at **prioritere patientkaldene så de vigtigste kan blive udført først.** Det har de ikke kunnet tidligere på grund af manglende information om årsagen til kaldene. Ligeledes vurderer personalet at de kan **sparre en del tid og dermed udnytte deres sygeplejekompetencer mere effektivt.** Også **antallet af skridt kan med stor sandsynlighed reduceres,** sammenlignet med det antal skridt personalet går for et patientkald med det nuværende kaldeanlæg. En estimeret skridt- og tidsbesparelse, som bygger på en vurdering fra en innovationskonsulent fra Regionshospitalet Randers siger at personalet kan **sparre halvt så meget tid og halvt så mange skridt pr. patientkald (12).** Projektgruppen har selv lavet et tidsstudie og sammenlignet tid og antal skridt for et patientkald med og uden tilknyttet årsag. Beregningerne viser i dette tilfælde at der rundt regnet kan spares **3000 skridt og mere end 1 time pr vagt ved implementeringen af PatientCare.** Dog afhænger disse tal meget af hvor patient og personale befinder sig på det givne tidspunkt, hvor store afstande der er på afdelingen og en lang række andre faktorer. Estimatet tager heller ikke højde for at PatientCare kan resultere i at patienterne sender flere patientkald end normalt, fordi de bliver mere motiveret for det. Det er dog ikke nødvendigvis et problem mener personalet fra gyn-obs i Randers, fordi selvom antallet af skridt måske ikke i det store hele reduceres, **så kan patienterne få det udbytte at de får mere pleje for samme antal skridt og tid.** Det kan kun en fremsigtet afprøvning i klinisk praksis bevise. Resultater fra analysen viser også at en implementering af en ny version af PatientCare systemet, hvor hver sygeplejerske kun modtager kald fra egne patienter, med stor sandsynlighed vil medføre at personalet ikke vil blive forstyrret af andres patientkald i samme grad som de gør med det nuværende kaldeanlæg.

Resultater fra en spørgeskemaundersøgelse viser at 75 % ud af 80 adspurgte svarede at de gerne ville benytte sig af en app som PatientCare (figur 41).



Figur 41 75% af de adspurgte vil benytte sig af en app som PatientCare

Nogle vurderer at de vil få en bedre patientoplevelse ved at være indlagt, når de får mulighed for at benytte PatientCare i stedet for kaldesnoren, da de opfatter snoren som noget man skal benytte i nødstilfælde. Med PatientCare har patienterne nu adgang til et system hvor de kan spørge efter hjælp på en alternativ måde end kaldesnoren, hvilket for nogle er at foretrække fordi kaldesnoren forbindes med noget akut. Patienterne ved dermed at plejepersonalet er klar over at der er et behov for hjælp og hjælpen kommer ligeså snart plejepersonalet har færdiggjort andre igangværende opgaver.

Implementeringen af PatientCare på gyn-obs vil ikke kræve store organisatoriske ændringer ud over at personalet kommer til at gå rundt med en smartphone i stedet for en nursefinder. Det vil på nuværende tidspunkt ikke erstatte displayet på gangen ved et kald gennem kaldesnoren, fordi kaldesnorsudløste kald ikke er integreret ind i PatientCare. Der vil i en opstartsfasen også være behov for en tovholder og en erfaren person udefra der kan hjælpe personalet i gang med det nye system. Der vil blive effektiviseret på det administrative arbejde i og med at tildeling af patientpleje på Nursefinderen undgås og det bliver kun nødvendigt at registrere ét sted, nemlig i Klinisk Logistik, som de i forvejen har på afdelingen. Derudover vil personalet skulle instruere sine patienter i brugen af den nye applikation og til en hjælp er der lavet en pjece (Appendix 6) henvendt til patienterne, som kan udleveres.

Da selve analysen er afgrænset til at tage udgangspunkt i én afdeling og ikke flere forskellige afdelinger kan man ikke konkludere at PatientCare kan have en positiv indvirkning på hospitalsafdelinger generelt set. Det efterspørges på Regionshospitalet i Randers at man arbejder videre med projektet, da de ser potentiale i det.

Status på prototype

I dette afsnit beskrives resultaterne af accepttesten og layout og funktionalitet for modulet.

PatientApp

Status fra accepttesten viser at PatientApp'en funktionalitetsmæssigt opfylder alle *must have* krav for dette modul fra Kravspecifikationen. Der er dog følgende som ikke er opfyldt:

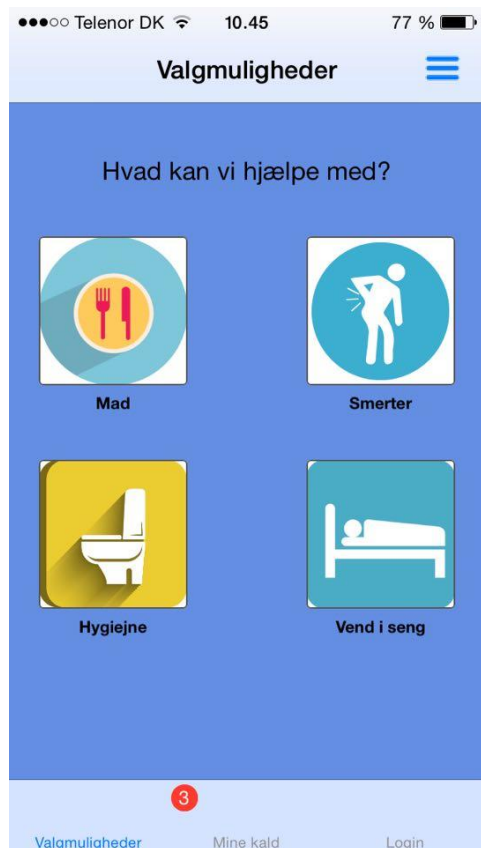
Use case 1.3 - Extension 4.1 Test om patient er indlagt. Kravet handler om at teste en situation hvor en patient er blevet udskrevet efter at være logget ind på PatientApp. Da patientinformationen om patientens indlæggelses- og udskrivelsesdata ikke er implementeret, kan dette ikke testes og derfor er kravet ikke opfyldt. Derudover passer teksten under valgmulighederne ikke altid når app'en bliver opdateret.

Prototypen for PatientApp benytter sig af en pull-funktion, når nye valgmuligheder skal indlæses fra administrationsmodulet. Samme fremgangsmåde benyttes når status på *mine kald* for patienten skal opdateres. Her skal patienten manuelt trække ned på viewet for at opdatere listen og dermed lave et pull-request for at hente ny data. Afprøvning med sygeplejersker i Randers viste at det ikke er god brugervenlighed at patienterne manuelt skal opdatere app'en. Det kan med fordel ske med en service og push teknologi.

Projektgruppen har formået at lave en cross-platform applikation, som patienterne kan installere og bruge på deres egen telefon. Frontenden er dog kun lavet til iOS for prototypen, da konceptet og implementeringen af funktionalitet er prioriteret højere end kvantiteten, for at få afprøvet konceptet med slutbrugerne.

Patienten kan knytte en årsag til patientkaldet i tilfælde hvor patienten har behov for en serviceydelse, som fx et glas vand, hjælp til at komme på toilettet eller lignende. Dette sker ved at patienten vælger mellem de foruddefinerede valgmuligheder på PatientApp, som er bygget op så patienten nemt og effektivt kan sende

et patientkald. Ud over at patienten kan specificere sit patientkald, kan patienten også følge med i hvornår kaldet er blevet oprettet og hvad status er på kaldet - om det er afventede eller udført. Det giver en indikation på om kaldet med succes er blevet sendt afsted. Ligeledes giver det en historik over den pleje patienten har haft behov for under indlæggelsen. På figur 42 og 43 ses screenshots af PatientApp med siden *valgmuligheder* som er en oversigt over de valgmuligheder der er defineret og siden *Mine kald* som er listen med de kald som patienten har oprettet hvor statussen for kaldet er indikeret.



Figur 42 screenshot af siden Valgmuligheder



Figur 43 screenshot af siden Mine kald

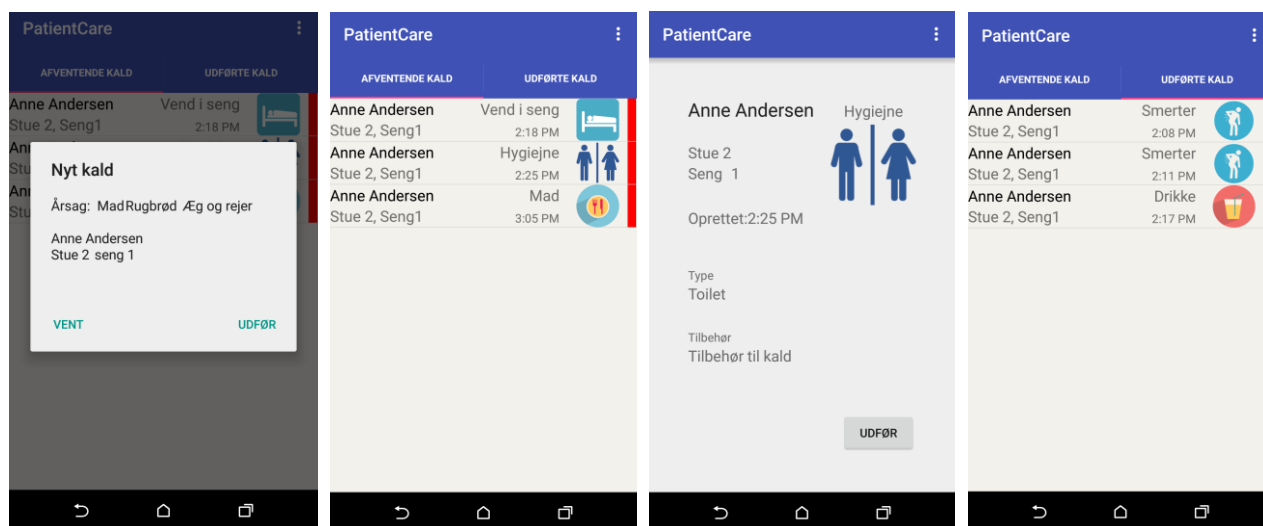
PersonaleApp

Status fra accepttesten viser at PersonaleApp funktionalitetsmæssigt opfylder alle *must have* krav for dette modul fra kravspecifikationen. Use Case 2.6 - Modtag kald, virker ikke optimalt. Dette skyldes enkelte fejl, når der vises en dialogboks ved et nyt kald. Hvis der trykkes *udfør* på dialogboksen modtages samme dialogboks igen og det er ikke meningen, da personalet kun skal notificeres én gang. Derudover stemmer informationen i dialogboksen ikke altid overens med det patientkald der lige blev sendt afsted.

Prototypen af PersonaleApp bruger ligesom PatientApp en pull-funktion til at hente de nye patientkald. Det ville have været mere optimalt at bruge en push-funktion så de nye patientkald bliver pushet ud til PersonaleApp, i stedet for PersonaleApp tjekker om der er kommet nye patientkald.

Projektgruppen har formået at lave en Android app der kan håndtere modtagelsen af patientkald og ændringen af status på patientkaldene efterhånden som patientkaldene bliver udført. Det har været hensigten at patientkaldet som udgangspunkt skulle sendes til den person som er primær plejer på den pågældende patient, for at forstyrre det øvrige personale mindst muligt. Fordelingen af patientkald skal ske i systemets WebAPI. Dette er ikke blevet implementeret i prototypen af PatientCare systemet endnu.

PersonaleApp gør personalet opmærksom på når der er et nyt patientkald ved at telefonen vibrerer og afspiller lyd, samtidig med at der kommer en dialogboks med information om patientkaldet. Når et nyt patientkald modtages kan den pågældende sygeplejerske tage stilling til at udføre kaldet med det samme eller lade det vente. Det er tydeligt hvad årsagen til patientkaldet er og hvor den pågældende patient er indlagt. På den måde bliver personalet gjort opmærksom på årsagen til patientkaldet og kan forberede det efterspurgt til patienten. Når personalet har mange afventende patientkald, kan de ses i en liste med statusfarven rød, der indikerer at de er afventende. Personalet kan vælge at prioritere patientkaldene ved at udføre dem som personalet vurderer er vigtigst først. Applikationen har også en liste over udførte kald, som afdelingen kan bruge til dokumentation af plejeopgaverne.



Figur 44 Screenshots af vigtige views for PersonaleApp

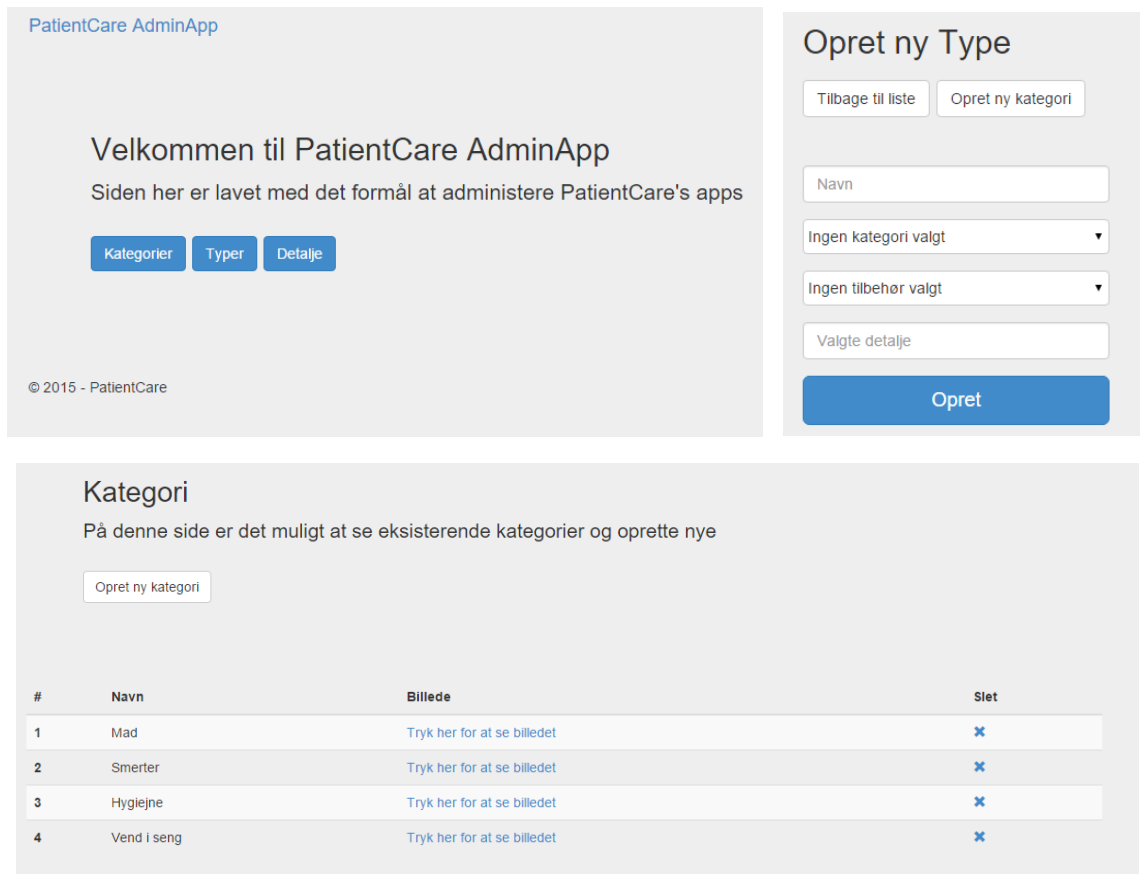
Figur 44 viser fire vigtige views på PersonaleApp, henholdsvis *Nyt kald*, *Afventende kald*, *detaljer om et kald* og *udførte kald*.

AdminApp

Status fra accepttesten viser at AdminApp'en funktionalitetsmæssigt opfylder alle *must have* krav for dette modul fra Kravspecifikationen. Der er dog følgende som ikke er opfyldt:

Use case 3.2 – Opret type, extension 4.1. Dette er ikke godkendt, da det i øjeblikket er muligt at oprette en type uden at tilknytte en kategori. Udover dette er det i øjeblikket ikke muligt at opdele de oprettede valgmuligheder efter afdeling.

Projektgruppen har formået at lave en webapplikation hvor valgmuligheder kan administreres. Der er på app'en mulighed for at se en oversigt over systemets tilpasning af kategorier, typer og detaljer. Administratoren har ligeledes mulighed for at oprette og fjerne systemets tilpasninger. Hver gang en administrator tilpasser systemet, bliver disse ændringer gemt i en fælles database.



Figur 45 Screenshots af tre vigtige views på AdminApp

Figur 45 viser tre vigtige views på AdminApp. Øverst til venstre ses velkomstsiden, hvor man kan tilgå kategorier, typer og detaljer. Nederst ses en oversigt over de kategorier der allerede er oprettet og øverst til højre ses siden hvor man kan oprette en ny type.

WebAPI

Status fra accepttesten viser at WebAPI er udviklet til at tage sig af integrationerne mellem modulerne for PatientCare og har forbindelse til databasen, MongoDB. Modulerne kommunikerer med WebAPI'et via http-requests i JSON-format.

10 Diskussion

I dette afsnit diskuteres hvilke udfordringer projektgruppen har haft og hvilke overvejelser der er gjort igennem projektforsøget.

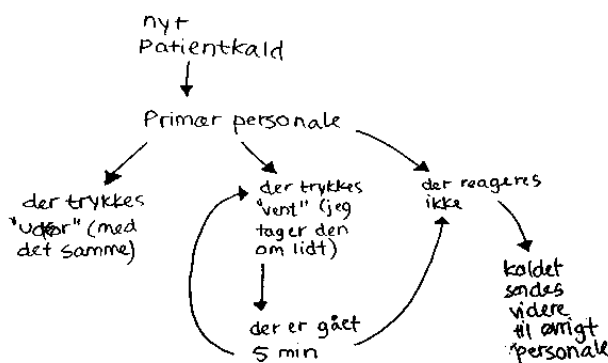
Mini-MTV

Det kan diskuteres i hvor høj grad det er muligt at estimere antallet af skridt og tidsforbrug personalet kan spare med PatientCare, da det afhænger af mange faktorer som er uforudsigelige.

Der er valgt at implementere PersonaleApp på en smartphone men det kan diskuteres om det kunne have været mere hensigtsmæssigt at personalet havde mulighed for at se patientkaldende ved **at kigge ned på et display i stedet for at skulle tage en smartphone op ad lommen for at se hvad patientkaldet drejer sig om.**

Det er især at foretrække i situationer hvor personalet har hænderne fulde, men bør afprøves i praksis med smartphonen før man kan vide om det reelt vil være et problem. Det kan ligeledes diskuteres om det er **etisk korrekt at differentiere mellem tilbud til indlagte patienter.** Med PatientCare er der et unikt tilbud til dem der har en smartphone og kan benytte sig af dem, men der vil også være en brugergruppe som ikke har mulighed for at gøre brug af app'en. Selvom app'en er et supplement til kaldesnoren og ikke en erstatning vil man ikke stille nogle patienter med færre kalderessourcer end de har i dag, vil det stadig stille nogle udenfor. Det er nok et spørgsmål om social ansvarlighed i og med at det rammer dem, der har få ressourcer og herved gør forskel på mennesker ud fra deres placering i samfundet. Det etiske spørgsmål ligger i om alle mennesker er lige meget værd og skal have lige ret og adgang til behandling.

Der er en række udfordringer forbundet med valget om at sende patientkald så de modtages af primær personale. Samtidig med at det øvrige personale kan blive forstyrret mindst muligt, kan det også være risikabelt at de ikke informeres om andet personales patientkald, hvis den primære personale af den ene eller den anden årsag ikke tager sig af patientkaldet. Der er nogle sikkerhedsmæssige foranstaltninger i at et display lyser på gangen i dag, hvor andet personale altid kan se når en patient kalder. Projektgruppen har gjort sig nogle overvejelser (Se figur 46) om hvordan et patientkald efter noget tid automatisk skal sendes videre til det øvrige personale på en afdeling eller evt. en sekundær personale.



Figur 46 Overvejelser om viderestilling af patientkald

Databaseskift

Det blev i første omgang 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 benyttet denne form for database og fordi datastrukturen til applikationerne i PatientCare passer godt til den relationelle datastruktur.

Projektgruppen fik en studielicens til Microsoft Azure som blandt andet tilbyder brugen af MySQL databaser på en databaseserver. Her oprettede gruppen tabeller i overensstemmelse med den datastruktur der var fastlagt for PatientCare. På grund af at et **begrænset antal credits hurtigt blev opbrugt** på Microsoft Azure, var gruppen nødt til at tage en beslutning om at skifte database. Det ville ikke være muligt at lave en ny studielicens til Microsoft Azure, da samme problem ville opstå igen. Ingeniørhøjskolen Aarhus kunne tilbyde at bruge deres database. Da den er afhængig af VPN (Virtual Private Network) blev der kigget efter en løsning hvor der kunne tilgås en database via internettet. Det kan man fx med MongoDB som systemets fælles database blev implementeret på.

Tankegangen med datastrukturen er ikke den samme i MongoDB som for relationelle databaser og det betød at datamodellen for systemet blev ændret. Hvilket havde konsekvenser for blandt andet WebAPI hvor et nyt framework og en ny databasestruktur skulle benyttes. Det har været en udfordring at skifte fra at bruge *Linq to SQL* klasser til at bruge MongoDB frameworket i WebAPI, for at tilgå databasen. Implementeringen for dette var ukendt for gruppen og krævede en del tid at lære.

Databaseskiftet har også haft konsekvenser for AdminApp, der benytter MongoDB på en måde som databasen ikke er bygget til. MongoDB har som nævnt ikke relationer mellem de forskellige collections, men da det er den måde strukturen for sammenkoblingen mellem patientgeneret data og mock data er tænkt på er MongoDB anvendt med relationer mellem data. Det gør at blandt andet en kategori på en type ligger som et id og ikke navnet på kategorien. AdminApp bliver nødt til at hente informationerne fra kategori collectionen, inden de kan vises på siden. **Hvis en kategori bliver slettet, er der dermed ikke et id som hører til det id, som en type har som sin kategori.** Programmet vil derfor fejle, fordi det ikke kan hente den efterspurgte data, eftersom det ikke findes mere. I en relationel database, vil man kunne lave *On Cascade delete* på en *foreign key relation*, men i MongoDB, skal dette gøres manuelt og er ikke blevet implementeret på WebAPI.

Fordelen ved at databasestrukturen i MongoDB ikke har stærke relationer er at der opnås lav kobling mellem data der persisteres. Det betyder at hvis der laves en ændring i databasestrukturen så har det ikke store konsekvenser for resten af det data der persisteres, fordi det er uafhængigt af hinanden i modsætning til i relationelle databaser.

Android Studio og GitHub

Android Studio har været et nyt udviklingsværktøj for ST'erne, som har haft ansvaret for udviklingen af PersonaleApp. De har sideløbende med projektforløbet fulgt undervisningen i et kursus der udbydes af Ingeniørhøjskolen hvor læringsmålene er at programmere en app i Android Studio og på den måde tilegnet sig viden om blandt andet en app's anatomi i Android.

Det har været en udfordring for udviklerne at versionsstyre PersonaleApp i GitHub gennem Android Studio, fordi der ikke er nogen i projektgruppen der har haft kendskab eller benyttet sig af GitHub før projektets start. Der har især været problemer med at merge arbejdet sammen når udviklingen af app'en er foregået samtidig flere steder. I disse tilfælde har projektgruppen arbejdet med branches der kan merges sammen.

JSON objekter vs. tekststreng

Når alle afventende patientkald hentes fra databasen, bliver de leveret som en JSON formateret tekststreng, hvilket er svære at omdanne til et objekt i koden. I stedet ville en bedre løsning være at benytte sig af JSON objekter, fordi det gøre det nemmere at omdanne.

Xamarin

Ud fra kravspecifikationen skal PatientApp udvikles i et miljø der gør det muligt at understøtte de tre største smartphone styresystemer iOS, Android og Windows, da PatientApp er tiltænkt til at være en *Bring your own device* løsning. Den traditionelle måde at løse det på, er at udvikle en applikation for sin platform i hvert sit sprog, henholdsvis *objective-c* eller *swift*, *Java* og *C#*. Dette vil opfylde behovet for at udvikle PatientApp til de tre platforme, men kræver at man kan udvikle i de tre sprog. Designmæssigt skal der designes en backend og en UI for hver platform, hvilket vil tage tre gange så lang tid at udvikle.

En af de udfordringer der har været med at udvikle PatientApp som en cross-platform løsning har været tilgangen til Xamarin platformen og opsætte et cross-platform projekt med Visual Studio. Det har været en udfordring, fordi det er **ny teknologi, og det kræver meget indsigt i forskellige styresystemer og deres API'er på trods af, at applikationerne programmeres i C#**. Det har især været en udfordring at få bygget og deployet et Xamarin.iOS projekt i Visual Studio. Da Visual Studio er en IDE, der som udgangspunkt kun kan køres på Windows, kræver det en virtuel maskine på en Mac OSX computer, hvor en Mac Build OS server på Windows skal kunne snakke med en Mac Build OS server på Mac OSX. **Kan man ikke kobles på build serveren i Visual Studio, kan projektet ikke bygges**. Og det har skabt udfordringer, fordi versionen af denne server skal passe i begge styresystemer – både i den virtuelle og ikke-virtuelle. Da Xcode midt i forløbet kom med en ny version, skulle build serveren opdateres, hvilket gav nogle problemer i forhold til at holde den synkroniseret med den rigtige version på Mac OSX og Windows. **En anden udfordring har været at bygge PCL'en i hvert projekt. Sker der en ændring i PCL, skal den bygges for hvert projekt, hvor den er inkluderet.**

Xamarin.Forms

I forbindelse med teknologiundersøgelsen stødte projektgruppen ind i nogle bedre tekniske alternativer efter et seminar hos Mjølner Informatics, der afholdte et oplæg omkring Xamarin. Her blev der snakket om to Xamarin løsninger. Den ene løsning var den traditionelle løsning, som PatientCare har valgt, og en anden løsning som gør brug af Xamarin.Forms. På dette tidspunkt var projektgruppen allerede godt i gang med implementeringen af den traditionelle løsning men Xamarin.Forms ville have været **mere effektiv og ville have været hurtigere for proof of concept af PatientCare**. I stedet for at hvert view og dens kontroller skal programmeres for hver platform, vil man med Xamarin.Forms kun skulle gøre dette én gang. Fx vil et tekstfelt blive til et UITextView på iOS, EditText på Android og TextBox på Windows.

Xamarin.Forms bygger også på princippet om at have en C#, fælles kodebase som den traditionelle Xamarin løsning. **Med Forms deles brugergrænsefladen også og dermed deles koden 100 %.**

Ulempen ved Xamarin.Forms er at **brugergrænsefladen skal skrives direkte i C# eller i XAML**, da det ikke er muligt at bruge drag-and-drop funktionalitet som med den traditionelle løsning, eftersom brugergrænsefladen er forskellig. Det kan være besværligt hvis man ikke kender teknologien. Derudover er **custom rendering svært og bør undgås i Xamarin.Forms**, da man ellers ligeså godt kan bruge den traditionelle Xamarin løsning. Design af layoutet er også svært, da hver layout er forskellig fra hver platform, fx kan font størrelsen være forskellige selvom samme størrelse er sat for dem alle. Derudover skal man kende til de forskellige platformes API'er på trods af at alt skrives i C# (13).

For PatientCare systemet har den traditionelle Xamarin været en god løsning. Men havde systemet benyttet sig af Xamarin.Forms vil det udviklings-og tidsmæssigt havde været en bedre løsning for *proof of con-*

cept sammenlignet med den nuværende løsning. Prototypen af PatientApp ville dermed også have fået en Android og Windows løsning. Det ville have betydet, at PatientCare ville have levet op til konceptet om *bring your own device*, som var det mål, der blev sat fra start.

Hvorfor er et system som dette ikke lavet før?

Man kan spørge sig selv hvorfor der ikke er lavet et system hvor patienten kan knytte en årsag til sit patientkald førhen, når teknologierne findes til det. Bachelorteamet har diskuteret at der er nogle udfordringer med at implementere et sådant system på offentlige sygehuse og det kan være en langsommelig proces blandt andet på grund af sikkerhedsmæssige foranstaltninger og standarder.

Den hurtige fremdrift i udviklingen af ny teknologi vil gøre at man kommer til at se mere og mere teknologi på hospitalerne end hidtil fordi mulighederne bliver flere. For blot fem år siden var der ret mange der havde en smartphone. I 2015 har 77 procent af den danske befolkning en smartphone og med tiden vil tallet højst sandsynligt stige mere (14).

Førhen har der været forbud mod at have mobiler på hospitalet fordi det kan give forstyrrelser i hospitalsudstyr (15). Dette forbud er i dag ikke gældende mere og man er begyndt at optimere på internetforbindelsen blandt andet på Aarhus Universitetshospital hvor der bliver installeret distribuerede antennesystemer som nedsætter de mobile enheders elektromagnetiske udstråling (16).

11 Konklusion

Formålet med dette bachelorprojekt var både, at udvikle en prototype af et system, der gør det muligt at knytte en årsag til patientkald men også at undersøge behovet for et sådant system.

Sandsynligheden for at der er behov for et system der kan knytte en årsag til patientkald er stor på **Gynækologisk – Obstetrisk Afdeling på Regionshospitalet i Randers**. Det viser resultater fra Mini-MTV-analysen.

Projektgruppen har med udgangspunkt i **møder med slutbrugerne** udviklet en prototype for PatientCare, som viser de væsentligste funktionaliteter for at knytte en årsag til et patientkald. De væsentlige funktioner består i at patienten vælger ud fra nogle valgmuligheder hvad patienten har brug for og denne valgmulighed bliver knyttet til patientkaldet personalet modtager.

Prototypen består af tre moduler som opfylder alle **must have kravene**. For **PatientApp** var målet at udvikle en smartphone app som gør det nemt for en patient at oprette et patientkald med tilknyttet årsag. Ligeledes var det vigtigt at denne app, kunne installeres på en telefon med iOS, Android eller Windows som operativ system. Dette er opnået ved at udvikle PatientApp i Xamarin platformen, dog er app'ens frontend på nuværende tidspunkt kun fungerende på **iOS**.

For **PersonaleApp** var målet at udvikle en smartphone app, hvor personalet kunne **håndtere og skabe overblik over patientkald med tilknyttet årsag**. Dette er opnået ved at udvikle en smartphone app, i Android Studio som kan installeres på personalets arbejdstelefon.

For **AdminApp** var målet at udvikle et administrationsmodul hvor administrator af **systemet kan tilføje, fjerne og slette valgmuligheder patienten kan vælge imellem**. Dette er opnået ved at udvikle en webapplikation der er **hostet på Azure**. Disse tre moduler **interagerer med hinanden ved brug af et WebAPI**, som er udviklet i forbindelse med dette projekt.

Projektgruppen startede i fællesskab med at specificere **prototypens krav**. Senere i forløbet delte gruppen sig op for hver især at udvikle modulerne. Modulerne blev løbende **testet isoleret set** fra resten af systemet og sammen med resten af modulerne, når kommunikationen mellem dem skulle testes. Projektgruppens medlemmer har udnyttet deres forskellige kompetencer, heriblandt kendskab til webudvikling og mobilløsninger. **ST'erne har blandt andet gjort sig erfaringer med ny teknologi under udviklingen af PersonaleApp i Android Studio og styrket evnerne til at kunne viderekommunikere slutbrugernes behov til IKT'erne**. Det sidstnævnte har været en mulighed fordi projektteamet repræsenterer to forskellige ingeniøruddannelser. **IKT'erne har erfaret at der er et vigtigt aspekt i at tilgodese slutbrugernes behov for at et system kan fungere i praksis, især indenfor hospitalsverdenen**.

Projektet har lagt op til mange problemstillinger og projektgruppens medlemmer har været udfordret både i analysen af behovet og udviklingen af prototypen. Der har været mange overvejelser igennem processen og beslutninger der skulle tages undervejs. Det har været nødvendigt at afgrænse projektet en del, da der har været mange inputs fra eksterne fagpersoner om hvilken retning projektet kunne tage.

12 Perspektivering

En videreudvikling af PatientCare vil bygge på krav fra MoSCoW-prioriteringen i Kravspecifikationen, hvor *should have* vil blive prioriteret som det første og derefter *could have*. Disse krav vil lede til en række nye use cases.

Fordeling af patientkald til primær personale

Det næste der skal implementeres indebærer blandt andet at fordele patientkaldene ud til primær personale, så alt personale ikke modtager alle kald. Når et patientkald bliver sendt ind i systemet, skal der kigges på hvilken personale der er primær på patienten og kaldet skal så sendes til vedkommende. Dette skal ske på WebAPI.

Validering af patient

På nuværende tidspunkt validerer systemet patienten ved at sammenligne CPR-nummer fra PatientApp med de CPR-numre der er lagret i fællesdatabasen. Hvis det stemmer overens er det udtryk for at patienten er indlagt. Fremsigtet er det meningen at der skal laves en bedre validering af om patienten er indlagt eller ej. Det er ønskværdigt at patientens indlæggelsestidspunkt benyttes af sikkerhedsmæssige årsager for at tjekke om patienten er indlagt og på den måde sikre at kun indlagte patienter kan sende patientkald afsted. Dette kan foregå via WebAPI ved at tjekke i fælles databasen ud fra data mock om patienten er indlagt.

Lokalisering af patient

En af de mere væsentlige overvejelser med hensyn til en udvidelse er lokalisering af patient. Som nævnt i afgrænsningsafsnittet er det et problem, at patienten har mulighed for at oprette et patientkald uafhængig af, hvor patienten befinder sig. Det er især et problem hvis patienten ikke er indlagt på et sengeafsnit. *Columna Service Logistics* benytter bl.a. en teknologi til at spore en sengs flow i systemet, hvilket er til gavn for portører, der skal transportere senge. Denne teknologi er RFID og lignende løsning vil kunne tages i brug til PatientCare til at tracke patienter og dermed sikre at patienten ikke kan foretage kald alle steder fra. Alternativt kunne være at benytte GPS og/eller Wi-Fi til at bestemme patientens placering på hospitalet.

Gør valgmuligheder afdelingsspecifikke

Det skal være muligt at gøre valgmuligheder afdelingsspecifikke i AdminApp for at PatientCare kan bruges på andre afdelinger end på gyn-obs.

Validering af personalets loginoplysninger

Der skal implementeres en bedre validering af personalet ved at tjekke på password og brugernavn i databasen.

Integrationer

Derudover er en integration til *Cetreas kliniske logistik* også prioriteret som noget af det næste, så mock data kan blive til reelle data. I could have er integrationen til *Systematic's opgavesystem* prioriteret. Samarbejdet og integrationen med dem ville betyde, at PatientCare ville kunne indgå som en del af den samlede service logistik-løsning. Ud over integrationer til service-logistiske og kliniske systemer vil en tredje integration, der vedrører kaldesnoren være endnu en god integration. Hvis der integreres til et eksisterende system som kaldesnoren benytter, vil der ikke længere være to systemer men ét hvor alle kald vil være at se.

Pull/push teknologi

Generelt kan der optimeres på **performance og brugervenlighed** ved at anvende en anden teknologi end den der på nuværende tidspunkt er implementeret i prototypen hvor der benyttes pull teknologi. Der tjekkes hele tiden på, om der er kommet et nyt patientkald, for at holde data opdateret på PersonaleApp. Performancemæssigt vil det være en bedre løsning at gøre brug af push, fordi man i forhold til pull ikke bruger **CPU ressourcer** på at stå og trække data for at holde det opdateret, i tilfælde hvor der ikke er kommet ny data. En implementering af push teknologien skulle **ske i WebAPI'et** som skulle sørge for at **sende patientkaldet videre til PersonaleApp**. Push teknologien kunne med fordel også benyttes når der på PersonaleApp er udført et nyt kald og det skal **tilbage igennem WebAPI'et og ud på PatientApp**, så statusændringen vises for patienten. Det kan løses ved at implementere **SignalR** teknologien. Det vil gøre systemet i stand til lave push beskeder til PatientApp og PersonaleApp.

Sikkerhed på WebAPI

WebAPI har **ingen form** for sikkerhed. Dette bør være næste skridt for WebAPI'et, da det ikke vil være acceptabelt at have et WebAPI som er tilgængeligt for alle og specielt ikke hvis PatientCare skal implementeres på en afdeling, hvor der er **personfølsomme data**.

Log ind på AdminApp

Det skal ikke være muligt for enhver at bruge AdminApp og derved **kunne slette eller oprette nye valgmuligheder**. Det skal kun være administratorer af PatientCare systemet der kan tilpasse systemet. Men på nuværende tidspunkt er der ikke udviklet sikkerhedsmæssige anordninger for dette.

13 Referencer

1. Statistik, Damarks. Fakta om sundhedsvæsenet – sundhedsvæsenet i tal. *besøgt d. 15/11-15*. [Online] 2015.
<http://www.regioner.dk/aktuelt/temaer/fakta+om+regionernes+effektivitet+og+%C3%B8konomi/kopi+af+fakta+om+sundhedsv%C3%A6senet>.
2. Afbrydelser i klinisk sygeplejepraksis. *Besøgt d. 2/12-15*. [Online]
https://www.idunn.no/klinisk_sygepleje/2010/01/afbrydelser_i_klinisk_sygeplejepraksis.
3. Sygehusbenyttelse . *besøgt d. 27/11-15*. [Online] 2014.
<http://www.dst.dk/da/Statistik/emner/sundhed/sygehusbenyttelse.aspx>.
4. Innovativ sporbarheds-it på vej til danske hospitaler. *Besøgt d. 12/5-15*. [Online] http://www.rm.dk/om-os/aktuelt/nyheder/News_2013/09-september/innovativ-sporbarheds-it-pa-vej-til-danske-hospitaler/.
5. Appendix 3 - Behovsanalyse.
6. Sygeplejersken - kontaktsygepleje - stærk, men sårbar. *Besøgt d. 2/10-15*. [Online]
http://www.dsr.dk/Sygeplejersken/Sider/SY-2007-04-35-1-Faglig_information.aspx.
7. Appendix 5 - Observation i Randers.
8. Opgavesystem. [Online] [Citeret: 10. 10 2015.] <https://da.systematic.com/healthcare/products/columna-service-logistics/apps/task-management/>.
9. Smartphone OS Market Share, 2015 Q2. [Online] [Citeret: 11. 12 2012.]
<http://www.idc.com/prodserv/smartphone-os-market-share.jsp>.
10. API. [Online] http://blogs.msdn.com/cfs-filessystemfile.ashx/___key/communityserver-blogs-components-weblogfiles/00-00-00-56-73/2318.WithAPIArchitecture.PNG.
11. Simple, powerful, cross-platform SQLite client and ORM - Updated version with PCL support. [Online] [Citeret: 25. 11 2015.] <https://github.com/oysteinkrog/SQLite.Net-PCL>.
12. Appendix 7 - Afprøvning af PatientCare. 1/2-15.
13. Working with a Local Database. [Online] [Citeret: 10. 12 2015.]
<https://developer.xamarin.com/guides/cross-platform/xamarin-forms/working-with/databases/>.
14. Elektronik i hjemmet. [Online] [Citeret: 11. 12 2012.]
<https://www.dst.dk/da/Statistik/emner/forbrug/elektronik-i-hjemmet>.
15. Philipsen, Anne Mette. Mobiler skyld i alvorlige fejl på hospitaler. [Online] [Citeret: 10. 12 2015.]
<http://www.dr.dk/sundhed/Sygdom/Artikler/2008/20080825113625.htm>.
16. Ingen huller i mobilnettet på DNU. [Online] [Citeret: 23. 10 2015.]
<http://www.dnu.rm.dk/presse/pressemeddelelser/2014/ingen-huller-i-mobilnettet-pa-dnu/>.