ProjektRapport

PatientCare – Patientkald med tilknyttet årsag

# Abstract

# Resume

Projekts formål

Følgende rapport beskriver vores arbejde med at udvikle PatientCare. Formålet er at udvikle en løsning, der effektiviserer arbejdsgangen for daglige personale på et sygehus. Effektiviseringen består i at supportere kaldesnoren på en stue i en afdeling, hvor en patient er indlagt. Patienten tilknytter en årsag til at kalde på personalet. Ved at personalet kender til årsagen til at der er behov for hjælp, kan personalet udføre den opgave, som hjælpen kræver på vej til patienten. Det medfører bl.a., at personalet sparer skridt ved at der på forhånd vides, hvad patienten har behov for.

Problemstilling

Projektet ligger op til mange funktioner, der skal være med i et produkt, der kan implementeres i klinisk praksis. De problemstillinger der er opstået undervejs i processen er fx at patientens mobil skal kunne lokaliseres. Det vil være et problem, hvis patienten kan sende et kald uafhængig af hvor patienten befinder sig, og PatientCare vil derfor være et upålideligt system, da personalet ikke vil kunne finde patienten. Og det vil resulterer i at projektets formål om at spare skridt ikke kan opfyldes. Men det er ikke blevet muligt at inddrage dette i slutproduktet, og det har dermed være nødvendigt at afgrænse integrationer, funktionaliteter og afdelinger.

Valgte løsninger

Afgrænsingen har givet en prototype, der viser kernen i projektet og underbygger de grundlæggende krav, som der er fundet frem til ud fra MTV-analyse og diverse behovsanalyser. Prototypen viser konceptet af PatientCare og som er prioriteret højst i projektet frem for performance og kvalitet. Den valgte løsning er endt ud i tre applikationer, som tilsammen skal demonstrere idéløsningen til PatientCare.

Anvendte metoder

I projektforløbet er der blevet gjort stor brug af iterativ udviklingsmetoder og brugerdreven processer, hvor de sundhedsprofessionelle har været involveret igennem store dele af forløbet.

Væstentlige resultater

De væsentlige resultater er endt ud i en mobilapplikationen, som installeres på en patients mobil enhed – uanset styresystem men som udgangspunkt for prototypen som en iOS applikation. Patienten har mulighed for at knytte en årsag til et kald ved at vælge en række valgmuligheder. Disse valgmuligheder kan konfigureres og opsættes i en webapplikation, så det passer til den pågældende afdeling. Ydermere har personalet en mobil enhed med Android som styrestystem, hvor det er muligt at modtage patientkaldet og forholde sig til det. Disse tre applikationer har tilsammen givet en prototype for PatientCare, der har værdi for brugeroplevelsen på sygehuset men også en betydelig bedre arbejdsdag for plejepersonalet.

Indholdsfortegnelse

1 Abstract 1

2 Resume 1

3 Forord 5

4 Indledning 6

Formål med projektrapport 6

Baggrund for projektet 6

Projektbeskrivelse 7

Vision 8

Forkortelser og definitioner 8

5 Afgrænsning 10

Afgrænsning af funktioner 10

Afgrænsning af afdeling 11

Afgrænsning af integrationer 12

Integration med Systematic 13

6 Systembeskrivelse 14

PatientApp 15

PersonaleApp 15

AdminApp 15

WebAPI 15

Aktørbeskrivelse 16

7 Krav 16

8 Systemarkitektur 19

Kommunikation mellem moduler 19

Data 19

Lokal data persistering 20

Trelagsarkitektur 20

9 Design 21

PatientApp 22

Cross platform 22

PCL og SQLite 23

Sekvensdiagrammer (Opret kald) for PCL 24

PersonaleApp 25

SQLiteDatabase 25

Sekvensdiagram (Udfør kald) 26

Bound Service (Modtag kald) 27

AdminApp 29

WebAPI 30

Database 30

Data mock 31

10 Resultater 32

Mini-MTV 32

Layout og funktionalitet 33

PatientApp 33

PersonaleApp 34

AdminApp 36

Status på prototype 36

PatientApp 37

PersonaleApp 37

AdminApp 38

WebAPI 38

11 Materialer og metoder 39

Udviklingsproces 39

Udviklingsdokumentation 40

Tidsplan 40

**Agil systemudvikling** 41

Mini-MTV 42

Projektstyring 43

Versionsstyring 43

Udviklingsværktøjer 44

Ansvarsområde 44

Faciliteter 45

Møder og eksternt samarbejde 45

Samarbejde med Systematic 45

Samarbejde med sundhedsfaglige 45

12 Diskussion 45

Overvejelser 46

Projektforløb 46

Skift af database 46

PatientApp – uden Xamarin 46

Valg af Xamarin 47

Android 48

Databaseskift 49

Sikring af stabilt netværk 52

Systemsikkerhed 52

WebAPI 52

AdminApp 52

Udfordringer 53

Xamarin 53

Github 54

Databaseskift 54

Lokalisering af patient 54

Patientens rettigheder 55

WebAPI 55

AdminApp 55

13 Konklusion 56

14 Perspektivering 56

15 Referencer 56

# Forord

# Indledning

I Danmark har vi 54 offentlige sygehuse og der arbejder rundt regnet 100.000 fuldtidsansatte på de offentlige sygehuse (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 648.415 (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 på en tryghed og en sikkerhed patienten har under indlæggelse. Når patienten trækker i snoren udløser det en alarm til personalet på afdelingen som skal tage 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 unødvendige 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. Det kan hurtigt blive til mange skridt i løbet af en dag, hvilket der med dette bachelorprojekt forsøges at gøre noget ved.

## Formål med projektrapport

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 for projektet

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, kan der spares tusinder af spildte 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 sommeren 2015. Det lød meget interessant og oplagt at det skulle foregå i Aarhus. Der var potentiale for at indgå i et projekt, hvor kompetencerne fra både IKT-ingeniørstuderende og sundhedsteknologistuderende kunne udnyttes. Gruppemedlemmet tog kontakt til de tre andre, 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 Columna Service Logstics som 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 omkring sporbarhed allerede var et pilotprojekt i implementering, hvorfor det ville være mere relevant for os at arbejde med et nyt projekt der krævede både 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 tilkalder personalet. Denne case valgte projektgruppen at arbejde videre med. Opgavesystemet fra Columna Service Logistics blev nævnt i forbindelse med casen. Opgavesystemet 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 Systematics opgavesystem skulle kunne håndtere opgaver rettet mod sygeplejerskerne også, 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 Opgavesystemet, hvorigennem 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.

Ideen med at sende patientkald ind i Opgavesystemet 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 for 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 Opgavesystemet til fremtidig håndtering af de plejeopgaver der følger med et patientkald. Det gav anledning til at projektgruppen kunne udforme en projektbeskrivelse og gå i gang med.

## 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 kald når patienten har behov for en serviceydelse, som ikke er akut. Det giver personalet mulighed for at medbringe det patienten forespørger og dermed sparre tid og skridt.

For at løse dette er der udviklet tre forskellige applikationer kaldet PatientApp, PersonaleApp og AdminApp som defineres som moduler til systemet.

* 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 ved at vælge mellem 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 hun vurderer at noget andet skal færdiggøres først. Personalet har hele tiden mulighed for at se afventede kald og en historik over de kald der er udført.
* AdminApp er en webapplikation som gør det muligt at tilpasse nogle foruddefinerede valgmuligheder til forskellige afdelinger f.eks. muligheden for hjælp til amning på fødeafdelingen, hvilket ikke vil give mening på akutafdelingen.

Modulerne har behov for at kommunikere med hinanden, fordi de skal udveksle data, da personalet har behov for at vide, når en patient har sendt et kald. Til dette er der udviklet et WebAPI der står for integrationen mellem de tre moduler. Modulerne vil derfor kommunikerer med hinanden gennem dette WebAPI.

Der benyttes en fælles database som 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 dem. 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.

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

|  |  |
| --- | --- |
| Værdier | |
| Patient | **Plejepersonale** |
| * 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 | * Mere informativt 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 * Effektiviserer arbejdsgangen * Fordeling af plejeopgaver * Færre forstyrrelser fra andres patienter * Bedre arbejdsmiljø * Dokumentering af plejeopgaver |

## Forkortelser og definitioner

***PatientCare*** Hele systemet

***PatientApp*** Den applikation patienter downloader på deres smartphone under indlæggelsen hvorfra de kan oprette patientkald

***PersonaleApp*** Den applikation som personalet håndterer patientkald fra på deres arbejdstelefon (smartphone)

***AdminApp***  Den applikation hvorfra de foruddefinerede valgmuligheder administreres

***WebAPI*** WebAPI er en softwaregrænseflade, der tillader software at interagere med andet software over internettet.

***Patientkald*** En forespørgsel på en ydelse som patienten sender afsted til PersonaleApp via PatientApp når patienten har behov

***Valgmuligheder*** Er de muligheder patienten har at vælge imellem når han/hun vil sende et Patientkald. Disse defineres i AdminApp, så de kan tilpasses den enkelte afdeling.

***Kategori*** Den højeste rangerede valgmulighed. En kategori kan godt eksistere uden en type og en detalje. Eksempel på kategori: Drikke

***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: Kaffe

***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: sukker

***Afventende kald*** De kald som er modtaget på PersonaleApp før de bliver udført af personale

# 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. Derefter afgrænses projektet til at tage udgangspunkt i en afdeling med en bestemt organisationsstruktur, da afdelinger er meget forskellige og derfor har forskellige måder at udføre plejeopgaver omkring patienterne på. Til sidst afgrænses de integrationer som projektgruppen finder relevante for PatientCare systemet at integrere til for at udveksle data.

## Afgrænsning af funktioner

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 (Se Bilag Behovsanalyse). 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 X 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.



”Send patientkald til primær personale” ligger på grænsen mellem kernen og periferien fordi det er en funktion som er det næstvigtigste for slutbrugerne, så 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 patienten 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.

## Afgrænsning af 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 systemet 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 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 (5).

**Patienter**Patienters forudsætninger for at kunne bruge en app afhænger blandt andet af deres sygdom og alder.   
På Gyn-obs i Randers 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 (6):

* *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. PatienCare er derfor ikke en erstatning af kaldesnoren men et supplement.

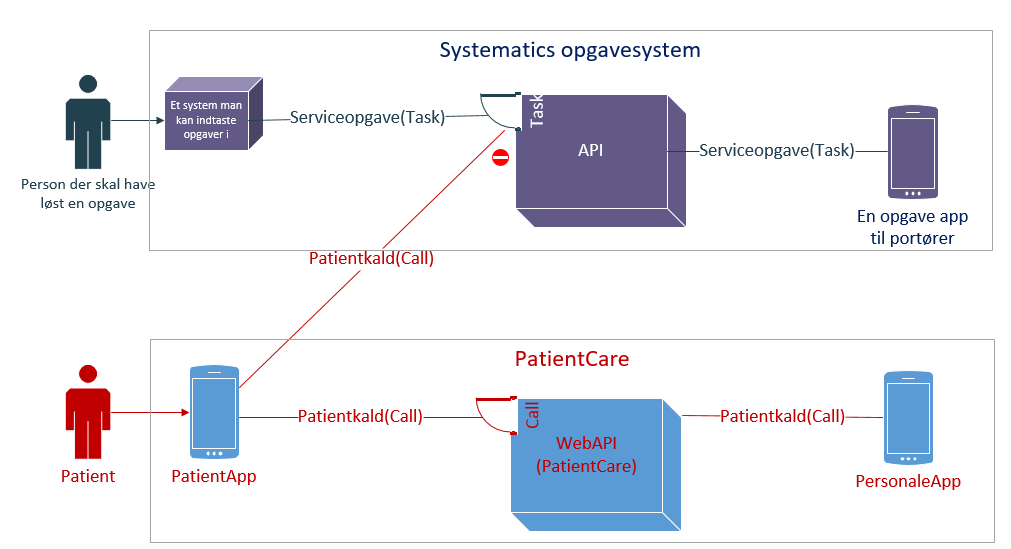
## Afgrænsning af integrationer

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

**Systematics Opgavesystem**  
Systematics Opgavesystem 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[[1]](#footnote-1). 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.

Opgavesystemet 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 opgavesystemet. Fx skal man i et portørkald skrive på hvor mange personer der skal til for at fuldføre en opgave, hvilket ikke passer med sygeplejerskernes arbejdsgang i forhold til patientpleje. I stedet for at forsøge at tilpasse patientkald til et system der håndterer portøropgaver og dermed forsøge at gøre portøropgaver på tværs af afdelinger til plejeopgaver på afdelingsniveau valgte bachelorgruppen i overensstemmelse med Systematic at lave en ny løsning. Hvis systemerne skal kunne snakke sammen skal der derfor foretages en udvidelse af opgavesystemet. Projektgruppen har derfor ikke haft mulighed for at integrere til opgavesystemet, men udvekslingen af data mellem PatientApp og PersonaleApp gennem WebAPI’et i PatientCare er løst med samme teknologi som det kræves at kommunikere til grænsefladen for Systematics Opgavesystemet. Dette foregår med http-protokollen. Der er derfor taget højde for en fremtidig integration.

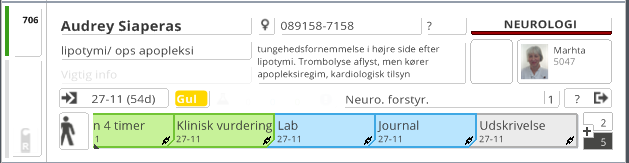
### Integration med Systematic



Figur X illustrerer at Systematics opgavesystem ikke kan modtage patientkald, men kun portøropgaver.

**Cetreas kliniske 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 registreres 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 kliniske logistik hver dag og det er derfor oplagt at kunne trække data herfra.   
Da det ikke har været muligt at samarbejde med Cetrea, har projektgruppen opsat et data mock for hver patient som skal illustrere et sådant system.

På figur 5 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 Patientkompontent Cetreas klinisk logistik

# Systembeskrivelse

På figur X 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.



### PatientApp

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

### PersonaleApp

Prototypen af PersonaleApp er udviklet til Android som ifølge statestikker fra markedsandelen for smartphone OS er det mest udbredte smartphonestyresystem[[2]](#footnote-2). Og til forskel fra PatientApp, der understøtter tre platforme, skal PersonaleApp understøtte én platform, og da Android er den mest udbredte, er det derfor denne der er blevet valgt som styresystem.

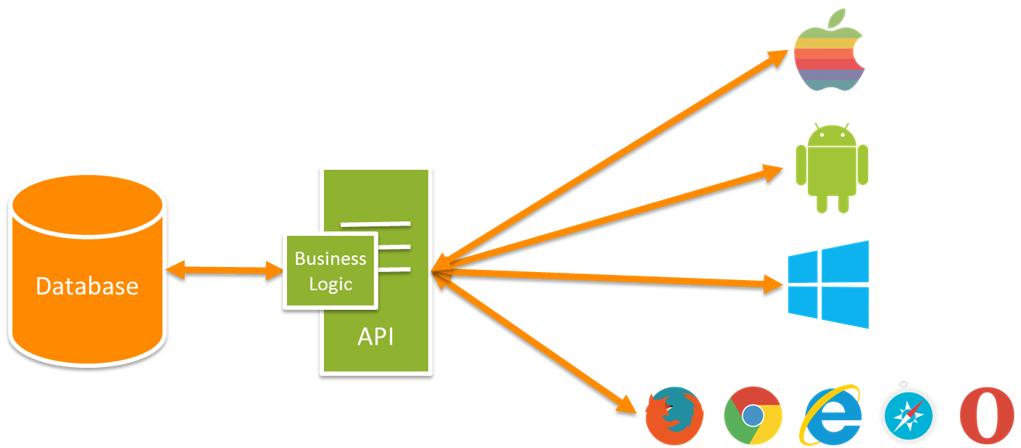
### AdminApp

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

### WebAPI

Hele PatientCare systemet 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 systemet (PatientApp, PersonaleApp og AdminApp). Kommunikationen foregår ved at data hentes fra eller leveres til en webside som WebAPI’et udstiller.

Figur X viser princrippet i et WebAPI, som projektgruppen har udnyttet.



Figur 11 http://blogs.msdn.com/cfs-filesystemfile.ashx/\_\_key/communityserver-blogs-components-weblogfiles/00-00-00-56-73/2318.WithAPIArchitecture.PNG

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. Fx hvis der skal implementeres en ny funktionalitet gør brugen af et WebAPI det simpelt. Det kan fx være med udgangspunkt i et should have krav i MoSCoW-prioriteringen:

SH1-5: som er et krav om at det skal sikres at patientkaldet kun kan foretages fra afdelingen hvor patienten er indlagt og altså ikke udenfor sygehusets område. Implementeringen af dette vil ske dels i PatientApp og dels i WebAPI’et.

Den måde WebAPI’et er struktureret med den fælles database på gør det nemt at benytte WebAPI’et til at håndtere valideringen af om patientkaldet sendes indenfor hospitalets rammer. Mens den fejlbesked der skal gives til patienten, hvis patienten ikke befinder sig på afdelingen, når kaldet oprettes skal implementeres i PatientApp’en.

## Aktørbeskrivelse

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



**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 den mobile applikation 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 mobilapplikationen 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.

# Krav

For at specificere kravene der skal være grundlag for udviklingen af prototypen af 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.

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, PersonaleApp og AdminApp som ses i figur X, X og X. Ud fra disse er der lavet fully dressed use cases med henblik på at kunne mappe kravene over til implementeringsdiagrammer over funktionerne i de enkelte moduler.







# 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 X ses det at WebAPI’et er bindeled mellem modulerne. WebAPI’et 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 Systemetic’s Opgavesysem foregår på.



## Data

Figuren viser at WebAPI’et har forbindelse til en fællesdatabase. Der er behov for at kunne persistere data i en fælles database for at alle moduler kan tilgå det. Det data der skal persisteres i fælles databasen består af data fra to forskellige steder; data som PatientCare systemet selv genererer i PatientApp (se Parametre PatientApp figur X) og data mock af data fra Cetreas kliniske logistik som PatientCare ønsker at gøre brug af (Datamock figur X).



Figur 1 viser en oversigt over hvor de forskellige parametre kommer fra.

Figuren viser også at PersonaleApp’en skal modtage det samlede data (Se Parametre PersonaleApp figur X).

Projektgruppens medlemmer er blevet 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 de har 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. Denne datamodel ligger til baggrund for de modeller WebAPI’et modtager og returnerer.

## Lokal data persistering

Udover 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.

## Trelagsarkitektur

Arkitekturen i systemets moduler er lavet ud fra 3-lagsmodel 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 og overskue programmet når kode skal implementeres, da de tre lag så vidt muligt holdes adskilt.

*Presentation layer* er det lag brugeren interagerer og håndterer modtagelse og præsentation af data.  
*Business logic layer* er det lag der håndterer udvekslingen af data mellem *presentation layer* og *data acces layer*. Når brugeren interagerer med App’en kaldes der funktioner i *business logic layer*.   
*Data acces layer* håndterer udvekslingen af data til det sted data lagres. 3-lags arkitekturen for hvert modul ses på figur X.



# Design

Som følge af systemarkitekturen er næste fase i udviklingsprocessen design af PatientCare. Hvert modul beskrives omkring hvilke designløsninger der er overvejet, samt de valg der er blevet taget og begrundelsen herfor. Figur X beskriver hvordan flowet af data ser ud er for systemets moduler.



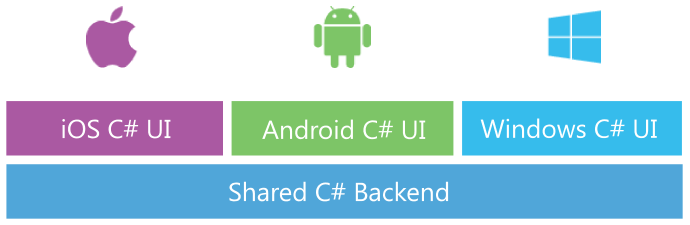
Det ses at hvert modul kommunikerer med hinanden via WebAPI’et, som fungerer som systemets grænseflade ud til modulerne. Fx når der i PatientApp oprettes et kald, sendes det til WebAPI’et, der skubber kaldet videre til fællesdatabasen. Det oprettede kald vil ligge i fællesdatabasen, som WebAPI’et tilgår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

## PatientApp

PatientApp er udviklet i Xamarin til henholdsvis iOS, Android eller Windows. For prototypen er der lagt vægt på implementering af iOS for at kunne fremvise konceptet.

### Cross platform

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



PatientApp er delt op efter trelagsarkitekturen som beskrevet under systemarkitektur. Det tre lag består af *Presentation Layer, Business Logic Layer* og *Data Acces Layer*. *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 et *Portable Class Library*. Hver platform har deres egen lokale database hvor databaseopbygningen og implementeringen er den samme. Strukturen for designet er den samme og ligger i *Data Acces Layer*.

**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, fx en wrapper 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
* UI og andet custom rendering såsom en menuknap eller en dialogboks. (da disse f.eks. jo er forskellig fra platform til platform)

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. Hvorefter disse klasser kan traversere rundt på tværs af iOS, Android og Windows via PCL og benyttes i logikken og vises på brugergrænsefladen i app’en.

### PCL og SQLite

Da logikken for PatientApp’en skal kunne 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-Extension[[3]](#footnote-3), som er en meget simpel ORM der tilbyder alle de relationer databasemodellen har brug for på toppen af et SQLite-net[[4]](#footnote-4) 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. Med SQLite – Net Extension kan man via attributter på sine properties angive relationerne. Og man skal ikke oprette tabeller eller kolonner i databasen. Af denne grund, har man fuld kontrol over databasens skema til at persistere entiteterne. SQLite-Net Extensions kræver kun, at man angiver fremmede nøgler, der anvendes til at håndtere relationer og resten finder SQLite 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.

Som nævnt i punktoversigten, skal hver platform implementere en metode til at returnere filstien for hvor SQLite databasen lokalt skal ligge. Og hver skal implementer en metode til at oprette forbindelse til deres SQLite database. Dette kræver et interface som implementeres af tre platformsspecifikke klasser med hver af deres SQLiteConnection. I DAL laget ligger der en klasse, der har en række metoder som skal bruges ifm. Indlæsning/persistering af data fra SQLite databaserne. Denne klasse skal også bruges i de tre platformsspecifikke klasser. Dette kan ses i klassediagrammet for PCL’en i designdokumentet.

### Sekvensdiagrammer (Opret kald) for PCL

For at illustrere hvordan de forskellige lag snakker sammen er der taget udgangspunkt i use case 1.3: Opret kald. Kommunikationen mellem disse tre lag kan vises med et sekvensdiagram der beskriver sekvensen fra en patienten opretter et kald til patienten kan se at kaldet er gået igennem systemet.



Sekvensdiagrammet følger følgende flow: Patienten har behov for at oprette et kald. Patienten vil finde ud af, at der er flere steps han/hun skal igennem afhængig af, om kaldet har behov for at blive specificeret yderligere. Dette foregår via brugergrænsefladen via en tabeloversigt illustreret i Figur X som en pakke af grafiske elementer, da tabellernes opbygning er platformsspecifikt. Efter patienten har valgt en valgmulighed på brugergrænsefladen, sker der en række andre ting som vedrører business logikken, bl.a. CPR validering og tjek på, om patienten er registreret i PatientCare systemet. På baggrund af dette vil kaldet nu blive tilknyttet en årsag. For at sende kaldet benyttes http protokollen, hvor der i den skal angives en række parametre. Bl.a. URL’en til WebAPI’et og indholdet. Indeholdet er i formattet json og derfor vil kaldet blive konverteret til dette format. Efter at kaldet er blevet sendt med et http request, modtages et http response sammen med en indikation på http requestet. Er denne OK, vil der i http responset ligge et id på det oprettet kald. Dette id er blevet genereret af MongoDB databasen som skal gemmes sammen med dette kald. På denne måde er id’et på kaldet ens i både lokaldatabasen på devicet men også på fællesdatabasen på MongoDB. Dette er vigtigt, da samme kald senere skal opdateres enten i form af, at det fortrydes af patienten eller udføres af personalet.

I Figur X vises ikke sekvensen for hvordan kaldet persisteres i lokaldatabasen efter det er blevet oprettet. Dette vises i et separat sekvensdiagram som følgende:



Kaldet der nu har et id genereret af MongoDB tilføjes til en liste af andre kald, der evt. er blevet persisteret i lokaldatabasen. Listen sendes videre sammen med en reference til en klasse der tilgår den platformsspecifik lokaldatabase. Sidst sker selve indsættelse af kaldet til databasen via SQLite API’et som i figur x ses som et komponent.

## PersonaleApp

PersonaleApp er udviklet til en smartphone som benytter sig af android styresystem. Den er udviklet på baggrund af at Systematics opgavesystem til serviceopgave ikke kan håndtere patientkald. PersonaleApp’en skal håndtere modtagelsen af patientkald og ændringen af status på patientkaldene eftersom de opgaver der er forbundet med dem bliver udført. PersonaleApp modtager patientkaldene fra modulet PatientApp via WebAPI’et.

### SQLiteDatabase

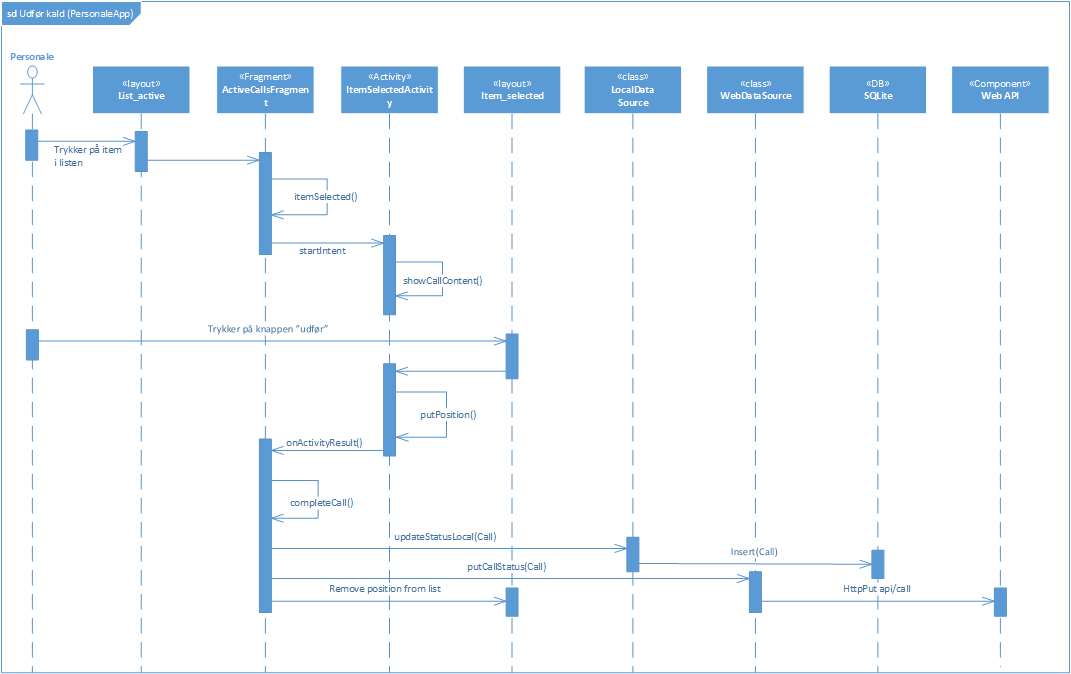
Android har forskellige måder at gemme bruger-og applikationsdata. En måde er at gøre brug af SQlite database. Det er anvendt i PersonaleApp’en til persistering af data fordi der er behov for at gemme flere parametre for et kald lokalt og nemt tilgå det ved at kendskab til et id. En måde er at gøre brug af Shared Preferences hvor data gemmes i key-value pairs, hvilket ikke er at foretrække, da det kræver en nøgle for hver eneste parameter og ikke gør det muligt at hente flere parametre samlet i én omgang. En af hovedprincipperne i SQL databaser er den formelle deklaration for hvordan databasen er organiseret. Dette vises med et skema, hvor tabelnavne og kolonner for den enkelte tabel defineres. For PersonaleApp laves en tabel vedrørende information omkring det modtaget kald fra hvilken patient, stue og afdeling. Disse informationer opstilles her og er dem som persisteres i PersonaleApp.

* Id
* PatientCPR
* PatientName
* Room
* Bed
* Department
* Category
* Choice
* Detail
* CreatedOn
* ModifiedOn
* Status

### Sekvensdiagram (Udfør kald)

Med udgangspunkt i Use Case 2.3 ”Udfør kald” laves et sekvensdiagram der beskriver sekvensen for at et kald udføres i PersonaleApp, hvilket er en af de primære sekvenser for dette modul.

Når patienten har sendt et patientkald og personalet har modtaget det, kan personalet vælge at udføre det. Kaldet vil komme frem på listen over andre afventende kald. Hvis personalet trykker på patientkaldet fra listen, ledes personalet videre til en ny side, hvor der vil stå detaljer omkring patientkaldet, bl.a. hvad årsagen er. Det ses i figur x ved at metoden ”showCallContent” kaldes, efter at personalet har trykket på patientkaldet. Da listen er indekseret i rækker fra 0 til antal kald, der ligger i listen, vil positionen af den række fra listen fra hvor kaldet blev valgt fra blive overført til den nye side via en intent. På denne nye side vil der være en knap, der viser at personalet kan udføre kaldet, og når personalet trykker udføre, bruges denne position til identificer kaldet, som skal udføres. Ud fra denne information opdateres kaldets status fra afventende til udført, hvilket indsættes i SQLite databasen på devicet og derefter sendes til WebAPI’et med http protokollen. Kaldet fra listen over afventende kald, vil blive slettet ud fra den given position og på den måde vil personalet se at kaldet er udført.



Figur 5 Sekvensdiagram for Udføre kald for PersonaleApp

### Bound Service (Modtag kald)

Når et nyt kald fra patienten er blevet sendt fra PatientApp’en, er der brug for en teknologi til at notificere personalet. Da det er prioriteret at vise konceptet fremfor performance, er der valgt en pull funktionalitet til at indlæse nye kald fra systemet. Denne pull funktionalitet kan enten laves ved at der trykkes på en knap eller 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.   
En service i android kommer i to udgaver: En 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 kalderen. 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 PatientApp’en er der behov for det sidstnævnte, da der løbende skal hentes patientkald fra en webserver via WebAPI’et. Derfor er der valgt en Bound service.

”MainActivity” er den aktivitet der starter op når brugeren af PersonaleApp er logget ind. I Mainactivity starter servicen også. Servicen henter alle afventede kald, dvs. de kald der ikke er blevet udført. De afventende kald hentes fra WebAPI’et via et http request til en URL. WebAPI’et returnerer en string af json objekter som deserialiseres til konkrete objekter – på samme måde som beskrevet for PatientApp’en .



Figur 10 Diagram der viser strukturen for en Bound Service for PersonaleApp

I figur 29 ses det hvordan servicen bliver oprettet i PatientApp. Til højre i diagrammet ses tidslinjen for aktiviteter og andet logik der kører i hovedtråden. Og til venstre ses tidslinjen for aktiviteter der kører i en separat tråd. Da en service som default kører i hovedtråden er der behov for at oprette en anden tråd, som servicen skal køre i, da servicen vil komme til at lave blokerende operationer. Bindingsprocessen ses i figur 29 hvor android frameworkets ”Service” implementeres af ”CallService” og for at tillade bindingen, skal onBind() implementeres. Denne metode returnerer et IBinder objekt, der gør det muligt for klienten at interagere med servicen. Når bindingen er sket, kaldes en metode ”showCall()” der vil lave en række operationer der sker i baggrundstråden i stedet for at blokere hovedtråden:

1. Hente nye kald fra en webserver via et WebAPI.
2. Indlæse eksisterende kald fra lokal databasen via SQLite.
3. Gemme resultaterne i servicen
4. Åbne en ny dialogboks med information omkring det nye kald

Eksisterende afventende kald indlæses fra fælles databasen hvor eventuelle nye kald er gemt. Det indlæste sammenlignes med afventende kald fra lokal databasen. Hvis det indlæste afviger fra det lokale data er det udtryk for at der er kommet et nyt afventende kald. Resultatet gemmes i servicen, så det kan tilgås via den aktivitet, som har bundet sig til den, for herefter at vise en dialogboks tilbage på hovedtråden med informationer om det nye kald.

Disse operationer sker i baggrundstråden fordi de indebærer en række intensive operationer og requests, som man som bruger af PatientApp’en ikke vil vente på, hvis de blokkerede hovedtråden.

Da requests til webserveren sker asynkront grundet det framework der benyttes til det, og da det kan tage noget tid at få respons tilbage, må baggrundstråden stå og vente på at operationerne er færdige, før den kan starte næste iteration. Her står den og venter hvert sekund til dette er sket. Dette er illustreret i figur 29 ved at der tjekkes på en bool variable ”isSuccess”.

Servicen er til stor gavn for PatientApp’en og en fremragende løsning hvor det ikke kræver en brugerinterkation for at gøre brug af pull funktionaliteten. Performancemæssigt vil det være en bedre løsning gøre brug af push funktionalitet, da man på den måde ikke forgæves brænder PatientAppen’s CPU resurser og tid af i en situation, hvor der ikke er kommet et nyt patientkald. Dette kræver dog at push funktionen implementeres i WebAPI’et som skal sørge for at patientkaldet der modtages fra PatientApp sendes videre til PersonaleApp. Konceptet er prioriteret højere end performance i det tilfælde.

## 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. Det er i sekvensdiagrammerne for AdminApp, ikke muligt at identificere de implementerede funktionskald, da diagrammerne og implementeringen ville være hårdt koblet sammen og ikke give anledning til ændringer. Med andre ord, hvis en use case ændres, som ville ændre sekvensen af use casens udførelse, vil det kræve ændring i diagrammet også. 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’et og måden denne kommunikation sker på, ændres ikke umiddelbart.

Views, Controllers og Models i AdminApp, følger MVC standarden hvor views er opdelt efter hvilken del af applikationen de står for. F.eks. ligger alle views som har noget med ”category”(Kategori) at gøre, i en mappe der hedder ”Views/category”.

Selve designet for indholdet af de forskellige views er der ikke tænkt videre over. Der har været behov for at oprette, se og slette nogle data og sidernes layout er lavet efter dette behov.

## WebAPI

For web API’et har designprocessen taget udgangspunkt i alle use cases i kravspecifikationen hvor en device har behov for en integration ud af eget system. Det er f.eks. ved oprettelse af et kald fra patientApp.

Det er forsøgt at udvikle web API’et først efter tankegangen om *API Driven Development*, og dermed have en klar datamodel for resten af systemet, men på grund af database skifte, hvilket der kan læses mere om i afsnit x, gjorde at udviklingen af web API’et måtte gentænkes og på daværende tidspunkt var udviklingen af resten af systemet godt i gang. I den efterfølgende udvikling af web API’et har der været løbende design og implementering efter systemet behov. F.eks. ved start af nu iteration, er der startet ved web API’et og arbejdet ud.

Web API’et er bygget i ASP.NET web API 2 som bygger på MVC frameworket, og derfor er opbygningen af mappe strukturen derefter. Denne template kommer med et forebygget *Area,* som står for at genererer hele dokumentations viewet. Dette *Area* benytter sig af Visual Studio *Summaries* og funktionsdefinitionerne i implementeringen til at danne et view der til dels virker optimalt.

### Database

I designprocessen af databasen har projektgruppen skulle blive enige om, hvordan datamodellen skulle se ud for systemet med udgangspunkt i hvilke data en patient har behov for, for at kunne oprette et patientkald og med ligeledes hvilke data personalet har behov for, for at kunne udføre et patientkald. Denne datamodel ligger til baggrund for de modeller WebAPI’et modtager og returnerer.

Det har givet anledning til 2 databaser, hvor den ene er en datamock. Denne datamock forestiller de data der regnes med at kunne trækkes fra Cetrea ved en evt. integration til deres systemer. Denne database består primært af én collection, hvor information om patienten og hvor på sygehuset patienten er indlagt bliver gemt.

Den anden database indeholder de data som et patientkald benytter sig af ved oprettelse et patientkald, kategorier, typer og detaljer og har hver deres *collection* hvor disse data er opbevaret.

Projektgruppen besluttede at skifte database midt i udviklingsfasen, til en NoSQL database hvor der ikke umiddelbart bruges relationer. Databasen burde have været redesignet til at passe til den nye database struktur i stedet for at benytte designet for den relationelle database.

Når en patient fx opretter et kald via PatientApp genereres der noget data om hvad årsagen til kaldet er. Dette skal kobles sammen med hvem patienten er og hvor patienten er indlagt før det modtages af personalet. Her beskrives oprettelsen af et patientkald for at vise princippet i hvordan data genereres på PatientApp’en, distribueres i databasen og benyttes på PersonaleApp’en.

Når et patientkald oprettes på PatientApp leveres det med tilhørende parametre til WebAPI’ets webside. Herfra gemmes kaldets parametre i den fælles database og knyttes sammen med patientens mock data ud fra CPR-nummeret som er tilknyttet kaldet. Fra WebAPI’et hentes alle parametrene for dette kald ned på PersonaleApp’en, hvorfra personalet modtager kaldet med alle oplysninger om hvem patienten er, hvor patienten er indlagt, hvad årsagen til kaldet er og hvornår det er oprettet med mere.

Hvis et patientkald oprettes med succes, afhængig af om patientens CPR-nummer findes i den fælles database, bliver kaldets status bliver sat til 0. Kaldets status er en C# enum, hvis denne er 0 er kaldet afventende, er den 1 er kaldet udført og er den 2 er fortrudt.



På figur X ses en oversigt over hvor de forskellige parametre kommer fra.

### Data mock

PatientCare er udviklet med henblik på at trække data fra et system som Cetreas kliniske logistik hvor brugbare oplysninger i forvejen registreres. For at PatientCare kan blive en realitet i praksis er det derfor nødvendigt at være understøttet af et system som dette, der leverer de nædvendige oplysninger. Da det ikke har været muligt at integrere til Cetrea i dette procesforløb har projektgruppen opsat et data mock af det data der skal bruges til prototypen af PatientCare, som ligner det data Klinisk Logistic kan levere.

Data mock består af data der indebærer følgende:



# Resultater

// Dette afsnit skal beskrive hvad resultatet er blevet

## Mini-MTV

Her beskrives de resultater der er kommet ud af Mini-MTV analysen.

Der er indikationer på 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 Gyn-obs i Randers. Det viser resultater fra Mini-MTV-analysen.

På Gyn-obs i Randers har personalet tildelt patientpleje, det vil sige at indlagte patienter primært har én kontaktperson under indlæggelsesforløbet. 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 på en travl dag. Det har de ikke kunnet tidligere på grund af manglende information om årsagen til kaldene. Det viser sig med al tydelighed at personalet kan sparre en del tid og dermed udnytte deres sygeplejekompetencer mere effektivt. Også antallet af skridt kan 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 viser at personalet kan sparre rundt regnet hvad der ligner *3000 skridt* og mere end *1 time* pr vagtved 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 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, 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 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 sikkerhed 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 der er patienter der vil få en bedre oplevelse ved at være indlagt, når de får mulighed for at benytte en app i stedet for kaldesnoren, da nogle opfatter snoren som noget man skal benytte i nødstilfælde. Med PatientCare kan patienterne tillade sig at spørge efter hjælp selvom det ikke er akut eller et nødstilfælde. 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 eller akutte opgaver.

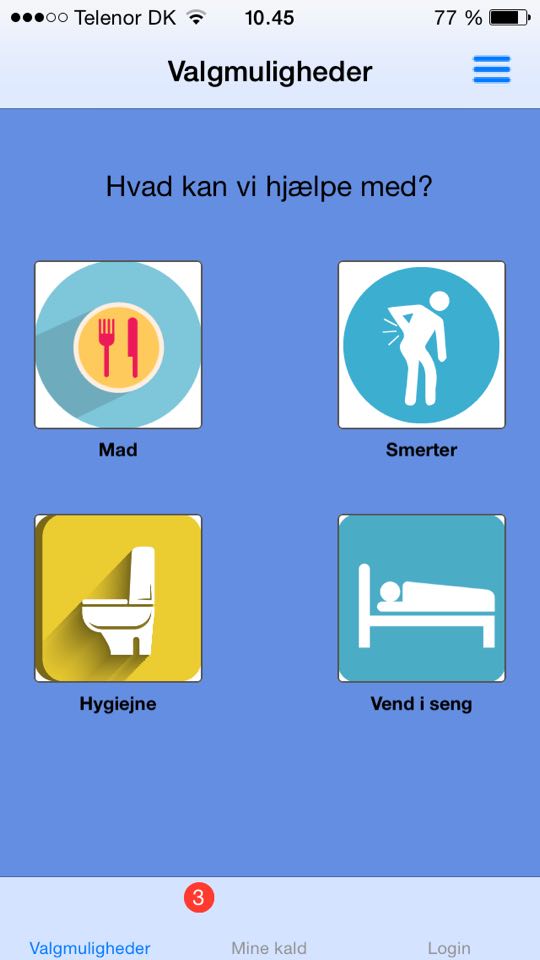
Implementeringen af PatientCare vil ikke kræve de store organisatoriske ændringer, men der vil i en opstartsfase være behov for en tovholder og en erfaren person udefra der kan hjælpe personalet i gang med det nye system. Det dobbelte administrative arbejde med tildeling af patientpleje der har været med det nuværende kaldeanlæg undgås og bliver med en implementering af PatientCare kun registreret 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 App og til en hjælp er der udviklet en pjece henvendt til patienterne, som med fordel kan udleveres til de patienter der måtte have interesse i at afprøve den nye teknologi.

Det efterspørges på Regionshospitalet i Randers at man arbejder videre med projektet, da de ser et potentiale.

## Layout og funktionalitet

### PatientApp

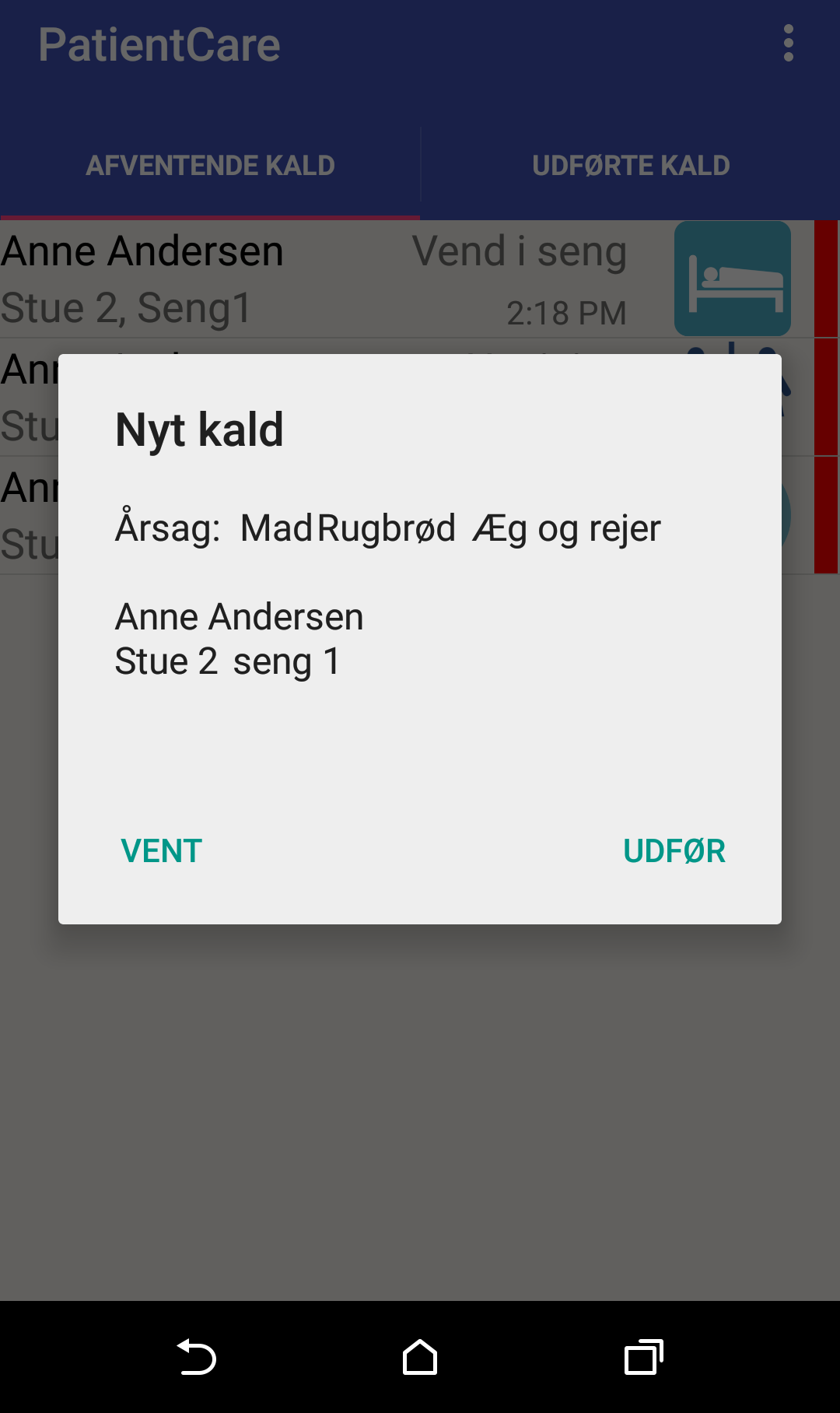
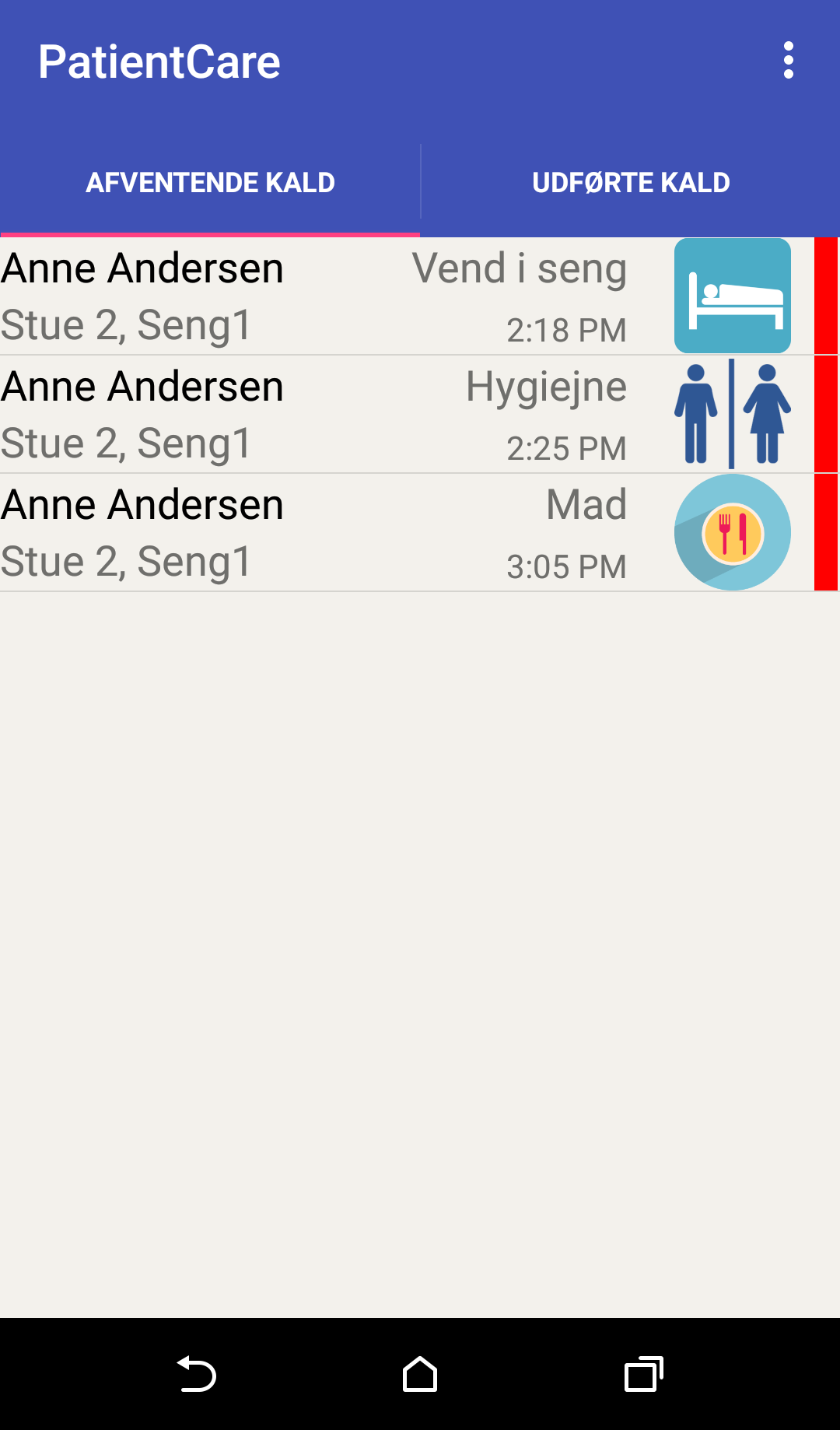
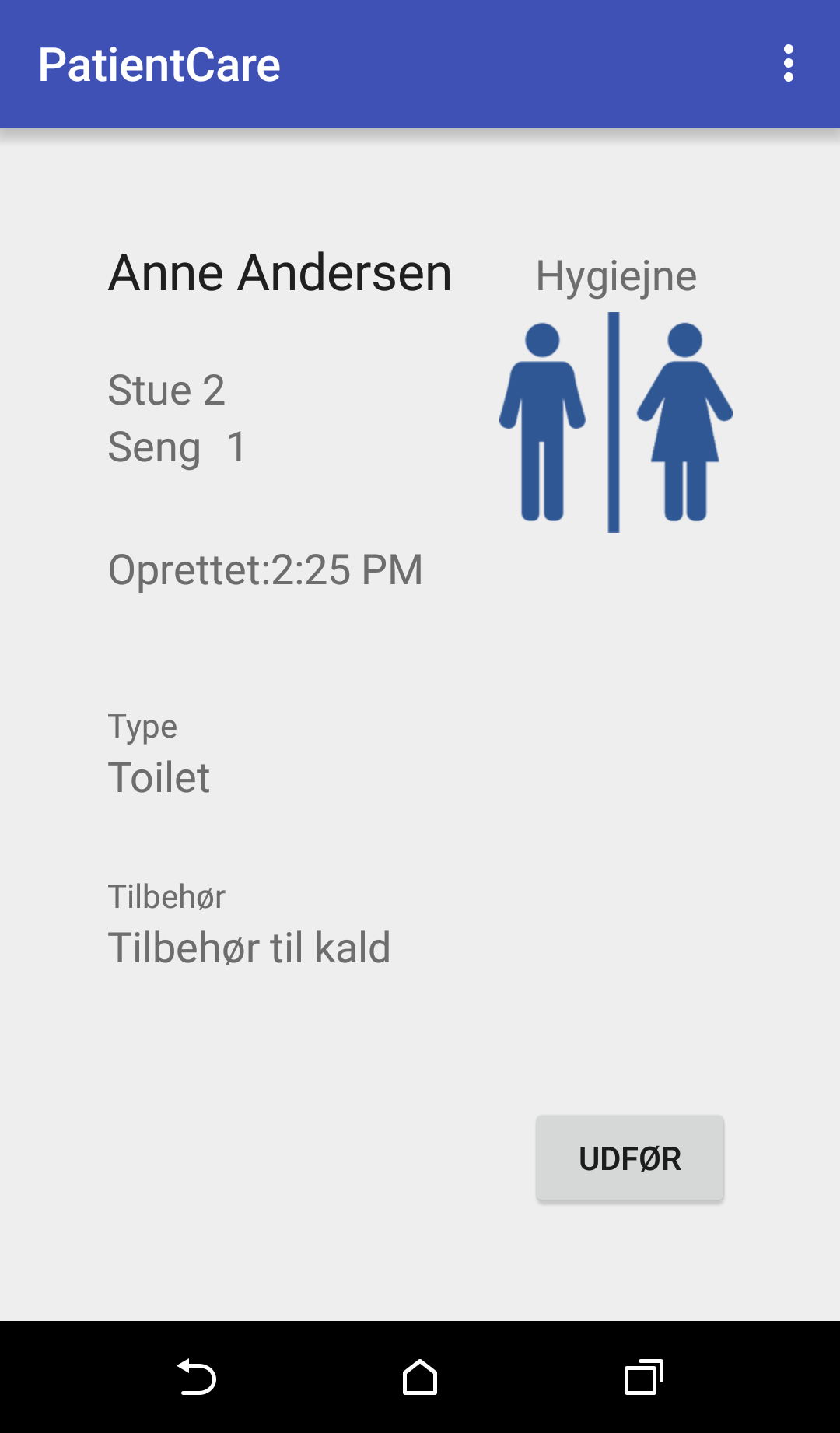
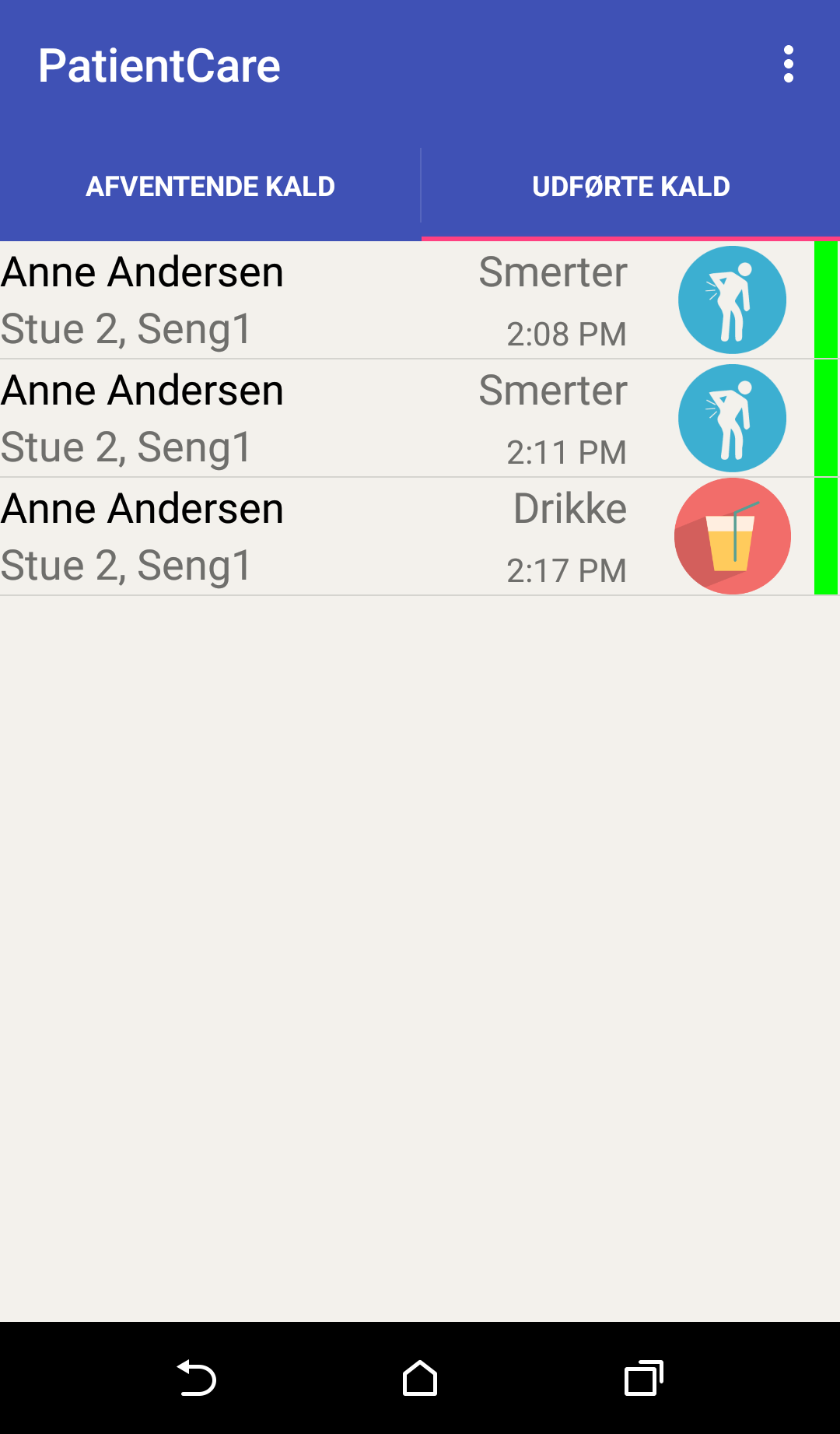
Patienten kan knytte en årsag til patientkaldet i tilfælde hvor patienten har behov for en serviceydelse, som fx et glas vand, hjælp til at komme på toilettet eller lignende. Dette sker ved at patienten vælger mellem nogle foruddefinerede valgmuligheder på App’en, som er bygget op så patienten nemt og effektivt kan sende et patientkald afsted, uden for megen stillingtagen. Udover at en patient 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 de behov for pleje patienten har haft.

Figur X og X viser screenshots to vigtige views på PatientApp’en henholdvis *Valgmuligheder* og *Mine kald*.

### PersonaleApp

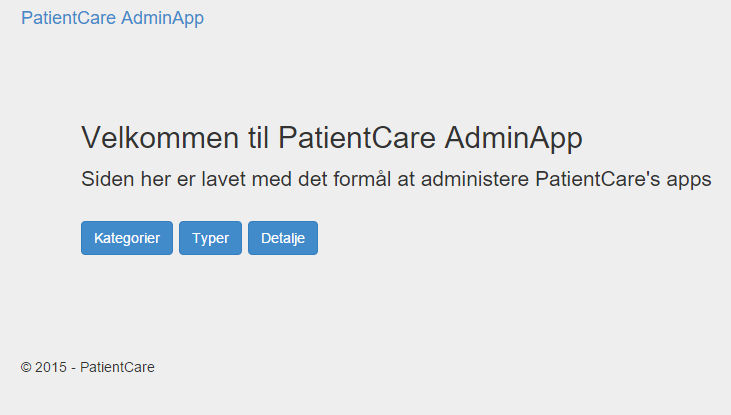
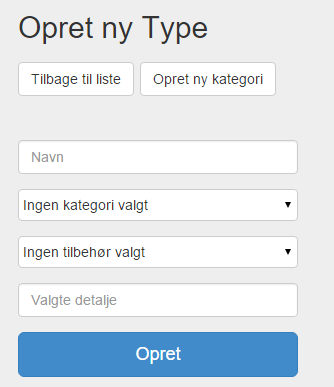
PersonaleApp’en gør personalet opmærksom på når der er et nyt patientkald ved at telefonen vibrerer og giver lyd fra sig samtidig med at der kommer en dialogboks med information om kaldet. Når et nyt patientkald modtages kan den pågældende sygeplejerske tage stilling til om at udføre det med det samme eller lade det vente. Det er tydeligt hvad årsagen til patientkaldet er og hvor den pågældende patient er indlagt. På den måde bliver sygeplejersken gjort opmærksom på årsagen til kaldet og kan forberede det efterspurgte til rette vedkommende. Når sygeplejersken har mange afventende patientkald kan hun se en liste over dem med statusfarven rød, der indikerer at de er afventende. Hun kan vælge at prioritere imellem patientkaldende ved at udføre dem hun vurderer til at være vigtigst først. App’en har også en liste over udførte kald, som personalet blandt andet kan bruge til at dokumentere plejen omkring patienterne på den pågældende afdeling.

Figur X, X, X og X viser fire vigtige views på PersonaleApp’en henholdvis Nyt kald, Afventende kald, detaljer om et kald og udførte kald.

### AdminApp

**Funktioner**  
Når en afdeling skal i gang med at bruge PatientCare skal systemet tilpasses i AdminApp. Dette gøres på AdminApp. Formålet med at tilpasse systemet til hver enkelt afdeling er at tilbyde noget unikt for hver afdeling der passer til patienternes behov og personalets arbejdsgang, da dette kan variere meget. Det er her det bestemmes hvilke valgmuligheder patienterne har når de skal kalde personalet gennem PatientApp. I AdminApp har en administrator 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 for hele systemet. Tilpasningerne bliver udstillet af WebAPI’et og bliver brugt af PatientApp’en til at vise de foruddefinerede valgmuligheder patienterne har mulighed for at vælge når de har behov.



Figur X, Y og Z viser tre vigtige views på AminApp. På figur X ses velkomstsiden, hvor man kan tilgå kategorier, typer og detaljer. På figur Y ses en oversigt over de kategorier der allerede er oprettet og på figur Z ses siden hvor man kan oprette en ny type.

## Status på prototype

Resultater fra Accepttest

### PatientApp

**Status**  
På nuværende tidspunkt kan prototypen af PatientApp køre på iOS og det vurderes at det kræver nogle ugers arbejde at programmere frontenden så app’en ligeledes kan køre på Android og Windows.

PatientCare systemet har formodet at lave en cross-platform applikation, som patienterne kan installere og bruge på deres egen telefon. Cross-platform løsningen er dog kun lavet til iOS da denne er en prototype med det mål at få vist konceptet af PatietCare systemet. Funktionalitets mæssigt har applikationen opfyldt alle Use Casene på baggrund af prototypen.

Prototypen for PatientApp fungerer lige nu sådan, at den benytter sig af pull funktionalitet, når nye valgmuligheder skal indlæses fra adminsmodulet. Samme fremgangsmåde benyttes når status på mine kald for patienten skal opdateres. Her skal patienten manuelt trække ned i på viewet for at refreshe og dermed lave et pull request for at opdatere data’en. Dette er ikke hensigtmæssigt for brugervenligheden, når patienten måske i nød ligger og har rygsmerter og gerne vil have respons med det samme, så snart kaldet er sendt. I stedet vil en bedre løsning være at lave et push request, når data’en har ændret sig. Grunden til at prototypen benytter sig af pull funktionalitet er, at teknologi mæssigt er det hurtigere og nemmere, og det var netop målet med prototype. I fremtidigt arbejde vil det være oplagt at benytte push funktionaliteten og det vil en yderst forbedring for brugervenligheden og brugeroplevelsen for patienten.

Prototypen af patientApp’en tjekker ikke om patienten er indlagt når patienten opretter kald, kun når patienten logger ind

### PersonaleApp

PersonaleApp er udviklet på baggrund af at Systematics opgavesystem til serviceopgave ikke kan håndtere patientkald. PersonaleApp skal håndterer modtagelsen af patientkald og ændringen af status på patientkaldene eftersom de opgaver der er forbundet med dem bliver udført. PersonaleApp modtager patientkaldene fra modulet PatientApp via WebAPI’et.

Behovet for at kunne gemme data lokalt på en SQLite database opstår i tilfælde hvor internetforbindelsen slår fra. Det gør det mere brugervenligt at brugeren af systemet kan se det data der sidst er blevet modtaget sammenlignet med slet ikke at kunne se noget data. SQLite er en lokal SQL database i App’en på smartphonen.

**Status**  
De funktionelle krav som blev stillet i kravspecifikationen er på nuværende tidspunkt opfyldt. 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. Denne fordeling af patientkald skal ske i systemets WebAPI, men er ikke implementeret i prototypen af PatientCare systemet, fordi andet funktionalitet er prioriteret højere. (Læs mere om primær plejer og tildelt patientpleje i Mini-MTV’en).

Prototypen af personaleApp’en bruger ikke push notifikationer når der kommer et nyt kald, men i stedet pull fordi der er prioriteret at vise konceptet fremfor performance

Ønskeliste:  
Lyd og vibration ved nyt kald

### AdminApp

Formålet med at lave et administrations modul til PatientCare systemet var at man kunne tilpasse en patients forudbestemte valgmuligheder som en patient kan benytte i forbindelse med et patientkald med en årsag. AdminApp er udviklet som en prototype og opfylder kravspecifikations must haves for AdminApp. AdminApp giver en administrator muligheden for at oprette nogle forudbestemte valgmuligheder for én afdeling på et sygehus. Dette har været det primære fokus da vi har fokuseret på at kunne fremvise konceptet, *patientkald med en årsag*. Efter møde med Randers sygehus hvor vi testede systemet, blev det klargjort at personalet gerne ser mange flere funktioner tilføjet til AdminApp, men det har vi valgt ikke at gå videre med på nuværende tidspunkt. Hvis man skulle gå videre med projektet ville det kræve en analyse af forbedringer til systemet.

### WebAPI

Formålet med at udvikle et WebAPI til at tage sig af integrationerne i systemet og være den eneste adgang til databasen. WebAPI’et var fra starten tænkt til at blive udviklet i tankerne om *API driven development* og det gik også meget godt da systemet var opbygget omkring den relationelle database. Alle i gruppen var enige om hvordan datamodellen så ud og der kunne udvikles omkring denne tankegang. Men udefra kommende problemer med databasen, mere information om dette kan læses i afsnittet om databasen, gjorde at gruppen valgte at skifte fra mySQL til MongoDB, som jo er en noSQL database, og slet ikke bygger på den datamodel. WebAPI’et var det tidspunkt ikke længere at der var blevet implementeret integrationen mellem patientApp og personaleApp, og de to apps var ikke helt klar til denne integration så gruppen mente ikke at det burde skabe de store problemer at skifte over. Datamodellen blev ændret og WebAPI’et ændret til at kunne kommunikere med mongoDB i stedet for mySQL databasen, som den før havde snakket med. Det skabte nogle problemer rundt i hele systemet som gruppen fik taget hånd om og omgået.

Efterfølgende blev WebAPI’et videre udviklet i takt med at systemets use cases blev implementeret. Fx har AdminApp brug for at tilføje eller slette kategorier, typer og/eller detaljer, hvilket gjorde at WebAPI’et skulle udvides.

WebAPI’et har ingen form for sikkerhed og kræver ikke at man sætter authentication headeren på http requestet. Dette bør være næste skridt for WebAPI’et, da det ikke vil være acceptabelt at have et WebAPI’et som står piv åbnet, og specielt ikke hvis WebAPI’et skal bruges i produktion, hvor der vil være person følsomme data som fx en patients CPR nummer.

Man vil i implantationen af WebAPI’et opdage at der er få controllers som benytter sig af et repository pattern, men langt fra alle. Det er meningen at alle controllers skal benytte sig af repository pattern, men er i udviklings fasen blevet nedprioriteret fremfor at få prototypen hurtigere klar. Dette forventes at blive implementeret efter aflevering af rapporten.

**Validering af patient**  
På nuværende tidspunkt validerer systemet patienten ved at sammenligne det CPR-nummer der bliver sendt afsted med det nyoprettede kald med de CPR-numre der er lagret i fællesdatabasen. Hvis det stemmer overens er det udtryk for at patienten med dette CPR-nummer 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’et ved at tjekke i fælles databasen ud fra data mock om patienten har et indlæggelsestidspunktet der er før ”nu” og et udskrivelsestidspunkt der er efter ”nu”.



På diagrammet ovenfor ses et aktivitetsdiagram over de overvejelser projektguppen har gjort sig I forbindelse med validering af om patienten er indlagt eller ej.

# 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 projektforløbet 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 fuldendte 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 ligger 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 skal løses. 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 PateintCare 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.

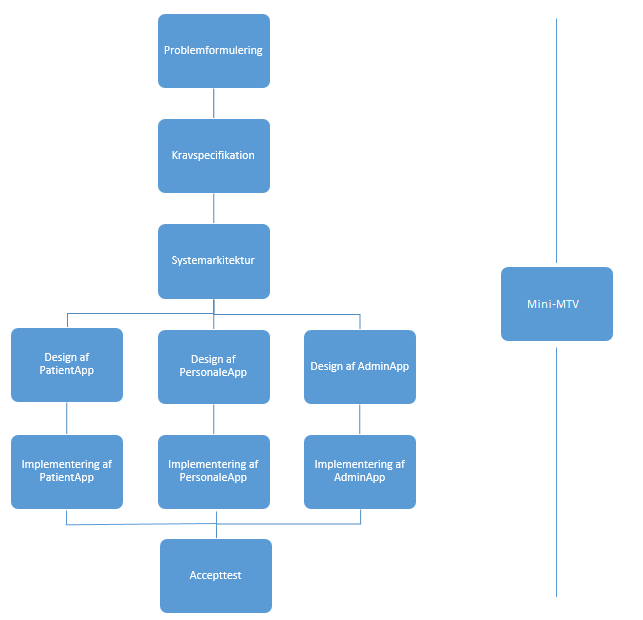
### Tidsplan

Tidsplanen blev brugt som en oversigt over hvilke faser af projektet der ville veje størst i bestemte perioder af den tid der var stillet til rådighed. Tidsplanen er opsat som et Gantt-skema der illustrerer ugerne som et resumé af projektforløbet. 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 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

### **Agil systemudvikling**

Figur X viser de udviklingsfaser som projektgruppen i store træk har været igennem. 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.



**Iterativ proces**Projektgruppen har fulgt en iterativ udviklingsproces som vist på figur X, hvilket giver et fleksibelt og tilpasningsdygtigt system, Da PatientCare systemet 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.

De opgaver der følger med kravene i use casene er løst i iterationer. I hver iteration designes, 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 sideøbende 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 – patientkald med tilknyttet årsag 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 om i dokumentet Mini-MTV.

**Interessentanalyse**For at undersøge hvilke interessenter der berøres af den nye teknologi er der lavet en interessantanalyse.

**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 litterur. 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 blev 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’en 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. Disse ansvarsområderne for hvert enkelt medlem kan læses herunder.

**Anders (IKT-studerende)**Har haft ansvar for implementering af WebAPI, AdminApp og fællesdatabase

**David (IKT-studerende)**Har haft ansvar for implementering af PatientApp. Har været medvirkende til implementeringen af PersonaleApp.

**Minna (ST-studerende)**Har haft ansvar for at undersøge slutbrugernes behov for et system hvor man kan knytte en årsag til patientkaldet i samarbejde med Camilla. Minna har haft ansvar for store dele af udarbejdelsen af Mini-MTV ud fra den teknologiske og patientmæssige perspektiv samt implementering af PersonaleApp i Android Studio og systemarkitektur og design af app’en der er henvendt til personalet på en hospitalsafdeling.

**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’en 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 og 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 Roshold 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 Systemtics IT-medarbejdere. Systematic har ikke fungeret som kunde på projektet men har fra start af vist stor interesse og set potentiale i projektet. Flere medarbejdere har været behjælpelige med at svare på spørgsmål undervejs, udlevere udviklingsdokumentation omkring Opgavesystemet og videregive kontaktoplysninger på sundhedsfaglige og IT-folk i Regionen.

### Samarbejde med sundhedsfaglige

Via Systematic har projektgruppen fået kontakt til en innovationskonsulent/projektleder fra Regionshospitalet i Randerrs, der fået Gynækologisk - Obstetrisk afdeling på Regionshospitalet i Randers med til 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.

# Diskussion

Her diskuteres hvilke overvejelser projektgruppen har gjort, hvilke udfordringer ….  
hvorfor vi har valgt det vi har gjort og hvorfor det er godt for projektet og hvorfor det ikke er godt for projektet. Hvad kunne vi have lavet om og gjort bedre?

Hvilke valgt har vi gjort i forhold til projekt forløbet – hvad kunne have være bedre?

## Overvejelser

### Projektforløb

Projektet bærer præg af at der har været mange bolde i luften på samme tid 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 på alle punkter, da der har været mange inputs fra eksterne fagpersoner om hvilken retning projektet kunne tage.

### Skift af database

Projektgruppen besluttede at skifte database midt i udviklingsfasen, fra en Relationel datbase til en NoSQL database hvor der ikke umiddelbart bruges relationer. Databasen burde have været redesignet til at passe til den nye database struktur i stedet for at benytte designet for den relationelle database.

Når en patient fx opretter et kald via PatientApp genereres der noget data om hvad årsagen til kaldet er. Dette skal kobles sammen med hvem patienten er og hvor patienten er indlagt før det modtages af personalet. Her beskrives oprettelsen af et patientkald for at vise princippet i hvordan data genereres på PatientApp’en, distribueres i databasen og benyttes på PersonaleApp’en.

Når et patientkald oprettes på PatientApp leveres det med tilhørende parametre til WebAPI’ets webside. Herfra gemmes kaldets parametre i den fælles database og knyttes sammen med patientens mock data ud fra CPR-nummeret som er tilknyttet kaldet. Fra WebAPI’et hentes alle parametrene for dette kald ned på PersonaleApp’en, hvorfra personalet modtager kaldet med alle oplysninger om hvem patienten er, hvor patienten er indlagt, hvad årsagen til kaldet er og hvornår det er oprettet med mere.

Hvis et patientkald oprettes med succes, afhængig af om patientens CPR-nummer findes i den fælles database, bliver kaldets status bliver sat til 0. Kaldets status er en C# enum, hvis denne er 0 er kaldet afventende, er den 1 er kaldet udført og er den 2 er fortrudt.

### PatientApp – uden Xamarin

Ud fra kravspecifikationen skal PatientApp’en udvikles i et miljø der gør det muligt at understøtte de tre største smartphone styresystemer iOS, Android og Windows, da PatientApp’en er tiltænkt til at være en *Bring your own device* løsning. Den traditionelle måde at løse det på er at udvikle en app for sin platform i hvert sit sprog.

De tre sprog er:

* Objective-c eller swift
* Java
* C#

Dette vil kunne opfylde behovet for at udvikle PatientApp’en til de 3 platforme, men foruden at kunne 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.

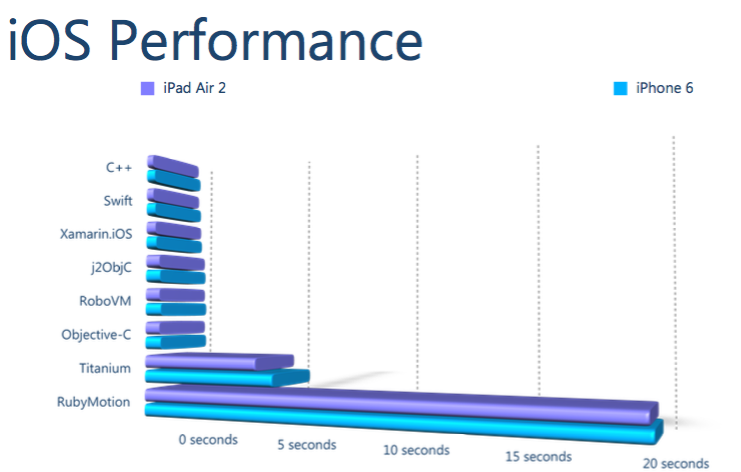
### Valg af Xamarin

I forbindelse med teknologiundersøgelse mht. Xamarin stødte gruppen i bedre tekniske alternativer ifm. Et oplæg hos Mjølner Informatics omkring Xamarin platformen. Der blev drøftet to Xamarin løsninger, hvor den ene løsning var den traditionelle løsning og en anden løsning som gør brug af Xamarin.Forms. På dette tidspunkt var gruppen allerede godt i gang med implementeringen af den traditionelle løsning men alligevel fik gruppen rigtig meget inspiration til hvordan PatientApp’en kunne have udvikles på en mere effektiv, og hurtigere måde for Proof of Concept af PatientCare systemet. På runtime vil hver side og dens kontroller blive mappet til platform-specific native brugerelementer. Fx vil et tekstfelt med Xamarin.Forms blive til et UITextView på iOS, EditText på Android og TextBox på Windows.

Men hvorfor er Xamarin bedst for fx iOS?

* Apple’s iOS SDK kan fås 100 % med C#.
* Xamarin.iOS app’en kompileres direkte til native ARM assembly kode, som betyder at app’en kan distribueres direkte til App Storen, som man normalt ville.
* Det er muligt at benytte .NET navnekonventioner til at få adgang til Objective-C API’er.
* Kalde eksisterende Objective-C kode i C#

Performance mæssigt er en Xamarin løsning også bedre:



Det ses at Xamarin.iOS performer bedre end Objective-C. Det betyder meget for PatientCare systemt, at performance er god, da man som patient ikke skal blive frusteret over lange responstider – især ikke i kritiske tilfælde, som når vedkommende har moderate rygsmerter.

**PatientApp – Løsning med Xamarin.Forms**

I forbindelse med teknologiundersøgelse mht. Xamarin stødte gruppen ind i bedre tekniske alternativer ifm. Et oplæg hos Mjølner Informatics omkring Xamarin platformen. Der blev drøftet to Xamarin løsninger, hvor den ene løsning var den traditionelle løsning, som PatientCare systemet har valgt, og en anden løsning som gør brug af Xamarin.Forms. På dette tidspunkt var gruppen allerede godt i gang med implementeringen af den traditionelle løsning men alligevel fik gruppen rigtig meget inspiration til hvordan PatientApp’en kunne have udvikles på en mere effektiv, og hurtigere måde for Proof of Concept af PatientCare systemet. På runtime vil hver side og dens kontroller blive mappet til platform-specific native brugerelementer. Fx vil et tekstfelt med Xamarin.Forms blive til et UITextView på iOS, EditText på Android og TextBox på Windows.

**Fordele**

Xamarin.Forms bygger på også på princippet om at have en C# fælles kodebase som den traditionelle Xamarin løsning, men med Forms deles UI også og dermed er der 100 % code sharing. Xamarin.Forms elementer mappes automatisk til nativ kontroller og adfærd.

**Ulemper**

UI’en skal skrives direkte i C# eller i XAML, da det ikke er muligt at bruge drag-and-drop funktionalitet som på den tradionelle måde, eftersom UI’en er forskellig. Derudover er custom rendering svært og bør undgås i Xamarin.Forms, da man ellers ligeså godt kan bruge den tradionelle, da det skal laves for hver platform. Style layoutet er også svær, da hver layout er forskellig fra hver platform, fx font størrelsen kan være forskellige selvom samme størrelse er sat på dem alle. Derudover skal man kende til de forskellige platformes API’er på trods af alt skrives i C#.

For PatientCare systemet har den tradionelle Xamarin være en udmærket løsning. Kendskab til iOS, Android og Windows udvikling har på forhånd være god, hvilket har gjort det nemt at sætte sig ind i de forskellige API’er med brug af C#. Men havde systemet benyttet sig af Xamarin.Forms vil det udviklings-og tidsmæssigt være en meget bedre løsning for Proof of Concept og pga. den korte udviklingstid i forhold nuværende løsning ville prototypen af PatientApp hurtigere havde nået en Android og Windows løsning. Det ville have betydet, at systemet ville have bedre levet op til konceptet om ”bring your own device”, som var det mål der blev sat fra start.

### Android

#### Fragments

Da PersonaleApp’en benytter sig af en tab menu til at skifte imellem listen af afventende kald og udførte kald, kræver det brugen af fragments. Fragments kan nemt genbruges i menuen og skiftes og det gør app’en modulertbart. Tab funktionaliteten i Android kræver også brugen af Fragments, da hver tab benytter en Fragment transaktions objekt. Fx når man i app’en står i listen af afventende kald, som ligger i en fragment for sig selv, og trykker på udførte kald tab, sker der en transaktion, hvor det pågældende fragment skiftes ud med listen af udførte kald, der også er et fragment. Dette giver en dynamisk og fleksibel brugergrænseflade. Derudover undgås redundans i form af, at koden ikke dubleres. Dette er fordi tab menuen laves i det samme view og fragmenterne ligger i den samme container nedenunder, og det er derfor det samme view, der bliver vist. Hvis det skulle laves med aktiviteter, skulle tabmenuen og dens funktionalitet dertil kopieres i hver aktivitet, den bruges.

#### Brug af service

Der er behov for en teknologi til at notificere personalet automatisk for indkommende kald fra patienterne, fordi det ikke vil være hensigtmæssigt at lade personalet indlæse disse manuelt. Dette teknologi kan være i form af 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. En anden metode til at håndtere baggrundsopgaver vil også kunne være brugen af en baggrundstråd, men dette vil være mest oplagt, hvis personalet interagere med app’en, og det vil oftest være én operation der kontinuert udføres I denne tråd. En service kan derimod udstille et client-server interface, hvilket gør det muligt at interagere med servicen, sende request og få resultater – selv på tværs af andre processor. Og da app’en løbende skal hente patienternes kald fra en webserver via WebAPI’et og vise en dialogboks på baggrund dette, er det derfor smartest at bruge en service. For at se hvordan servicen er konstrueret, og hvordan PersonaleApp’en benytter den i praksis, henvises der til underafsnittet ”Bound Service (Modtag kald) under designafsnittet for Designdokumentet.

#### Volley

For at sende http requests til WebAPI’en for at hente indkommende kald, er det blevet valgt at bruge et http bibliotek kaldt ”Volley”, som gør netværkskommunikation nemt med Android. Med ”Volley” er det muligt at lave http requests meget nemmere, hurtigere og uden at skrive for meget kode. Og som standard, er alle http requests med ”Volley” asynkrone, hvilket betyder at requestene kan laves uden at være en bremseklods for resten af PersonaleApp’en.

### Databaseskift

Det blev besluttet at bruge en relationel database og lagre data i tabeller med kolonner og rækker. Beslutningen om at gøre brug af en relationel database blev taget ud fra det faktum at alle i gruppen har 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 brug af MySQL databaser på en databaseserver. Her oprettede gruppen tabeller i overensstemmelse med den datastruktur som var fastlagt for PatientCare. På grund af et begrænset antal *credits* som hurtigt blev opbrugt på Microsoft Azure, var gruppen nød 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, men da den er afhængig af VPN (Virtual Private Network) er det ikke muligt for vores projekt. Derfor faldt valget af database på MongoDB. Tankegangen med datastrukturen er ikke den samme i MongoDB som for relationelle databaser. Med MongoDB gemmes hele objekter ned i *collections*.



WebAPI’et benytter et repository pattern, hvilket gør at databasen kan udskiftes uden det har betydning for systemet. Ved et eventudelt databaseskift, vil man ikke skulle ændre noget i den eksisterende kode, men lave en ny implementation af sit repository pattern så systemet kan tilgå den nye database, som i bund og grund munder ud i tankegangen om ”open-closed” princippet fra SOLID.

Databasen forbindes til resten af systemet gennem WebAPI’et og da WebAPI’et er et webinterface til resten af systemet spiller det ingen rolle for vores apps hvilken database der ligger bag ved WebAPI’et. Web API’et modtager og sender data i json format til og fra vores apps. Det har ingen betydning for databasen, da omdannelsen af json format til det format som databasen bruger, sker i WebAPI’et.

Ved oprettelse af nye *documents* i MongoDB skal der være et unikt ID som identificerer det enkelte *document* fra de andre. Dette ID benyttes som unikt ID på et patientkald. Denne ændring betød at PersonaleApp’en og PatientApp’en skulle tilpasses en lille smule for at kunne håndtere dette. *Det viser sig efterfølgende at man kunne have undgået denne ændring ved at indsætte vores eget unikke id i MongoDB dokumenterne og ikke benytte os af det auto genererede Id MongoDB ville sætte.*

Nedenstående diagram er lavet ud fra de data som vi ønsker at gøre brug af fra PatientApp og Klinisk Logistik samt deres relation. Disse data er valgt ud fra de funktioner PatientCare ønsker at gøre brug af.



### Sikring af stabilt netværk

Distribuerede antennesystemer (på DNU) optimere på at brugeren har stabilt netværk. Når man kommer ind på hospitalet nedsættes netforbindelsen på patientens mobiltelefon, så det ikke trækker alt på det offentlige netværk, det forsøger man at optimere på med distribuerede antennesystemer.

### Systemsikkerhed

Vi kobler ekstern device (patientens smartphone) på offentligt netværk hvordan sikrer vi at det ikke får systemet til at gå ned?

Hvorfor er der ikke lavet patientkald med årsag førhen? Teknologierne findes til det. Hvilke udfordringer er der med at implementere på offentlige hospitaler?

### WebAPI

I den videre udvikling af WebAPI’et er der umiddelbart tænkt over følgende punkter,

* Sikkerhed, for patientApp når patient logger ind, skal der laves en validering af om patienten er indlagt og om patienten må oprette patientkald i systemet. For personaleApp skal logges ind med brugernavn og password som gemmes et sted i databasen og valideres når personalet logger ind.

Umiddelbart vil det kunne løses ved at tilføje brugernavn, salt og password felt på et personale, hvor salt bliver brugt til at finde frem til det krypterede password. Det kunne også tænkes at der skulle interageres med et AD hvor personalet er oprettet i, men det skal der tages stilling til på et senere tidspunkt.

* Tildelt personale, når et patientkald bliver sendt ind i systemet, skal der kigges på hvilket personale der er primært og sendes til det primære personale først, hvor der efter et tids interval bliver sendt videre til andet personale, hvis patientkaldet ikke blev udført af det primære personale.
* Generelt kæves der en anden form for teknologi end det der er lavet på nuværende tidspunkt, men burde kunne løses ved at implementere SignalR[[5]](#footnote-5) teknologien. Kort sagt vil det gøre os i stand til lave push beskeder til patientApp og personaleApp i stedet for at skulle lave pull requests hver gang data skal opdateres. Uden at vide om det skal implementeres i WebAPI’et, er det taget med i disse overvejelser.

### AdminApp

Foruden en bedre implementeringen af de omtalte problemer, skal AdminApp fremadrettet videre udvikles til at kunne håndtere flere afdelinger på et sygehus og det skal være muligt at tilpasse de forskellige afdelinger uafhængigt af hinanden.

## Udfordringer

I dette afsnit beskrives de udfordringer bachelorgruppen er stødt ind i undervejs i projektet og hvordan de er blevet tacklet.

### Xamarin

En af de udfordringer der har været med at udvikle PatientApp’en har været at sætte sig ind i 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 også været en udfordring at få bygget og deployet et Xamarin.iOS projekt i Visual Studio. Da Visual Studio er en IDE, der kun som udgangspunkt kan køres på Windows, kræver det en virtual 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 computeren. Kan man ikke kobles på build serveren i Visual Studio, kan projektet ikke bygges. Og det har haft udfordringer fordi versionen af denne server skal passe i begge styresystemer – både i den virtuelle og ikke-virtuelle. Og da Xcode kom med den nye version 7,1 skulle build serveren opdateres, hvilket gav nogle problemer.

### Github

versionsstyring af Android

Android debugging i Android Studio

### Databaseskift

Opståede problemstillinger

* Lokalisering af patient
* Tilknytte personale
* Log ind
* Fjern patient fra system efter x dages passivitet
* Skub opg. videre fra primær til sekundær SP
* Personale udfører opgave ved et uheld

Overvejelser:

* Den nærmeste sygeplejerske skal modtage kaldet og ikke andre? (Lokalisering af nærmeste)
* Alle sygeplejersker på afdelingen skal modtage alle kald fra patienter?
* Det skal være synligt for andre sygeplejerske at et kald er ”taget” af en kollega?
* Der er knyttet primær og sekundær sygeplejersker på en patient og kun de to skal modtage patientens kald?
* Det skal være muligt at ”tage” opgaver før de udføres?
* Der er behov for at man kun klikker færrest mulige gange for at udføre en opgave?
* Det skal være muligt for patienten at fortryde et kald der er sendt afsted?
* Det skal være muligt for sygeplejersken at oprette et kald for en patient via sygeplejerskens arbejdstelefon?
* En sygeplejerske skal kunne ”tage” flere opgaver inden hun udfører dem

### Lokalisering af patient

Løsninger:

* + Aktiv og passiv RFID
  + GPS og WIFI er ikke præcist nok på et hospital med mange etager

Valgt:

* + WIFI er den valgte løsning fordi xxx

Sikring af patientfølsomme oplysninger (samtykke erklæring) sundhed.dk? Ikke set før

### Patientens rettigheder

Patientens rettigheder er minimale. Det er op til den enkelte afdeling hvilke valgmuligheder patienten skal have for at sende et patientkald afsted gennem PatientCare. Som udgangspunkt er det ønskværdigt at patienterne ikke skal kunne have mulighed for at modificere på det valgte kald, således at det er forudbestemt hvad der er muligheder for at modtage på personalets side. Det betyder at patienten som udgangspunkt ikke kan tilføje kommentar til det valgte kald.

### WebAPI

Et af de helt store problemer for WebAPI’et har været skiftet fra at bruge *Linq to SQL* klasser til at tilgå databasen, og til at bruge MongoDB frameworket til at tilgå databasen. Implementeringen for at tilgå mongoDB, var ukendt for gruppen og krævede en del tid at sætte sig ind i mongoDB frameworket.

Man prøver altid at få ting til at ligne noget man kender så det kan relateres til og give en bedre forståelse, og i dette tilfælde resulterede det i nogle mærkelige workarounds som set i bagklogskabens klare lys aldrig skulle have været lavet på den måde. Fx når man henter alle aktive patientkald ud fra databasen, bliver de leveret som en lang json formatteret tekststreg i stedet for at benytte sig af json objekter og så sende dem ud af WebAPI’et. Eller at lave en manuel mapping hvor mongo c# driveren har en udmærket bson deserializer som tager et bsondocument og laver det om til et c# object som json.net deserializeren (som er standard i ASP.NET WebAPI) bedre kan håndtere.

MongoDB benytter sig af et dokument id (\_id) til at identificere alle sine dokumenter med et unikt nummer som altid bliver sat om man så selv gør det eller ej. Hvis man manuelt tilføjer id’et til dokumentet inden det bliver sendt ind i databasen, er det nemmere at styre dette id’s formatet. Hvis man lader mongoDB selv stå for at tilføje et id til dokumentet for man et id som er af typen objectId som er en mongoDB type. Når man er forholdsvis ny til mongoDB hvor fokus ikke ligger på at omgå databasen typer, men om at forstå hvilke funktioner der skal bruges for at kommunikere med databasen, så er det dybt frustrerende at der lige pludselig er en type som man ikke kender til. Det samme gør sig gældende med formateringen af en c# datetime til bsondocument som bliver lavet om til en ISODate inde i mongoDB dokumentet. Igen dybt frustrerende at arbejde med, men her efterfølgende ikke det store problem, men sådan er det med mange ting. Det er svært, indtil det er nemt.

### AdminApp

På nuværende tidspunkt er der ikke udviklet nogen form for sikkerhed ind over administrations modulet. Det skal ikke være muligt for enhver at logge sig ind på administrations modulet og derved kunne slette eller oprette nye valgmuligheder. Derfor skal der laves en form for sikkerheds anordning så det kun er administratorer af PatientCare systemet der kan tilpasse systemet.

Et andet problem er at AdminApp forsøger at benytte mongoDB på en måde som databasen ikke er bygget til. Da databasen er en mongoDB og ikke har relationer mellem de forskellige collections, og da vi har valgt at opbygge databasen efter den relationelle tankegang, gør det bl.a. at en kategori på en type ligger som et id og ikke navnet på kategorien. Det gør at AdminApp bliver nødt til at hente informationerne fra kategori collectionen på mongoDB ud inden de kan vises på siden. Hvis så en kategori bliver slettet, er der altså ikke et id som hører til det id som en type har som sin kategori og programmet fejler, fordi det ikke kan hente den efterspurgte data da den data ikke findes mere. I en relationel database, vil man kunne lave *On Cascade delete* på en foregin key relation, men i mongoDB, skal dette gøres manuelt og det er ikke blevet implementeret på WebAPI’et.

# Konklusion

Hvad har vi opnået af projektet? jf. læringsmål.

Konkludere på processen

* Samarbejde ml. IKT og ST
* Vi bygger broer

# Perspektivering

Hvordan kan produktet videreudvikles med henblik på en implementering i praksis.

Integrering med andre eksisterende systemer fx kaldesnor, Cetrea osv.

Forbedringer i AndroidApp:  
Vi åbner og lukker den lokale database meget ofte  
Brug i stedet en DTO (Singleton)  
Push i stedet for pull

Nice to have:  
Slet mine kald  
Slet udførte kald

Personale skal kunne oprette portøropgaver på PersonaleApp i stedet for at bruge en PC på kontoret. Så kan personalet oprette opgaver inde ved patienten. Dermed undgår man også at personalet glemmer det.

Tjek lokation af

# 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. 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.**

**6. Bilag Observation i Randers. 2/10-15.**

**7. Statens Institut for Medicinsk Teknologivurdering. *Medicinsk teknologivurdering Hvorfor? Hvad? Hvornår? Hvordan?* 2000.**

1. https://da.systematic.com/healthcare/products/columna-service-logistics/apps/task-management/ [↑](#footnote-ref-1)
2. http://www.idc.com/prodserv/smartphone-os-market-share.jsp [↑](#footnote-ref-2)
3. https://bitbucket.org/twincoders/sqlite-net-extensions [↑](#footnote-ref-3)
4. https://github.com/praeclarum/sqlite-net [↑](#footnote-ref-4)
5. [↑](#footnote-ref-5)