DTU Compute
Institut for Matematik og Computer Science

Logistisk håndtering af udstyr

Værktøj til at forudsige behov og fordeling af ePOS terminaler i real-time

Diplom Afgangsprojekt

DTU Compute

07. december 2015

I samarbejde med Thomas Cook Airlines Scandinavia A/S

Udarbejdet af:

DTU Compute
Institut for Matematik og Computer Science

Matias D. Bennike

mbennike@gmail.com

Studienummer: s113600

Institut:

DTU Compute – Institut for Matematik og Computer Science

Richard Petersens Plads
Bygning 324
2800 Kgs. Lyngby

Vejleder: Stig Høgh, shog@dtu.dk

Ekstern vejleder: Morten Østergaard Nielsen, mon@thomascook.dk

Indholdsfortegnelse

Abstract

Opgavebeskrivelse

Opgaven går ud på at udtænke og udvikle et værktøj til at forudsige hvornår og hvor der kommer der at mangle ePOS[1] terminaler til TCAS[2] flyvninger. Opgaven giver frihed til også at beskrive optimeringsprocesser omkring ePOS håndtering.

Baggrund

Der foretages hver uge ca. 200 flyvninger, hvoraf der på nogle skal være ePOS terminaler til stede og andre hvor der ikke skal.

Antallet af destinationer kan varierer fra uge til uge og afhængigt af om det er vinter eller sommer, men en typisk uge kan have omkring 40 destinationer, hvoraf en mindre del håndterer ePOS terminaler.

ePOS er den betalingsterminal som TCAS anvender til at modtage betalinger ombord på flyene. Betalinger vedrører både Toldfri varer købt via "Airshoppen"[3] samt drikkevarer, måltider/snacks og impulskøb ombord. ePOS terminaler bliver lagt ombord TCAS fly hver morgen fra hoved baserne – primært i Skandinavien – under vinterprogrammet også fra Tenerife / Las Palmas.

Upload af data på ePOS og batteriopladning og skift varetages af toldlagerne som er op til 4 forskellige catering partnere. Dette er betinget af byen/landet.

TCAS trafikmønster er ret kompliceret og ikke identisk fra dag til dag eller sågar fra uge til uge. Opgaven må nødvendigvis inkludere information om trafikmønsteret fra SABRE/NETLINE[4].

Opgaven er afgrænset til den fysiske håndtering af terminaler til TCAS produktion og skal ikke inkludere de IT systemer som håndterer betalingsdata til og fra ePOS.

[1] ePOS: electronic Point Of Sale (håndholdt kasseapparat).
[2] TCAS: Thomas Cook Airlines Scandinavia.
[3] Airshoppen: Webshop der giver mulighed for at bestille toldfrie varer hjememfra, der udleveres ombord på flyet.
[4] SABRE/NETLINE: 3. parts programmer der indeholder al information om flyvninger; til/fra og tider samt mere.

Indledning

Projektet er udført i samarbejde med IT Project Manager Monica Lindhardt hos Thomas Cook Airlines Scandinavia A/S. Virksomheden beskæftiger sig primært med charterflyvninger og sælger primært flysæder til virksomhederne Spies (Danmark), Ving(Sverige og Norge) samt Tjäreborg (Finland). IT-afdelingen beskæftiger sig primært med udvikling af ny software, den daglige drift og opbevaring af den information der skal være tilgængelig for at denne kan lade sig gøre.

Da firmaet beskæftiger sig med flytrafik er der mange krav fra myndighederne til den information der findes om flyvninger, personale og passagerer. I forbindelse med dette projekt betyder det at al information om flyvninger er tilgængeligt og relativt nemt kan tilgås.

Et af de vigtigste områder for virksomheden er salg af toldfrie varer på flyvninger, i den forbindelse er vedligeholdelsen af hjemmesiden ”Airshoppen” også et vigtigt område for IT-afdelingen. På hjemmesiden kan man bestille toldfrie varer man får udleveret på hjem- eller ud-turen (eller på begge, hvis man ønsker dette). Der er derudover mulighed for at købe et mindre udvalg varer ombord på flyvningerne, samt at købe forfriskninger.

Salg af toldfrie varer samt salg af varer ombord på flyet udgør en af de største indtjeningskilder for virksomheden, så det er afgørende at det altid er muligt at gennemføre salg.

Til dette formål bruger virksomheden såkaldte ePOS terminaler (electronic Point Of Sale) der kan bruges til at scanne varer (for at få information om pris samt lave lagerstyring) samt gennemføre betalinger med kreditkort og kontanter, ombord på flyene.

Terminalerne fungere udelukkende off-line ombord på flyene, så data om transaktioner på terminalerne skal uploades til en database i virksomheden, så betalingerne kan blive gennemført.

Denne rapport vil belyse løsningen af problemstillingen angivet i abstractet. I det efterfølgende vil jeg starte med at uddybe problemstillingen, analysere de udfordringer der har været i løbet af processen samt beskrive såvel anvendte teknologier samt anvendte løsninger.

Jeg har i løbet af projektet fulgt de ”best practices” der anvendes i virksomheden.

Uddybning af problemstilling

Som tidligere beskrevet omhandler opgaven håndtering af ePOS terminaler. TCAS foretager flyvninger til en lang række lufthavne, men for denne opgave er det kun følgende lufthavne der har interesse, da det kun er disse lufthavne der har direkte interaktion med ePOS terminalerne.

Herunder fremgår både navnet på lufthavnen, samt den tilhørende IATA[5]-kode, der er en unik kode på 3 bogstaver, der angiver en lufthavn. Ud over IATA-koder bruges også såkaldte ICAO[6]-koder,

[5] International Air Transport Association
[6] International Civil Aviation Organisation

begge dele er tilgængeligt i det byggede system, men i projektet anvender jeg udelukkende IATA koder, da disse er mest brugt i virksomheden.

Listen af lufthavne der håndterer ePOS terminaler er følgende:

- Stockholm, Arlanda – ARN
- Göteborg – GOT
- Malmö – MMX
- København – CPH
- Billund – BLL
- Oslo – OSL
- Bergen – BGL
- Helsinki – HEL
- Gran Canaria – LPA
- Tenerife Syd – TFS

Følgende er ikke lufthavne, men steder hvor TCAS har faciliteter som også bruges i forbindelse med ePOS terminaler.

- Lund, Sverige
    - Her er varehus til Airshoppen der forsyner flyvninger med varer.
- Thomas Cook Airlines Scandinavia, i Kastrup Lufthavn

Det er på disse lokationer at udbytning og håndtering af terminaler sker, så der altid er terminaler klar til brug ombord på flyene.

Tidligere har det været et problem at man i en af disse lufthavne, finder ud af man mangler ePOS terminaler. I dette tilfælde har man mulighed for at få sendt ekstra terminaler fra en af de andre lufthavne, dog er det typisk for sent, hvis man først finder ud af dette samme dag.

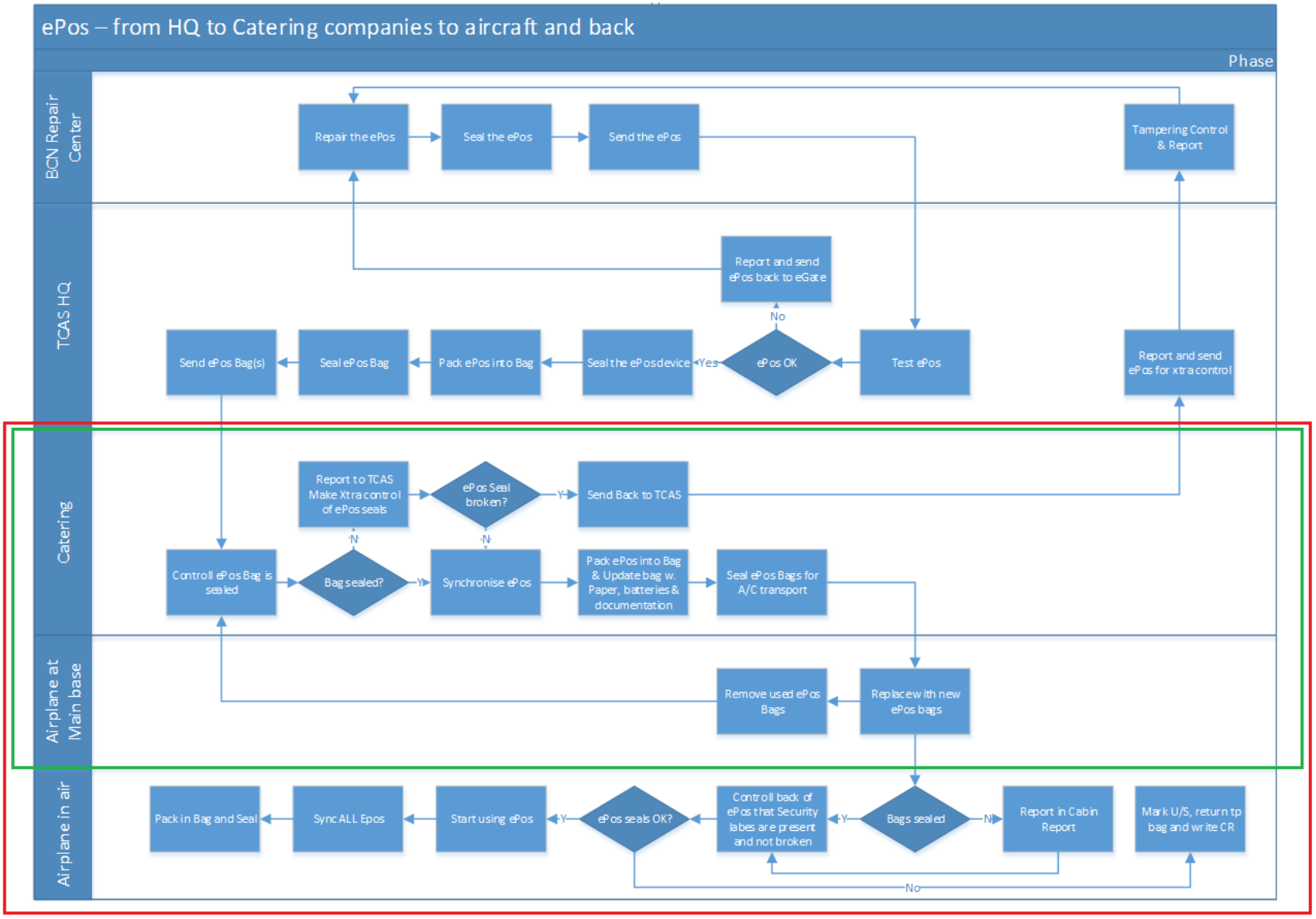
Formålet med dette værktøj er altså at give den ePOS ansvarlige på kontoret i Kastrup Lufthavn, Monica Lindhardt, mulighed for at få en approksimation over hvor der kan komme til at mangle terminaler for så at kunne handle på dette i så god tid, at der ikke opstår problemer.

Håndtering af ePOS terminaler

Når et fly lander i en af de nævnte lufthavne, bliver der sørget for at de brugte terminaler bliver lastet af flyet og overført til et kontor i den pågældende lufthavn.

Derudover bliver der også synkroniseret lagerstyring, så man ved hvor meget der er blevet solgt ombord på flyene.

På figuren herunder ses alle de processer som involverer ePOS terminalerne. Jeg har markeret det område der har direkte relation til dette projekt med en grøn kasse og det område der kunne tænkes at være relevant i forhold til en alternativ proces og håndtering af epos terminalerne, er markeret med rød.



Figur 1 - Flowdiagram over ePOS håndtering

Disse procedurer bliver kun udført i de lufthavne der varetager ePOS terminaler, disse lufthavne er nævnt i afsnittet ”Udbydning af problemstilling”.

Det fremgår af tegningen at hele processen med at håndtere ePOS terminaler kan være ganske kompleks. Den gøres ikke mindre kompleks af, at der er store krav omkring sikkerhed og sporbarhed i forbindelse med håndtering af terminalerne. I den øverste bane ses både ”Seal the epos” og ”Tampering Control & Report” disse to referere til at ePOS terminalerne skal være plomberede og de skal løbende undersøges for at fastslå om der er blevet ændret ved dem.

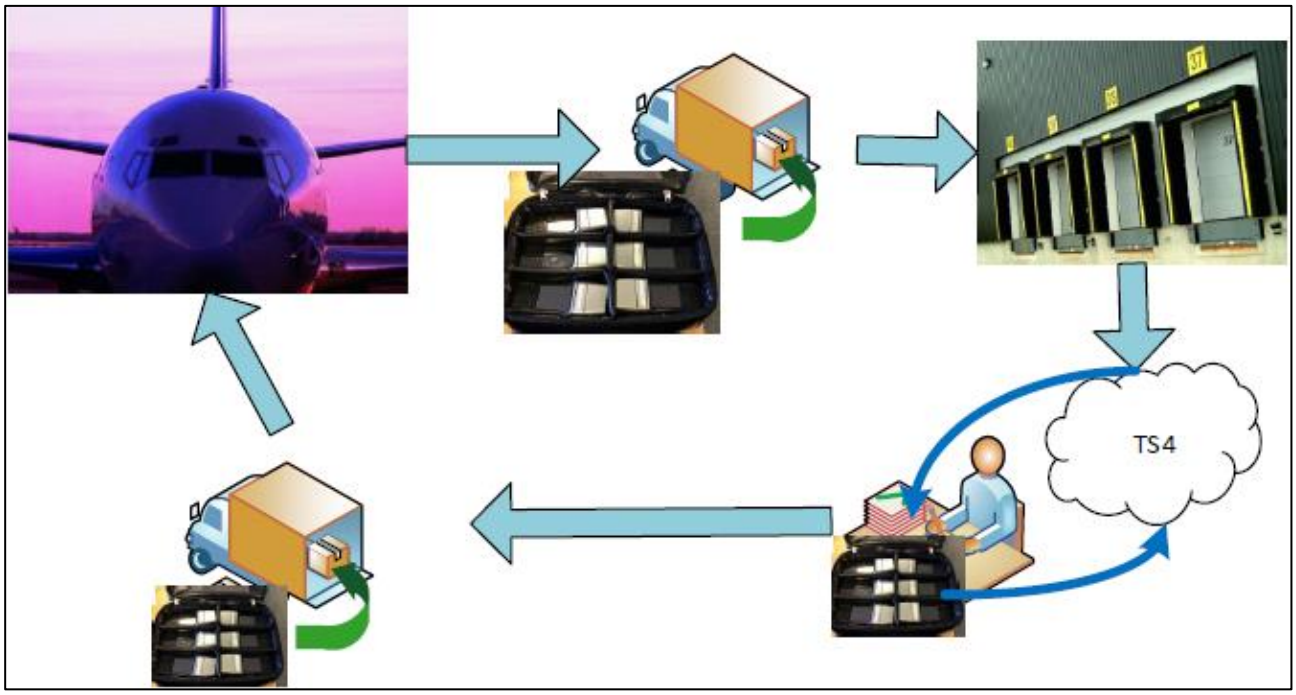
I anden bane bliver ePOS terminalerne også forseglet i tasker – ”Seal ePos Bag”, der er tasker der kan rumme 6 ePOS terminaler af gangen. I tredje bane, som omhandler de firmaer der står for håndteringen af ePOS terminalerne, bliver det sikret at alting er forseglet, data synkroniseres og terminaler gøres klar til at kunne blive sendt ombord på et fly. I det tilfælde at en ePOS taske eller ePOS terminal ikke er forseglet, bliver denne sendt til TCAS (Kastrup Lufthavn) hvor det bliver undersøgt om der er forsøgt eller er blevet manipuleret med terminalerne.

Det er også i dette skridt at data bliver uploadet til den betalingsgateway der bruges, eGate Solutions TS4[7]. Dette er også grunden til at terminalerne igennem disse led skal have checket forseglingen, da de bruges til at håndtere betalinger og betalingsdata.

Jeg vil til sidst i opgaven argumentere for alternative håndteringsmodeller, som vil begrænse sig til området afgrænset af den røde kasse.

Herunder ses et diagram der viser præcis de processer der relaterer til denne opgave, altså de skridt der søges bedre og mere pålidelig data omkring.

[7] Et firma der hovedsageligt laver betalingsløsninger til luftfartsselskaber http://www.egate-solutions.com/home



Figur 2 - Håndtering af ePOS terminaler

Hvis vi starter med flyet, øverst til venstre, og går mod urets retning er først skridt at ePOS terminaler bliver lastet af flyet. Herfra bliver de fragtet til det firma TCAS har aftale med om håndtering af ePOS terminaler. Dette vil være det samme firma der står for håndtering af madvarer ombord på flyet.

Herfra bliver terminalerne undersøgt for om plombering er intakt og data uploades, bl.a. til TS4 der håndterer betalinger. Her bliver batterier også sat til opladning og ePOS terminalerne bliver pakket i en taske med friske batterier. Herefter bliver ePOS terminalerne kørt ud til den næste afgang, enten 1 eller 2 tasker – dvs. 6 eller 12 ePOS terminaler, afhængigt af flyets størrelse.

I et forsøg på at indføre større sporbarhed omkring hvor terminalerne befinder sig, har man på hver terminal påsat en passiv RFID[8] chip. Denne chip indeholder et unikt ID-nummer, der gør det muligt at identificere en terminal med en RFID læser.

Når terminalerne befinder sig på kontoret er den nuværende procedure at terminalerne skal lægges i tasker der indeholder 6 terminaler og at taskerne skal lægges på nogle specielt designede hylder. Disse hylder indeholder RFID læsere der i teorien gør det muligt at aflæse præcis hvilke terminaler der befinder sig på dem. I realiteten har det dog vist sig at disse RDIF læsere ofte ikke kan finde og aflæse alle de terminaler der ligger på hylderne.

Specifikt har det vist sig at manglende data fra RFID læserne ikke altid er en fejl i læserne, men simpelthen fordi procedurerne ikke blive overholdt og terminalerne aldrig kommer i nærheden af RFID læserne.

[8] Radio Frequency Identification, indeholder hukommelse der kan aflæses trådløst.

Håndteringen af terminalerne bliver varetaget de samme firmaer der står for at laste og aflaste madvarer ombord på flyet, og er således ikke håndteret af TCAS personale. Dette har også medført endnu et niveau af usikkerhed, da de opstillede procedurer ikke altid bliver udført og det har vist sig besværligt at få rettet op på dette.

Grundet denne usikkerhed har man ønsket en løsning der bruger anden information til at beregne og forudsige hvor mange terminaler en given lufthavn skal have, for at kunne sikre der er nok til alle afgange.

Det er i forbindelse med den problemstilling, at dette projekt blev udtænkt.

For at få et præcist tal over hvor mange terminaler der ligger på de forskellige lufthavne, bliver der hver tirsdag lavet en optælling hos de forskellige lufthavne, denne information sender de til virksomheden, hvor den bliver samlet sammen, denne optælling er en direkte konsekvens af usikkerhed omkring RFID aflæsning.

Denne information bliver brugt i programmet til at skabe en korrekt baseline, der er med til at øge præcisionen.

Generelt om udvikling i virksomheden

Programmet er skrevet i C#, i Visual Studio Enterprise 2015 og databasen er en Microsoft SQL database.

I forbindelse med udviklingen er der brugt Team Foundation Server (TFS) som repository.

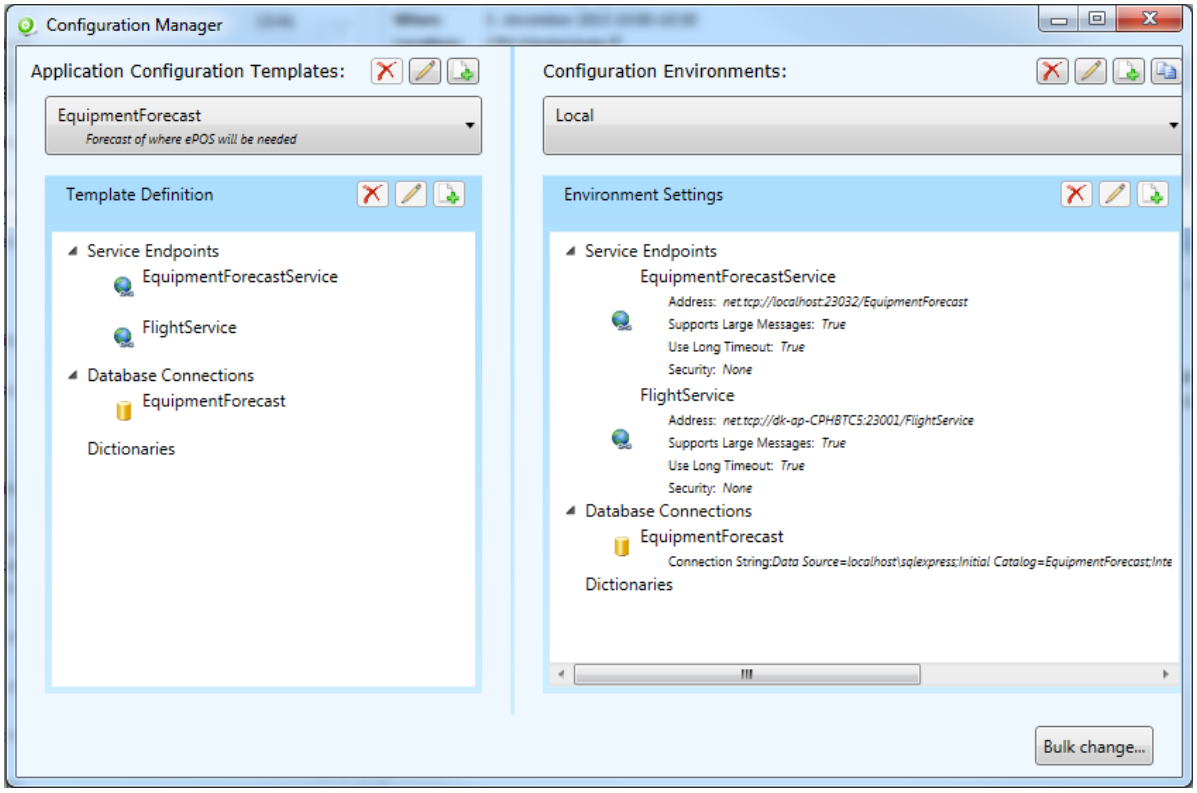
Derudover gør programmet brug af en række API'er og redskaber der enten er udviklet i virksomheden eller som er indhentet udefra.

Her kan specielt nævnes et værktøj kaldet "Configuration Manager", der er vist et billede af interfacet til dette værktøj herunder, på efterfølgende figur.

Det ses på dette billede hvilke indstillinger jeg har lavet til miljøet "Local", skridtet før jeg tester programmet på et development miljø.

Det fremgår af billedet at endpoints'ne på såvel FlightService som på EquipmentForecastService er indtastet her, så de er tilgængelige for programmet. EquipmentForecastService er den service, der varetager håndteringen af data til og fra frontend delen af programmet.

Der er ligeledes lavet en reference til en database, hvor data bliver gemt.



Figur 3 - Billede af værktøjet "Configuration Manager"

Dette værktøj har til formål at gøre det nemt for udviklerene at teste og gøre udviklet software tilgængeligt på hhv:

- Local
  - Bruges ikke altid, men kan være en fordel når der tidligt i processen skal testes specielt data håndtering
- Development server
  - Udvikler kan teste produktet og lave fejlfinding, på et server miljø
- Staging server
  - Slutbrugere kan teste produktet uden at ændre ved produktionsdata
- Production server
  - Produktet sættes i produktion og kan nu bruges af alle slutbrugere

Dette værktøj gør det nemt at ændre forskellige variable i koden, der er afhængige af hvilket miljø det kører på. Dette kan inkludere brug af forskellige databaser, forskellige netværksmapper til eksport eller import af data samt generelt mulighed for at angive forskellige variable på en nem og overskuelig måde. Rent teknisk kræver dette at de ønskede variable hentes fra ”Configuration Manager”-servicen og derudover en angivelse i den ”app.config” fil der findes i ethvert ”Windows Application”-projekt i C# omkring hvor på netværket denne information kan findes (der hvor ”Configuration Manager” lytter på forespørgsler).

Denne måde at opdele udviklingen på, i forskellige miljøer igennem udviklingen, har flere fordele; ved at kunne sikre at en opdatering til eksisterende software eller nyt software fungere korrekt på et development server miljø, før det sættes i produktion, er med til at sikre kvalitet.

Før produktet sættes i produktion kan det derudover blive testet af slutbrugere, både for at give disse træning i brug af ny funktionalitet, men også for at sikre sig at produktet lever op til de forventninger der er opstillet.

Generelt om programmet og data

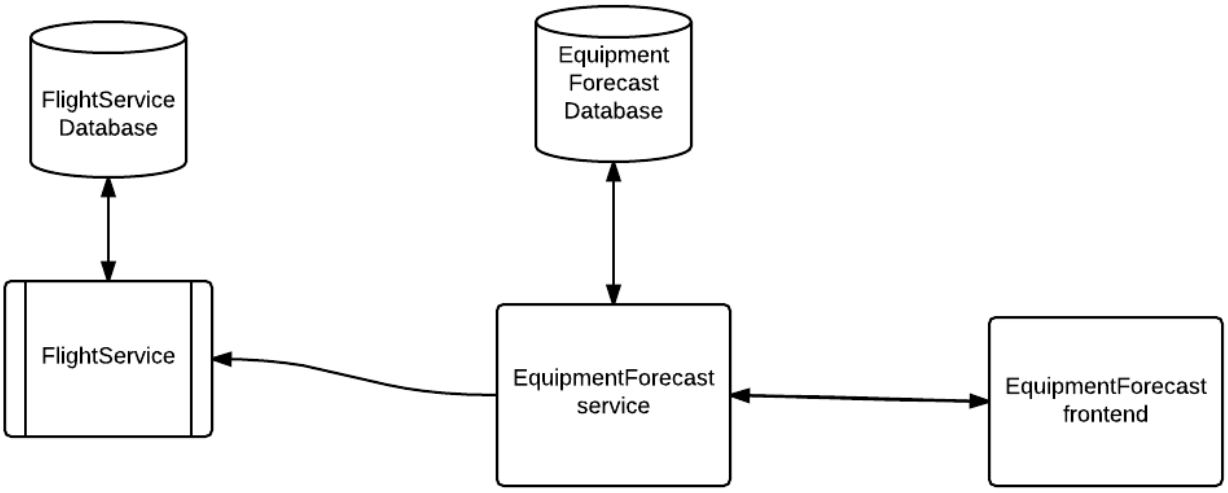
Programmet har fået navnet EquipmentForecast og vil i denne rapport blive benævnt sådan. Jeg skelner mellem EquipmentForecast Service og EquipmentForecast Frontend. Service'en er backend delen, der står for dataopsamling samt datahåndtering til Frontend delen, som er det brugerinterface brugere af programmet interagerer med.

Jeg har i designet af hhv. backend- og frontend-delen valgt at lade data være så generel som muligt i backend-delen, for så i frontend-delen at lave de nødvendige sorteringer og generel behandling af data. Rationalet bag dette er, at specialiseret eller begrænset data i backend-delen (som er den eneste data supplier til frontend-delen) vil sætte begrænsninger på hvad der kan vises i frontend-delen.

Dette kan betyde at programmet er en smule mere krævende for brugerens computer at afvikle, men gør udviklingsarbejdet nemmere og hurtigere, idet der kun skal laves ændringer i frontend-delen for at vise mere data, eller for at vise data på en anden måde.

Programmets opbygning

Programmet er opbygget i tre dele; den ene del er en service der altid er aktiv, som står for både dataindsamling omkring flyafgange og datahåndteringen til klienten, den anden del er den klient som brugeren kan åbne, for at få adgang til den ønskede data. Til sidst har servicen sin egen database tilsluttet og kan levere den relevante data til brugeren, uden at skulle lave kald mod de større og mere generelle services, der udbyder data der bruges i virksomheden.



Figur 4 - Relation mellem data udbydere og frontend

Ovenstående figur er en high-level repræsentation af forholdet. Til venstre ses den service der hedder FlightService, denne kan give systemer adgang til en lang række data i forbindelse med flyvninger udført af virksomheden. Denne har en tilhørende database, som kun denne service kan tilgå der ligger på samme fysiske server, for at minimere belastning på netværket.

I midten ses EquipmentForecast servicen, samt dennes database. Når servicen startes op, laver den en forespørgsel mod FlightService for at få en liste over alle flyvninger i den nærmeste tid. Denne information skrives til databasen, for at have en lokal kopi af data, der skal bruges i forbindelse med programmet.

Figuren er et billede af den relation der findes i det øjeblik servicen startes op, efterfølgende er der ikke længere behov for at lave forespørgsler til FlightService, for at få information omkring flyvninger.

Den anden information der skal bruges i forbindelse med programmet, er de ugentlige optællinger af ePOS terminaler rundt omkring i de relevante lufthavne. Her giver programmet, EquipmentForecast frontend, til højre, mulighed for at indtaste information om disse og gemme denne information i den database der tilhører EquipmentForecast servicen. På denne måde vil det både være muligt for flere forskellige brugere at tilgå det samme data og der kan laves input validering i interfacet for at sikre korrekt indtastning af data.

Dette viste sig nødvendigt i en tidligere version hvor programmet kunne tage imod et excel ark med den ønskede information, her viste det sig at der ofte var stor inkonsistens omkring hvordan specielt datoer indtastes af de forskellige lufthavns kontorer, heraf nødvendigheden for input validering. Dette medførte den nuværende løsning, hvor en bruger manuelt kan indtaste informationen i programmet.

På sigt vil det også være muligt at give lufthavns kontorene adgang til programmet, for på den måde at lade dem selv indtaste data. Indtil videre bliver dette dog håndteret hos TCAS.

Når EquipmentForecast servicen er startet op og har modtaget information om kommende flyvninger, skal den stadig være i stand til at modtage information om nye flyvninger eller ændringer i eksisterende flyvninger. For at undgå en situation hvor servicen bliver nødt til at lave hyppige kald mod FlightService (polling), laver programmet efterfølgende et subscription kald til FlightService, hvori det kan abonnere på forskellige typer beskeder. Dette vil blive behandlet mere uddybende i afsnittet "Event Driven Messaging".

Programmets database

Databasen der er tilknyttet programmet er en Microsoft SQL server, den består af to tabeller; denne ene indeholder information om flyvninger (FlightInfo) og den anden indeholder information om ePOS beholdningen i de forskellige lufthavne (AirportInventory).

De to tabeller er vist efterfølgende:

FlightInfo

Column Name	Data Type	Allow Nulls
LegId	varchar(50)	☐
ACReg	nchar(10)	☐
EposRequired	int	☐
BatteriesRequired	int	☐
DepartureAirportIata	varchar(3)	☐
ArrivalAirportIata	varchar(3)	☐
UtcDepartureDate	datetime	☐
UtcArrivalDate	datetime	☐
LocalDepartureDate	datetime	☐
LocalArrivalDate	datetime	☐
FlightType	nchar(10)	☐
AircraftType	nchar(10)	☑
FlightIdentifier	nchar(10)	☐
FlightNumber	nchar(10)	☐
		☐

AirportInventory

Column Name	Data Type	Allow Nulls
AirportIATA	varchar(3)	☐
TimeChecked	datetime	☐
EposInWarehouse	int	☐
Defect	int	☑
InAir	int	☑
NeededNext3Days	int	☑
MHSWorking	int	☑
MHSNotWorking	int	☑
Batteries	int	☑
EposHandles	int	☑
SdCards	int	☑
		☐

Figur 5 - SQL tabeller

I tabellen FlightInfo er LegId en unik nøgle for hver flyvning, der kan bruges til at referere til en specifik flyvning, denne bruges hovedsageligt i forbindelse med ændringer i data modtaget fra FlightService som en besked (event driven messaging).

Denne tabel indeholder information omkring antallet af påkrævede ePOS terminaler, som bliver udregnet baseret på flytypen af EquipmentForecast servicen og bliver efterfølgende skrevet til databasen. Grunden til at der er forskelligt behov af terminaler er såvel flyets størrelse – enten 6

eller 12 terminaler er påkrævet ombord, men også at nogle flyvninger slet ikke har behov for terminaler, da de af den ene eller anden grund ikke flyver med passagerer.

Denne information gør det muligt at bruge et FlightInfo objekt som en ændring i antallet af ePOS terminaler på bestemte tidspunkter for både afgangs og ankomsts lufthavnen, altså; når et fly der har behov for f.eks. 6 terminaler ombord flyver fra en lufthavn, falder antallet af ePOS terminaler i lufthavnen med 6. Ligeledes vil antallet af ePOS terminaler stige i ankomstlufthavnen med 6, når flyet lander, såfremt det er en af de lufthavne der håndterer ePOS terminaler.

Tabellen indeholder både UTC tider såvel som lokale tider, til dette formål er det gunstigt at bruge UTC tider da man på den måde kommer ud over eventuelle problemer med beregning af tidszoner og problemer med sommertid og vintertid på forskellige lokationer i verden, et aspekt der hurtigt kunne give grund til fejl.

AirportInventory tabellen indeholder al den data man hidtil har indsamlet fra lufthavnskontorene. Det er på nuværende tidspunkt ikke til at sige hvorvidt al informationen er nødvendig at have, da det ikke er alle værdierne der bruges til beregningen i programmet, men under alle omstændigheder er det fornuftigt at bevare så meget data som muligt i tilfælde af at det på et senere tidspunkt bliver relevant, eller det viser sig at det er muligt at forbedre præcisionen ved brug af dette ekstra data.

Service Oriented Architecture (SOA)

SOA er en måde at tænke arkitektur hvori forskellige services og programmer udveksler data over et netværk. Hver del, eller service, i SOA har en bestemt og typisk begrænset funktionalitet der kan benyttes af andre services, uden indblanding.

En af de største fordele ved brug af services på denne måde, er at programmer der bruger en given service ikke behøver information om hvordan den pågældende service fungerer, men blot behøver at vide hvordan man interagerer med servicen. Derudover bør en sådan service have et klart og veldefineret ansvarsområde, hvilket er med til at øge både hvor nemme de er at vedligeholde, men også for at beholde det overblik som SOA kan give.

I den forbindelse er en af de hyppigste faldgrubber at services langsomt vokser sig større og større i takt med at der opstår nye behov. Her er det vigtigt at insistere på at services holdes små og kompakte, da både sporbarhed og vedligeholdelse lider meget ved vokseværk i services.

Om SOA i virksomheden

I virksomheden begyndte man i midten af 2015 at lægge mere vægt på hvordan de forskellige services og programmer der var blevet udviklet i virksomheden kommunikerede indbyrdes samt hvordan de overordnet blev opbygget.

Resultatet af denne overvejelse blev at man i højere grad ville opdele eksisterende og fremtidige services i så små moduler som muligt. Tanken har været at man kunne skabe bedre overblik over

hvilke funktioner de forskellige services udførte samt gøre vedligeholdelse og udbygning nemmere.

Figur 4 viser ganske udmærket resultatet af denne proces, hvor fokus er lagt på at have en længere række services der hver især har et specialiseret formål. Dette gør det nemmere at vedligeholde de forskellige services, da kodebasen i hver service typisk er mindre og man undgår at hver service bliver unødig kompleks.

Den lave kompleksitet skal i teorien være med til at skabe og vedligeholde et godt overblik, men kræver i praksis en del retningslinjer og fokus for at undgå at individuelle services får deres ansvarsområde udvidet på en uhensigtsmæssig måde. Med andre ord skal der hver gang man laver en tilføjelse til en service, overvejes om det er hensigtsmæssigt for denne specifikke service at varetage denne ekstra funktionalitet eller om der skal oprettes en ny service til dette.

SOA i projektet

For dette projekt betyder det at der både gøres brug af service provideren FlightService samt at projektet indeholder en service, EquipmentForecast service, der både agerer service provider og service consumer.

Denne gør det muligt at gøre brug af programmet uanset hvor man befinder sig, da det relevante data er vedligeholdt for sig selv. Det betyder også at det vil være muligt at gøre brug af dette data i andre sammenhænge end blot dette projekt, hvis det på et tidspunkt skulle vise sig gunstigt.

De forskellige services gør en række metoder tilgængeligt for andre services/programmer, disse kan enten tage imod nogle parametre for at kunne behandle forespørgslen, tage imod nogle (valgfrie) parametre eller ikke tage imod parametre. Derudover kan disse services returnere data til det program der forespørger data – enten det forespurgte data eller, hvis man vælger dette, en fejlbesked om at handlingen ikke kunne lade sig gøre.

I projektet har jeg derfor opstillet en række metoder der kan tilgås af frontend-delen af programmet. Dette inkluderer følgende metoder:

- CreateEposInventory
    - Gør det muligt at sende data omkring ePOS beholdninger i lufthavne, denne data bliver skrevet til databasen af EquipmentForecast servicen. Denne metode har en overload, så der både kan sendes enkelte objekter (for en enkelt lufthavn) eller en liste af objekter, for flere lufthavne.
- DeleteEposInventory
    - Gør det muligt at fjerne information om beholdningen i en lufthavn. Dette er gavnligt i tilfælde af at forkerte værdier er indtastet.
- GetEposInventory
    - Gør det muligt at modtage en liste over ePOS beholdningerne i de forskellige lufthavne, ud fra den data der tidligere er indtastet af brugeren
- GetFlights

- Gør det muligt at modtage en liste med flyvninger i en angivet periode, der kan vælges af brugeren i frontend-delen af programmet

Disse metoder gør det muligt for frontend-delen af programmet at snakke med backend-delen (EquipmentForecast-servicen) og derved både modtage og sende information imellem de to.

Event Driven Messaging (EDM)

Event driven messaging er en måde at opstille en infrastruktur på, der sigter efter at optimere dataudveksling på netværket, ved at bruge EDM kan man helt afskaffe brugen af polling på databaser og services.

EDM fungerer ud fra et publisher/subscriber forhold, typisk med en eller anden form for service bus eller middle layer der står for at håndtere hvilke services der abonnerer på hvilke typer data.

I praksis fungerer dette ved at en publisher sender beskeder til sine subscribers, når der sker en handling (event), som en eller flere har subscribet på. Det kan være at der er indhentet ny information, ændring i eksisterende information eller sletning af information.

Om EDM i virksomheden

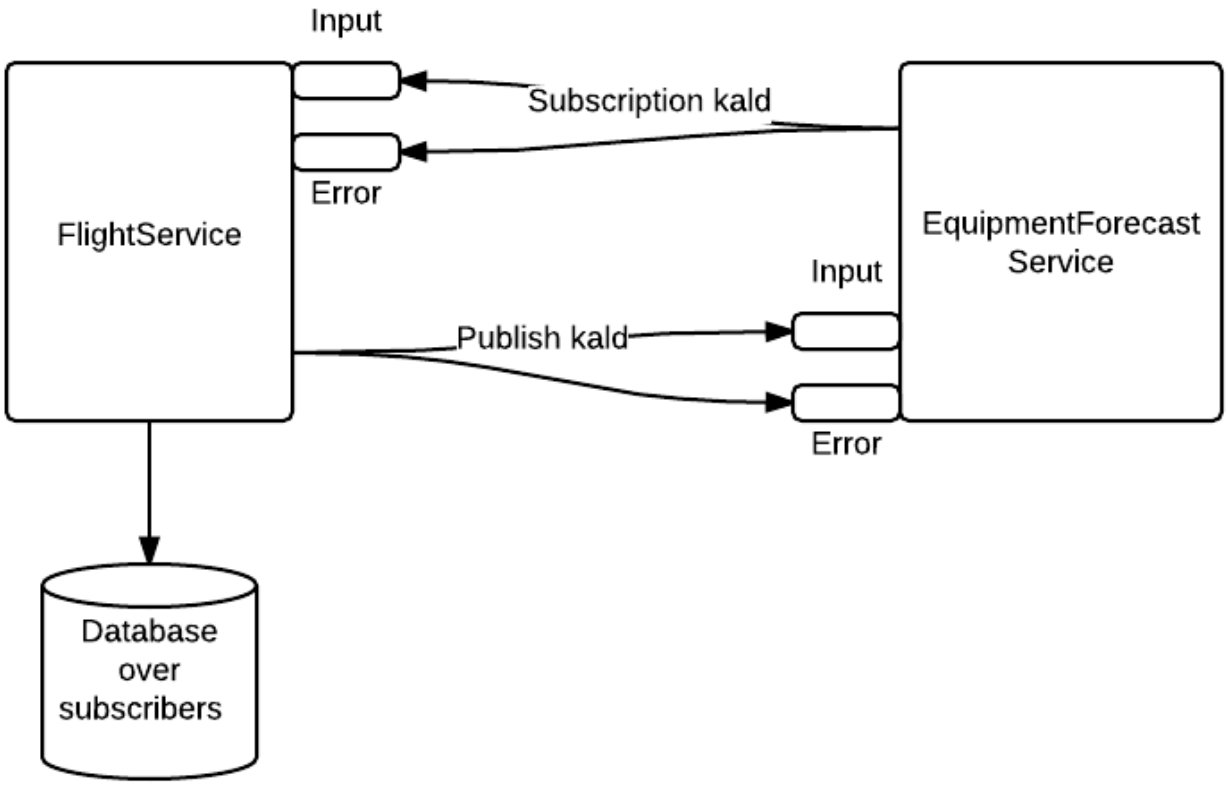
Hos Thomas Cook Airlines Scandinavia bruger man en løsning ved navnet Rebus[9], denne gør det muligt nemt at gøre en service i stand til at være publisher og samtidig gør Rebus det også nemt at få services til at abonnere på den data der gøres tilgængelig.

Rebus fungerer ikke som en traditionel service bus, forstået på den måde at der ikke findes en central enhed (bus) der holder styr på alle publishers og subscribers, i stedet kan man abonnere direkte på den service der udbyder det data man ønsker at blive holdt opdateret omkring.

Der findes en række services der udbyder EDM i virksomheden, det er endnu ikke al data der er tilgængelig på denne måde, men udviklingen peger i den retning. I forbindelse med dette projekt, har det kun været nødvendigt at abonnere på én service – FlightService, som har al information omkring flyvninger udført af Thomas Cook Scandinavia.

Herunder, på Figur 6, er der vist en illustration der beskriver hvordan denne publisher-subscriber proces sker.

[9] Skaberen af Rebus har følgende hjemmeside: http://mookid.dk/oncode/rebus



Figur 6 - Forhold mellem publisher og subscriber

Rebus fungerer ved at sende beskeder til Microsoft Message Queues (MSMQ). En publisher modtager en subscription, her fra EquipmentForecast Service, såfremt beskeden er formuleret korrekt, vil denne blive sat ind på en input kø hos FlightService. Hvis beskeden ikke kan modtages eller der er fejl i den, vil den i stedet blive sat i en fejl kø (error).

Disse subscribers bliver overført til en database, så Rebus altid ved hvilke subscribers der subscriber på hvilken data.

Så snart en service der tilbyder data til sine subscribers har ny data at sende, vil denne sende beskeder til alle der subscriber på denne data. Disse beskeder sendes igen til en MSMQ, hvis beskeden er korrekt sammensat og der ikke opstår problemer med overførslen, bliver denne indsat på den pågældende scubscribers ”input” queue, hvis ikke bliver der lagt en besked i ”error”-køen.

Fordelen ved at bruge MSMQ i denne sammenhæng, er at selv om den modtagende service ikke kører på det tidspunkt den modtager en besked er det stadig muligt for den, når den starter op, at læse beskederne på køen. Det er kun hvis hele den server som servicen kører på går ned, at beskederne går tabt.

Ulempen ved at bruge MSMQ er, at services der ikke får ryddet op efter sig selv på sigt kan være med at lave message queues der indeholder ekstremt meget data, dette kan i så fald skabe

problemer næste gang et program kigger på den pågældende kø og pludselig skal håndtere flere hundreder af beskeder.

EDM i projektet

I projektet er der, som tidligere nævnt, anvendt en eksisterende publisher – FlightService der kan give beskeder om oprettelsen af nye afgange, ændringer i eksisterende afgange eller sletning af eksisterende afgange. EquipmentForecast servicen begynder at abonnerer på alle disse så snart den bliver startet og først herefter bliver data hentet for det næste halve år.

Denne rækkefølge er bevidst valgt, så der ikke opstår et tidsrum hvori der ikke abonneres på beskeder om flyvninger hvilket der potentielt kunne resultere i tabt information omkring flyvninger.

Til brug i dette projekt abonnerer programmet på 3 typer beskeder fra FlightService via EDM:

- FlightCreated
- FlightUpdated
- FlightDeleted

Efter at have modtaget en FlightCreated besked vil programmet skrive informationen om den nye flyvning til databasen. I det uventede tilfælde at information om flyvningen allerede findes i databasen, vil den gamle information blive opdateret.

Dette er valgt ud fra at den nyeste information altid bør være den der mest korrekt og derved den der skal beholdes.

Når programmet modtager en FlightUpdated besked vil det prøve at finde det angivne fly i databasen og ændre oplysningerne på flyet til at passe til den modtagne besked. Såfremt det pågældende fly ikke findes i databasen, vil en ny linje blive oprettet i databasen med det pågældende data.

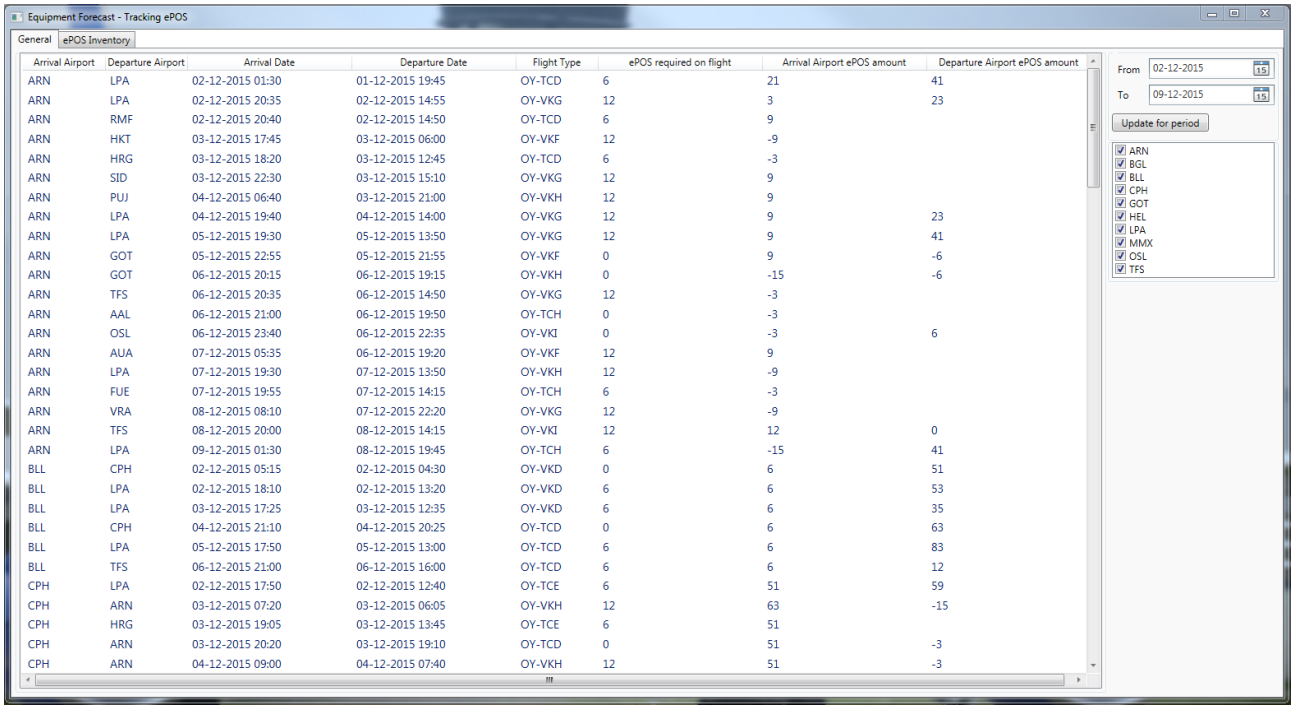
En FlightDeleted besked vil resultere i at information om den pågældende flyvning bliver fjernet fra databasen.

I alle tilfælde bruges den unikke værdi ”LegId” som identifikation.

Opbygning af frontend delen af programmet

Frontend-delen er bygget i Windows Presentation Foundation (WPF) der gør det muligt at skabe grafisk bruger interfaces både ud fra et grafisk interface, men også ved at ændre eller tilføje til den XAML kode der udgør alle elementer i WPF.

Dette er meget gavnligt da de mange attributter på UI-elementer meget nemmere sættes ved ændring i XAML-koden end ud fra interfacet i WPF. Dette betyder at det oftest er fordelagtigt at lave det overordnede design med brugerfladen (”drag and drop” af forskellige typer elementer) og så tilpasse disses egenskaber ved at ændre i XAML koden.



Arrival Airport	Departure Airport	Arrival Date	Departure Date	Flight Type	ePOS required on flight	Arrival Airport ePOS amount	Departure Airport ePOS amount
ARN	LPA	02-12-2015 01:30	01-12-2015 19:45	OY-TCD	6	21	41
ARN	LPA	02-12-2015 20:35	02-12-2015 14:55	OY-VKG	12	3	23
ARN	RMF	02-12-2015 20:40	02-12-2015 14:50	OY-TCD	6	9	
ARN	HKT	03-12-2015 17:45	03-12-2015 06:00	OY-VKF	12	-9	
ARN	HRG	03-12-2015 18:20	03-12-2015 12:45	OY-TCD	6	-3	
ARN	SID	03-12-2015 22:30	03-12-2015 15:10	OY-VKG	12	9	
ARN	PUJ	04-12-2015 06:40	03-12-2015 21:00	OY-VKH	12	9	
ARN	LPA	04-12-2015 19:40	04-12-2015 14:00	OY-VKG	12	9	23
ARN	LPA	05-12-2015 19:30	05-12-2015 13:50	OY-VKG	12	9	41
ARN	GOT	05-12-2015 22:55	05-12-2015 21:55	OY-VKF	0	9	-6
ARN	GOT	06-12-2015 20:15	06-12-2015 19:15	OY-VKH	0	-15	-6
ARN	TFS	06-12-2015 20:35	06-12-2015 14:50	OY-VKG	12	-3	
ARN	AAL	06-12-2015 21:00	06-12-2015 19:50	OY-TCH	0	-3	
ARN	OSL	06-12-2015 23:40	06-12-2015 22:35	OY-VKI	0	-3	6
ARN	AUA	07-12-2015 05:35	06-12-2015 19:20	OY-VKF	12	9	
ARN	LPA	07-12-2015 19:30	07-12-2015 13:50	OY-VKH	12	-9	
ARN	FUE	07-12-2015 19:55	07-12-2015 14:15	OY-TCH	6	-3	
ARN	VRA	08-12-2015 08:10	07-12-2015 22:20	OY-VKG	12	-9	
ARN	TFS	08-12-2015 20:00	08-12-2015 14:15	OY-VKI	12	12	0
ARN	LPA	09-12-2015 01:30	08-12-2015 19:45	OY-TCH	6	-15	41
BLL	CPH	02-12-2015 05:15	02-12-2015 04:30	OY-VKD	0	6	51
BLL	LPA	02-12-2015 18:10	02-12-2015 13:20	OY-VKD	6	6	53
BLL	LPA	03-12-2015 17:25	03-12-2015 12:35	OY-VKD	6	6	35
BLL	CPH	04-12-2015 21:10	04-12-2015 20:25	OY-TCD	0	6	63
BLL	LPA	05-12-2015 17:50	05-12-2015 13:00	OY-TCD	6	6	83
BLL	TFS	06-12-2015 21:00	06-12-2015 16:00	OY-TCD	6	6	12
CPH	LPA	02-12-2015 17:50	02-12-2015 12:40	OY-TCE	6	51	59
CPH	ARN	03-12-2015 07:20	03-12-2015 06:05	OY-VKH	12	63	-15
CPH	HRG	03-12-2015 19:05	03-12-2015 13:45	OY-TCE	6	51	
CPH	ARN	03-12-2015 20:20	03-12-2015 19:10	OY-TCD	0	51	-3
CPH	ARN	04-12-2015 09:00	04-12-2015 07:40	OY-VKH	12	51	-3

Figur 7 - Screenshot af hovedvinduet

Herover ses det primære vindue i programmet, der hvor brugere kan få et overblik over hvordan beholdningen af ePOS terminaler ændrer sig over tid. Der er mulighed for at vælge en tidsperiode, samt at begrænse antallet af lufthavne der bliver vist, enten ud fra interesse eller for at få et bedre overblik.

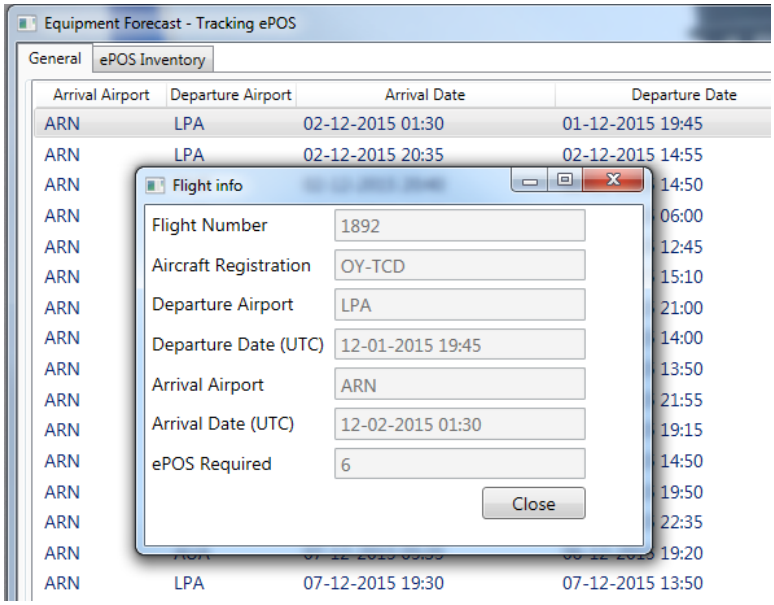
Hver linje repræsenterer her et fly og der er angivet nogle informationer i dette vindue der har med flyet at gøre. Ud over information om flyet, er der to kolonner; ”Arrival Airport ePOS amount” og ”Departure Airport ePOS amount” som angiver hvor mange ePOS terminaler der er tilbage på lufthavnen efter det pågældende fly er fløjet.

Et fly der flyver fra en lufthavn vil således fjerne et antal ePOS fra lufthavnen og et fly der ankommer, vil tilføje et antal til beholdningen af ePOS terminaler.

Det ses af billedet at en del af værdierne for ”Departure Airport ePOS amount” er tomme, dette skyldes at de angivne afgangs-lufthavne ikke håndterer ePOS terminaler og derfor ikke har mulighed for at have terminaler.

På billedet er listen sorteret efter ankomstlufthavnen, dette kan hurtigt ændres til afgangslufthavnen ved at klikke på ”Departure Airport” kolonnen.

Det er muligt at dobbelt-klikke på hver linje for at få vist et pop-up vindue der indeholder flere informationer om den pågældende flyvning, et eksempel på sådan et vindue ses herunder:



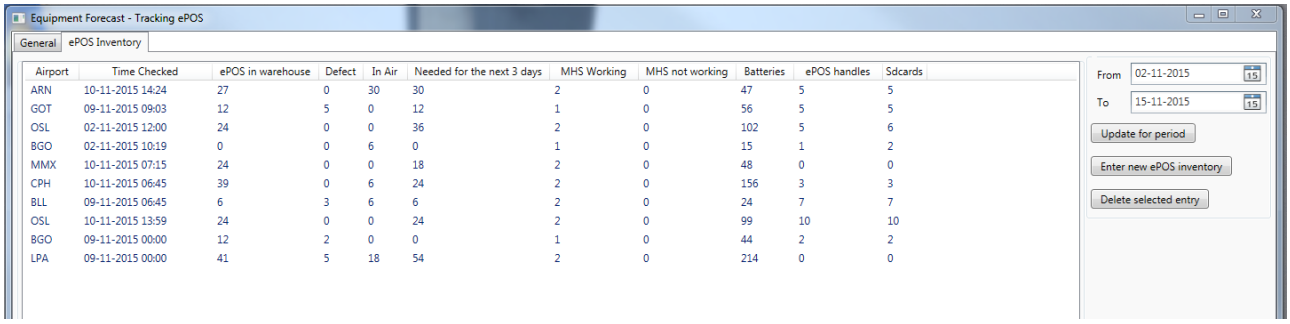
Figur 8 - Pop-up med fly-information

Her kan vises mere information om præcis denne flyvning, information der umiddelbart ikke behøver være på oversigten, men som kan være nyttig at have. Eventuelt i det tilfælde man har behov for at fortælle en lufthavn at der skal lastes ekstra ePOS terminaler på en bestemt flyvning.

En betydelig udfordring har været at udtænke den mest hensigtsmæssige måde at få vist information om antal ePOS terminaler i hver lufthavn, dog er muligheden for at sortere specifikt på en enkelt lufthavn vejen frem.

I toppen af Figur 7 og Figur 8 ses at der er endnu et faneblad, med navnet ”ePOS Inventory”, dette er den side hvorpå det er muligt at få vist tidligere optællinger af ePOS terminaler i lufthavnene, oprette nye eller fjerne entries.

Herunder ses vinduet der giver mulighed for at se tidligere indtastede værdier omkring antallet af ePOS terminaler på de forskellige lufthavne.

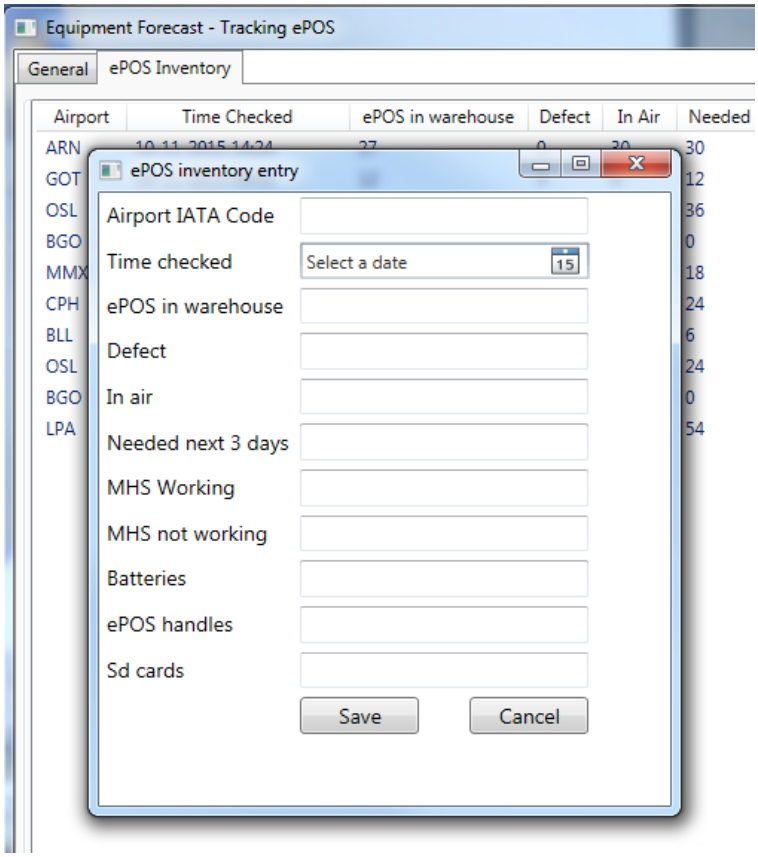


Figur 9 - Vindue med ePOS opgørelser

Her er der igen mulighed for at vælge en tidsperiode, derudover kan man markere en linje og trykke på ”Delete selected entry” for at fjerne den valgte linje.

Knappen ”Enter new ePOS inventory” åbner et nyt vindue, hvori det er muligt at indtaste informationer om ePOS beholdningen på en enkelt lufthavn. Man kan kun indtaste for en enkelt lufthavn af den grund, at alle lufthavne meget sjældent rapporterer disse tal på den samme dag. Således er det stadig muligt at opdatere noget af informationen for på den måde at få det bedst tilgængelige billede af virkeligheden.

Dette vindue ses herunder:



Figur 10 - Oprettelse af ny ePOS beholdning

Her er det muligt at indtaste ovenstående information. Disse felter er taget mere eller mindre direkte fra det nuværende materiale, dog er det kun nogle af felterne der er påkrævede da en gennemgang af tidligere data viste at det ikke er alle værdier der bliver rapporteret hver gang.

De vigtigste værdier er her ”Airport IATA Code”, ”Time Checked” og ”ePOS in warehouse”. Den første angiver hvilken lufthavn der er tale om, den anden hvilken tid og dato optællingen er foretaget og til sidst hvor mange ePOS terminaler der befandt sig i den pågældende lufthavn.

Den valgte løsning set i forhold til problemstillingen

Som det fremgår af opgavebeskrivelen under afsnittet ”Abstract” så var opgaven fra starten ganske åben og udsprang af et konkret behov for information, det var således op til mig at sætte

mig ind i hvordan håndteringen af ePOS terminalerne foregik og hvilken information der var nødvendig i denne sammenhæng.

Dette betød også at jeg igennem processen har itereret igennem forskellige løsningsmodeller, typisk i takt med at mere information omkring processen blev tilgængelig, men også når eventuelle løsninger jeg havde forestillet mig, senere viste sig ikke at være hensigtsmæssige.

Rent teknisk havde det været muligt kun at lave en ren frontend-del, der i så fald skulle lave kald mod den større ”FlightService”, men da holdingen i virksomheden er at bruge mere eller mindre dedikerede services til unikke funktioner og det derudover ville kunne give en uhensigtsmæssigt belastning på ”FlightService” mener jeg at den valgte løsning er den korrekte i denne sammenhæng.

Forslag til alternative måder at håndtere problemstillingen

En stor del af problemstillingen omhandler håndteringen af ePOS terminalerne og at forkert håndtering typisk er grunden til fejl. Derudover har det har vist sig at procedure ændringer ikke har været effektive nok til at skabe forbedringer gode nok til at løse problemstillingen.

En måde at komme ud over dette kunne være en helt fundamental ændring i hvordan ePOS terminalerne bliver håndteret. I stedet for at aflaste og laste terminalerne hver gang et fly lander eller skal flyve, så kunne man i stedet lade terminalerne blive ombord på flyene.

Der er umiddelbart to krav til ePOS terminaler:

- Opladning af batterier til ePOS terminaler
- Internetforbindelse så data på terminalerne kan synkroniseres med systemerne i virksomheden

Det første punkt bør kunne opfyldes ombord på flyene, da det vil være muligt at oplade batterier i hele den tid flyet er i luften samt den tid det befinder sig på jorden, med personale ombord.

Det andet punkt er mere kritisk, da det er nødvendigt at uploade data fra terminalerne, men også for at få ny information om priser og lagerbeholdning ned på terminalerne, selve forbindelsen kan klares med et simpelt 3G/4G modem (internet via mobiltelefon nettet).

Det skal i denne forbindelse nævnes at man i virksomheden har valgt ikke at følge andre selskaber som ”SAS” og ”Norwegian” og ikke kommer til at tilbyde internet adgang til sine passagerer og derved heller ikke vil have adgang til en internetforbindelse ad denne vej.

Dette betyder også at en løsning med 3G/4G modem kun vil kunne benyttes så længe flyet befinder sig på landjorden.

Problemet med denne løsning er, at det ikke er alle steder i verden man kan være sikker på en tilstrækkelig dækning, for at kunne opdatere terminaler ombord på flyet.

Det viser sig at denne løsningsmodel allerede er blevet diskuteret for ca. 4 år siden og at den største udfordring man på det tidspunkt løb ind i var at der ikke var tilstrækkelig dækning til de 3G modemmer man benyttede. Derudover vil det også kræve træning af personale i at håndtere synkronisering af ePOS terminaler, samt generel håndtering af terminaler, der i dag bliver varetaget af andre.

Det kan tænkes at der findes bedre dækning siden man sidst overvejede denne løsning og man bør overveje om der findes en overkommelig måde at få testet dette på.

En anden løsningsmodel, der kan ses som en mellemting mellem den ovenstående model og den nuværende, kunne være at håndtere opladning af batterier på flyet. I stedet for at flytte ePOS terminalerne fra flyet, så nøjes med at tage de SD-kort der sidder i ePOS terminaler ud og overflytte disse til det kontor der plejer at varetage upload af data.

På den måde vil der ikke opstå tvivl omkring hvor ePOS terminalerne befinder sig, da de aldrig under normale omstændigheder skal forlade flyet. De nuværende procedurer for at håndtere terminaler der er gået i stykker kan stadig anvendes, man kan endda forestille sig at der ombord på flyene findes ekstra SD-kort med prisinformation og andet der skal bruges i forbindelse med salg af varer ombord. På den måde vil de enkelte fly ikke længere være 100% afhængige af at der bliver lavet korrekt håndtering på kontorene for at de har det korrekte antal ePOS terminaler.

Konklusion

Som programmet er udformet kan det i sin nuværende form være med til at give et bedre overblik over hvordan behovet for terminaler er, rundt omkring i de lufthavne, der varetager denne opgave. Hvis denne data på sigt kombineres med et mere robust system omkring RFID aflæsning af ePOS terminaler, kan disse to metoder tilsammen skabe et bedre system end hver for sig.

Som programmet er udformet bør det være ganske simpelt at implementere dette. Det vil kunne gøres på to måder; enten at bruge RFID data i stedet for den manuelle optælling der på nuværende tidspunkt sker en gang om ugen. Den anden mulighed er at bruge RFID data udover denne optælling, det vil blot kræve at man tager stilling til i hvor høj grad den manuelle optælling er mere præcis end den maskinelle optælling ved hjælp af RFID.

Derudover kan det være en rigtig god ide endnu en gang at overveje den løsning der allerede for flere år siden var oppe at vende; nemlig at holde så meget håndtering af ePOS terminaler ombord på flyene, for på den måde at være uafhængig af 3. parts selskaber og flytning af terminaler, da det hidtil har vist sig at være en uhensigtsmæssig måde at håndtere problemstillingen på.

For at finde ud af hvorvidt det er muligt at bruge denne løsningsmodel, som blev anset for uanvendelig for 4 år siden, må man teste 3G/4G dækningen på de forskellige lokationer.