

Digital besökslogg: Ett steg mot ett modernare kundbemötande

Digital Visitor Log: A Step Towards a Modernized Customer Encounter

Lucas Pettersson Strömsjö

Fakulteten för hälsa-, natur och teknikvetenskap

Datavetenskap

C-uppsats 15 hp

Handledare: Johan Garcia

Examinator: Johan Eklund

20190116

X

Lucas Pettersson Strömsjö
Författare

X

Johan Garcia
Handledare

X

Johan Eklund
Examinator

Sammanfattning

Denna uppsats beskriver utvecklingen av en lösning vars syfte är att modernisera det bemötande som gäster får på uppdragsgivarens kontor. Lösningen består av en applikation som körs på en surfplatta, en applikation för administration av dess data samt en molnbaserad databas, där Entity Framework är en central del för att koppla samman dessa komponenter.

När en gäst anländer får gästen registrera sig i applikationen på surfplattan och sedan får den anställde som ska ta emot gästen en mobilnotifikation. Notifikationen gör den anställde observant om gästens ankomst. För att identifiera gästen skrivs även en speciellt framtagna ut gästetikett där angiven information från surfplatteapplikation finns angiven.

Abstract

This essay will describe the development of a solution whose purpose is to modernize the reception a guest will get at the employer's office. The solution contains an application which will run on a tablet, an application to administer the data and a cloud-based database, where Entity Framework is the central part of the connection between these components.

At arrival, the guest will register in the application on the tablet and then the employee at the office will receive a mobile notification to notify the employee of the guest's arrival. A custom label will be printed out which contains the assigned information from the application based on the tablet.

Innehåll

Sammanfattning	I
Abstract.....	III
Figurförteckning.....	VII
1. Inledning.....	1
1.1 Syfte	1
1.2 Avgränsningar	1
1.3 Uppdragsbeskrivning	1
1.4 Disposition	2
2. Bakgrund	3
2.1 Distributionsaspekter.....	3
2.2 Microsoft Azure.....	5
2.3 Utvecklingsramverk.....	6
3. Design	11
3.1 Utvecklingsmiljö.....	11
3.2 Team Foundation Server	12
3.3 SQL Server Management Studio	12
3.4 Utvecklingsspråk.....	12
3.5 Använda designmönster.....	14
3.6 Applikationsgränssnitt.....	14
3.7 Directory Services	15
3.8 NuGet.....	15
4. Implementation	17
4.1 SQL-databas.....	17
4.2 Receptionsapplikation för surfplatta.....	22
4.3 Administrationsportal.....	29
4.4 Gästetiketter	35

4.5 Hårdvara	37
5. Resultat	39
5.1 Slutprodukten.....	39
5.2 Målsättning.....	40
5.3 Problem.....	40
6. Slutsats	41
Referenser.....	43
Bilagor	47
Bilaga 1: Lapp för besökare från pärmerna.....	47
Bilaga 2: Arbetsflöde Receptionsapplikation.....	48
Bilaga 3: Arbetsflöde incheckning.....	49
Bilaga 4: Arbetsflöde utcheckning	50
Bilaga 5: Projektspecifikation	51

Figurförteckning

Figur 2:1 : Övergripande bild över lösningens struktur	3
Figur 2:2 : Servicemodeller som lager i en stack, från [4].....	5
Figur 2:3 : Komponenterna i .NET, från [33].....	7
Figur 2:4 : De olika typerna av databindningar, från [9]	8
Figur 3:1 : LINQ-anrop till databas.....	13
Figur 3:2 : Syntax i XAML.....	13
Figur 3:3 : Resultat av kod från figur 3:2	14
Figur 4:1 : Databasdiagram genererat av SQL Server Management Studio	18
Figur 4:2 : Attribut i databasens Entity Data Model.....	20
Figur 4:3 : Kod som används för att skapa ett objekt i tabellen Visit	21
Figur 4:4 : Arbetsflödet i Receptionsapplikationen.....	22
Figur 4:5 : Arbetsflöde vid incheckning i receptionsapplikationen	23
Figur 4:6 : LINQ-query som används för att hitta de anställda som gästen sökt efter. ..	24
Figur 4:7 : Arbetsflöde vid utcheckning i receptionsapplikationen.....	24
Figur 4:8 : Linq-query för att jämföra sökning med ej utcheckade gäster.	24
Figur 4:9 : Kod som körs för att kalla på funktionen SignOutGuestsAfterWorkHours....	25
Figur 4:10 : Utseendet på en resursfil för språk	26
Figur 4:11 : Hur värdet på Content hämtas från en resursfil.....	26
Figur 4:12 : Kod som krävs för att byta resursfil, i detta fall till svenska.....	26
Figur 4:13 : Hur gemensamma attribut på ett element kan sättas i app.xaml.	27
Figur 4:14 : Kod för att skicka ett SMS med Twilio.....	28
Figur 4:15 : Vy för att skapa gäst	30
Figur 4:16 : Vy som returneras vid skapande av ny gäst.....	30
Figur 4:17 : Vy som tar emot en HTTP-post.....	30
Figur 4:18 : Vy för att lägga till gäster via Excel-fil	31
Figur 4:19 : Layout excelmall för tillägg av flera gäster.....	31
Figur 4:20 : Presentation av en post ur tabellen.....	32
Figur 4:21 : Html-helpers för att visa displaynamn samt postens värde för förnamn	33
Figur 4:22 : Actionlänk som visas i form av en knapp.....	33
Figur 4:23 : Kod för sortering, först förnamn och sedan efternamn samt med svensk kultur.	34

Figur 4:24 : Html-element för sökfält.....	34
Figur 4:25 : Tidigare etikett för gäster	35
Figur 4:26 : Layout för ny gästetikett.....	36
Figur 4:27 : Kod för utskrift av etikett	37

1. Inledning

Detta arbete redogör för hur ett företag med hjälp av dagens teknik har möjlighet att modernisera sitt kundbemötande. Med detta i åtanke så går detta arbete ut på att utveckla en lösning till Sogetis kontor i Karlstad, där tanken är att gå från att gäster ska skriva in sig med papper och penna till att kunna göra detta på en surfplatta. För att göra denna lösning möjlig så kommer en skrivbordsapplikation som körs på en surfplatta utvecklas samt en webbapplikation för att administrera data, utöver det tillkommer en molnbaserad databas.

1.1 Syfte

Syftet med projektet är att modernisera bemötandet på Sogetis kontor i Karlstad med hjälp av ny teknologi. Detta görs möjligt genom att utveckla en lösning bestående av en skrivbordsapplikation som ska köras på en surfplatta och en webbapplikation för att kunna administrera data. Utöver det tillkommer även funktionalitet för att skriva ut uppdaterade gästetiketter som klistras på kläderna istället för de klämmor som använts tidigare.

1.2 Avgränsningar

Under utvecklingen av detta projekt har tjänster lanserade av Microsoft varit i fokus, då uppdragsgivaren Sogeti har högsta partnerstatus med Microsoft [1].

1.3 Uppdragsbeskrivning

Uppdraget går ut på att skapa en applikation som ska köras på en surfplatta där det ska vara möjligt att checka in samt checka ut en gäst. Den information som efterfrågas från en gäst är för- och efternamn samt företag och informationen ska sparas i en databas i Microsofts molntjänst *Azure*. För att motverka för gäster att vara anonyma kommer det att vara krav på gästen att ange för- och efternamn vid incheckning. Vid incheckning ska gästen även välja en anställd på kontoret att besöka och då ska en notifikation skickas till den anställdes mobil för att uppmärksamma den anställda att gästens ankomst.

Som en option ska det även vara möjligt för en anställd att kunna verifiera sig genom att ta kort från antingen surfplattan eller en extern webbkamera som sedan jämförs med bilder på de anställda.

1.4 Disposition

Kapitel 2 redogör för bakgrunden för projektet. Detta kapitel innehåller information om den utvecklingsmiljö och de ramverk som använts för att utveckla projektet.

Kapitel 3 redogör för de olika verktyg som används för att komma fram till lösningen.

Kapitel 4 redogör för hur det gått till vid utveckling med de olika komponenterna. Från att ge en mer översiktlig bild av hur systemet är uppbyggt till att gå in på hur designen av databasen och gästetiketter samt information om de olika applikationerna. Utöver det tillkommer kortfattad information om den hårdvara som används för att kunna presentera lösningen samt hur asynkron programmering använts under utvecklingen.

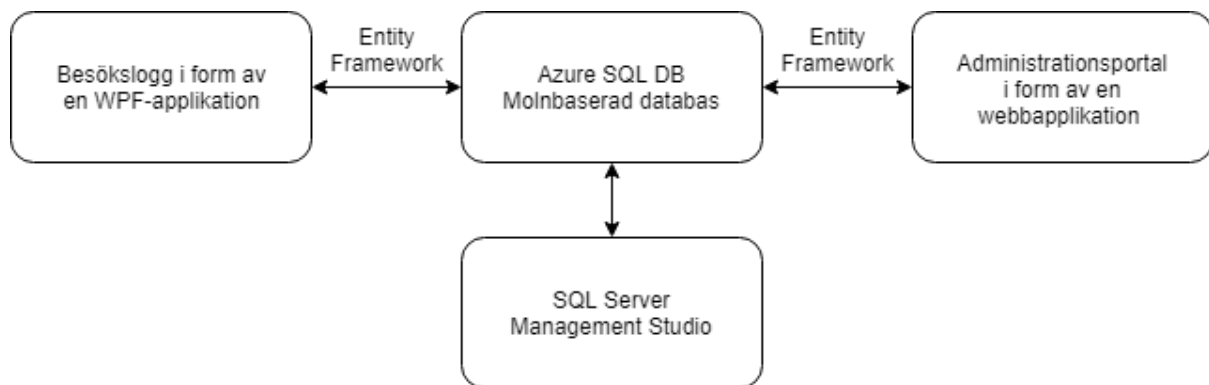
Kapitel 5 redogör för vad som åstadkommits under utvecklingen av projektet samt jämfört det med vad som var angett i projektspecifikationen. Utöver det redogörs för vissa problem som uppstått under utvecklingen av projektet.

Kapitel 6 redogör för vad författaren lärt sig under utvecklingen av projektet, vad som har gått bra och dåligt samt olika förslag på vidareutveckling av projektet.

2. Bakgrund

I detta kapitel delges en överblick över de olika delar av systemet som är relevanta för utvecklingen samt en del information om dessa.

Avsnitt 2.1 beskriver olika service- samt distribueringsmodeller och även hur systemresurser kan användas med hjälp av molntjänster. Avsnitt 2.2 ger en översiktlig beskrivning över vad Microsoft Azure är samt vissa egenskaper hos dess relationsdatabas SQL Azure Database. I avsnitt 2.3 beskrivs de olika ramverk och plattformar som används vid utvecklingen av projektet. I figur 2.1 nedan visas en översiktlig bild hur systemet är upplagt och hur de olika komponenterna är kopplade till varandra.



Figur 2:1 : Övergripande bild över lösningens struktur

2.1 Distributionsaspekter

Detta avsnitt redogör för olika aspekter angående hur en tjänst förmedlas till diverse kunder. I avsnitt 2.1.1 beskrivs hur en molntjänst fungerar samt dess fördelar. Avsnitt 2.1.2 ger en översiktlig bild över hur olika servicemodeller kan hänga ihop samt hur dessa modeller används för att erbjuda tjänster. I avsnitt 2.1.3 beskrivs de olika standarder som finns angående distribuering av molntjänster.

2.1.1 Molntjänster

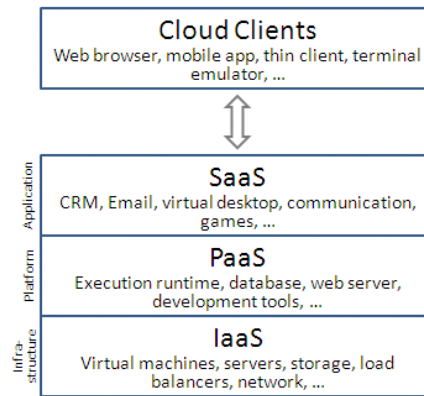
Molntjänster [16] är ett sätt att kunna konfigurera stora mängder systemresurser i olika delade pooler, vilka är skalbara och därför är möjliga att anpassas på ett smidigt sätt till önskad storlek. Då molntjänster har möjligheten att förse kunder med stora systemresurser online har det gjort att många företag lägger mer

pengar på detta för att undkomma att lägga stora resurser på en egen infrastruktur med datorer samt underhåll.

2.1.2 Servicemodeller

Servicemodeller kan ses som lager på varandra i en stack (se figur 2:2), även fast de inte behöver ha en direkt koppling med varandra. Dessa modeller används för att kunna erbjuda en högre mängd abstraktion. Enligt *National Institute of Standards and Technology* (NIST) [16] finns det tre olika modeller som är standard för att kunna erbjuda de tjänster som finnas att tillgå. De olika modellerna heter *Infrastructure as a Service (IAAS)*, *Platform as a Service (PAAS)* och *Software as a Service (SAAS)*.

- *IAAS* används för lagring, nätverk, uträkningar samt distribuering av till exempel applikationer. Detta är möjligt att använda för att ge en användare kontroll över operativsystemet samt lagring, utan att ha rätt att kontrollera eller hantera infrastrukturen i molnet.
- *PAAS* används för att en användare ska kunna distribuera applikationer skapade av användaren. Inte heller denna modell ger några behörigheter till molnets infrastruktur, men har möjlighet att kontrollera applikationen samt möjligtvis kunna konfigurera inställningar i miljön där applikationen körs.
- *SAAS* möjliggör för en användare att kunna använda en leverantörs applikation och sedan köra den i molnet. Dessa applikationer kan bli tillgängliga för användaren på olika sätt, exempel på en sådan applikation är mail som är möjlig att användas i en browser eller i ett användargränssnitt.



Figur 2:2 : Servicemodeller som lager i en stack, från [4]

2.1.3 Distribueringsmodeller

Vid distribuering av de tjänster som molnet använder sig av finns det enligt *NIST* fyra olika standarder [16]. Dessa är privat moln, publikt moln, hybridmoln samt gemensamt moln (community cloud).

- *Privat moln* menas med att en organisation eller enhet har möjlighet att få exklusiv tillgång till infrastrukturen i molnet där molnet kan ägas, hanteras samt drivas av organisationen eller tredje part.
- *Publikt moln* menas med att infrastrukturer är öppna för användning för allmänheten, med behörighet att äga, hantera eller drivas av ett företag eller en myndighet, alternativt en kombination.
- *Gemensamt moln* menas med att molnets infrastruktur förses med exklusivt användande till ett specifikt samfund av konsumenter som har liknande intressen, krav på säkerhet, uppdrag eller liknande policys. Även denna distributionsmodell kan ägas, hanteras samt drivas av flertalet företag eller organisationer eller en kombination av dessa.
- *Hybridmoln* är när en molninfrastruktur en sammansättning av två eller fler av de distribueringsmodeller som nämndes ovan, där de trots att de är separata entiteter binds samman och har möjlighet att förse en användare med fördelar för flertalet distribueringsmodeller.

2.2 Microsoft Azure

Azure [6] är en molnplattform från Microsoft med en stor mängd olika molntjänster riktade mot olika branscher för att kunna hantera diverse

affärsutmaningar. I de två kommande avsnitten beskrivs Azure's molnbaserade databas samt dess kognitiva tjänster.

2.2.1 SQL Azure Database

SQL Azure Database [12] är Microsofts relationsdatabas som är baserad i Microsofts molntjänst Azure. Genom att abstrahera den fysiska lagringsstrukturen, använda automatisk lastbalansering samt anknytningar för att kunna få tillgång till data presenteras data för en användare genom en logisk databas. Varje SQL-databas i Azure är associerad med en Azure-server, vilket är möjligt att ses som en behållare för hantering av en samling databaser som har en gemensam användare.

2.2.2 Cortana Cognitive Services

Cortana Cognitive Services [5] är en del av *Microsoft Azure* där olika problem löses med hjälp av kognitiva tjänster. Dessa tjänster är möjliga att använda för bearbetning av bilder efter visuellt innehåll med hjälp av *Face API* [3]. Utöver funktionalitet för att kunna analysera bilder finns även tjänster för att analysera texter, tal samt att effektivisera sökningar.

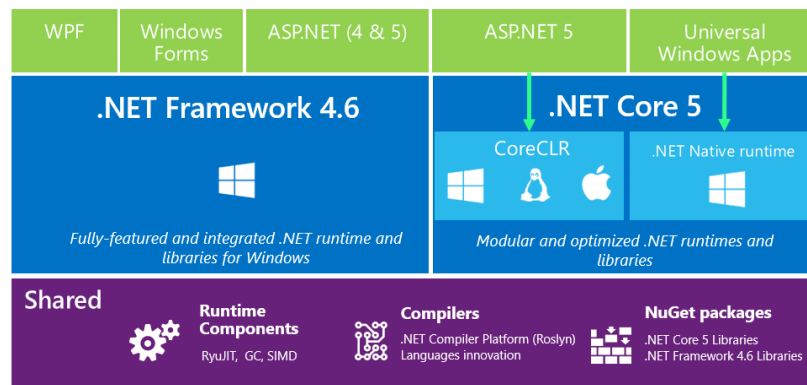
2.3 Utvecklingsramverk

Detta kapitel redogör för de olika plattformar och ramverk som används under utvecklingen. I avsnitt 2.3.1 beskrivs den plattform som lösningen är baserat på. Avsnitt 2.3.2 beskriver den typen av applikation som används för att kunna presentera den skrivbordsapplikation som ska köras på en surfplatta. I avsnitt 2.3.3 beskrivs det ramverk som används till webbapplikationen. Avsnitt 2.3.4 beskriver den teknologi som används för att få tillgång till databasen. Denna teknologi ligger sedan till grund för det ramverk som beskrivs i avsnitt 2.3.5.

2.3.1 .NET

.NET [28] är en open source-plattform som möjliggör för att kunna utveckla många olika typer av applikationer. Genom att använda sig av *.NET* är det möjligt att använda sig av flera olika editorer, språk, samt bibliotek för att kunna utveckla till exempel webb-, mobil- eller skrivbordsapplikationer. I figur 2:3 är det möjligt

att se de olika komponenter som finns i .NET och hur det går att använda med varandra.



Figur 2:3 : Komponenterna i .NET, från [33]

2.3.2 Windows Presentation Foundation

Windows Presentation Foundation (WPF) [10] är ett sätt att skapa applikationer för Windows, där det är möjligt att skapa ett gränssnitt på ett enkelt och användarvänligt sätt med hjälp av *Extensible Application Markup Language (XAML)* (se avsnitt 3.4).

En central del för WPF är att via en framställningsmotor som är oberoende av upplösning kunna använda modern grafisk hårdvara på ett fördelaktigt sätt. Detta är möjligt då WPF är en förlängning av det redan omfattande mängd av funktioner för applikationsutveckling som till exempel tidigare nämnda XAML, databindning, mallar, animeringar och grafik. Då WPF även är en del av Microsoft .NET-ramverket är det möjligt att bygga applikationer med element ur dess klassbibliotek.

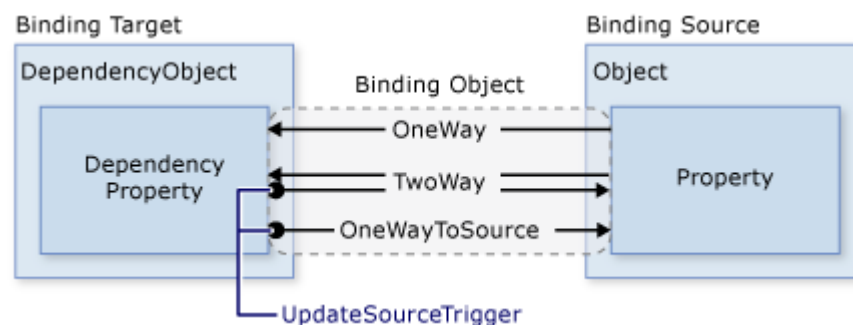
WPF förser tjänster och typer med något som tillsammans brukar kallas för en applikationsmodell. Den gör det möjligt för en utvecklare att utveckla en applikation som antingen är helt självständig eller som körs via en browser, där det sistnämnda kallas för en *XAML Browser Application (XBAP)*.

Dessa applikationer har funktionalitet för att förse användare med sätt att editera och presentera data. Det är möjligt med hjälp av olika tekniker som *Microsoft SQL Server* och *ADO.NET*. Vid tillgång till data som laddas in i applikationens objekt är det möjligt att antingen kopiera och sedan presentera i olika kontroller, för att sedan antingen visa för en användare eller för att editera.

Det går även att användas för att försäkra sig om att den data som en controller har använt sedan kopieras tillbaka till objekten. För att underlätta detta används databindning, där en Binding-klass används. Dess uppgift är att binda ihop den data som använts i kontrollen med dataobjektet, i detta fall är data som används i kontrollen bindningens mål och dataobjektet är bindningens källa.

När det kommer till databindning har WPF olika inbyggda datatjänster som gör möjliggör för applikationsutvecklare att binda data för att sedan kunna manipulera data inom en applikation. Vid databindning stöds fyra olika typer av bindningar (se figur 2:4) [9]:

- *One time*: när klienten ignorerar eventuella uppdateringar på servern.
- *One way*: när klienten får tillgång till data men inte kan manipulera den.
- *Two way*: när klienten kan både skicka data till servern samt läsa data.
- *One way to source*: När klienten endast kan ändra data.



Figur 2:4 : De olika typerna av databindningar, från [9]

2.3.3 Active Server Pages.NET

Active Server Pages (ASP) [25] är ett kostnadsfritt ramverk som används vid utveckling av webbapplikationer och webbsidor. ASP är baserat på .NET och vid utveckling av webbsidor eller webbapplikation har ASP.NET möjlighet att erbjuda tre olika ramverk, Web Forms, ASP.NET MVC (se avsnitt 3.5) och ASP.NET Web Pages. Under utvecklingen av administrationsportalen (se avsnitt 4.4) används ASP.NET MVC.

2.3.4 ADO.NET

ADO.NET [27] används för att få tillgång till datakällor som SQL Server, XML eller OLE DB. Det gör det möjligt för applikation som delar data att kunna koppla upp mot dessa datakällor för att kunna hämta, uppdatera eller hantera dess data. För att kunna koppla upp mot en databas, exekvera kommandon samt hämta resultat av dessa kommandon används .NET-ramverkets *data providers*, då ADO.NET är en del av .NET (se avsnitt 2.3.1). Det är även möjligt att arbeta mot en konceptuell modell istället för den underliggande storage-modellen, detta för att kunna nå en högre nivå av abstraktion för sin applikation. Detta görs med hjälp av *Entity Framework* (se avsnitt 2.3.5).

2.3.5 Entity Framework

Entity Framework (EF) [17] är ett ramverk för ADO.NET med fokus på *Object-Relational Mapping (ORM)*. Vilket menas att det är en del av den mängd teknologier i ADO.NET som används för dataorienterad applikationsutveckling. Utvecklare har med hjälp av EF möjlighet att arbeta mot data i form av domänspecifika objekt och attribut utan att behöva ta hänsyn till var data lagras eller utseendet på databastabellerna. Andra fördelar med EF är att det gör att utvecklare har möjlighet att använda mer abstraktion av data samt skapa och underhålla en applikation på ett effektivare sätt än tidigare.

För att knyta ihop ens applikation med dess data används en *Entity Data Model (EDM)* [17]. Det programmeringsgränssnitt som EF förser använder sig av EDM vid varje interaktion med data, vare sig data ska sparas eller hämtas. Vid användande av EDM behöver inte utvecklaren oroa sig för hur databastabellerna ser ut, utan endast arbeta mot den modell och de klasser som representerar modellens entiteter. Då EF används kommer dess runtime att hantera databaskopplingar, generering av kommandon, utförande av frågor mot databasen samt eventuella ändringar till databasen.

3. Design

Detta kapitel redogör för de olika verktyg som används under utvecklingen av lösningen samt varför de används. I avsnitt 3.1 beskrivs den utvecklingsmiljö som används under projektet samt val av skrivbordsapplikation för lösningen. Avsnitt 3.2 ger en kortfattad beskrivning av det samarbetsverktyg som används genom projektet. I avsnitt 3.3 beskrivs översiktligt den mjukvara som används för att kunna hantera databasen. Avsnitt 3.4 beskriver de utvecklingsspråk som använts flitigast under utvecklingen. Avsnitt 3.5 ger en grundläggande beskrivning av det designmönster som används för webbapplikationen. I avsnitt 3.6 beskrivs det applikationsgränssnitt som används för mobilnotifikationer. Avsnitt 3.7 beskriver tillvägagångssättet för att hämta information i form av kontaktuppgifter till de anställda på kontoret. I avsnitt 3.8 beskrivs hur det är möjligt för utvecklare att dela kod med varandra.

3.1 Utvecklingsmiljö

Under projektet har utvecklingen skett i utvecklingsverktyget *Visual Studio 2017* [2], vilket i detta nu är den senaste versionen av Visual Studio. Visual Studio är en integrerad utvecklingsmiljö från Microsoft vilket stämmer överens med de avgränsningar som antagits för projektet. Visual Studio möjliggör för utveckling av applikationer mot flera olika plattformar som till exempel webb-, Windows- eller mobilapplikationer. Vid skapande av en skrivbordsapplikation så har Windows Presentation Foundation-applikation (se avsnitt 2.3.2) använts istället för en Universal Windows Platform-applikation, detta är för att Windows Presentation Foundation är en del av .NET Framework, medan Universal Windows Platform en del av .NET Core (se figur 2.3). Utöver det så är Universal Windows Platform-applikation anpassade för att kunna köras på flera olika sorters plattformar, vilket i detta fall inte kommer att vara nödvändigt då Windows Presentation Foundation är väl lämpat för att endast kunna köras på en surfplatta med Windows.

3.2 Team Foundation Server

Team Foundation Server (TFS) [7] är ett samarbetsverktyg för utvecklare som är möjligt att integrera i ens utvecklingsmiljö. Används för att effektivisera samarbetet inom team under utvecklingsprocessen av ett projekt. Under utvecklingen av detta projekt användes främst dess versionshanteringssystem [11] samt dess backlog, där det är möjligt att arbeta med olika versioner av projektet, se hur arbetet fortlöpt under utvecklingen samt att kunna skapa arbetsflöden för ens process.

3.3 SQL Server Management Studio

SQL Server Management Studio (SSMS) [20] är en miljö för att för att hantera, administrera och konfigurera data inom Microsoft SQL Server. För att underlätta innehåller SSMS både grafiska verktyg samt möjlighet att skriva skript för att arbete med funktionalitet samt objekt för servern. Av alla funktioner är det Object Explorer som används mest, där användare har möjlighet att välja och söka bland de olika objekten som finns på servern. I detta projekt används SSMS i samband med en SQL-databas från Microsoft Azure (se avsnitt 2.2).

3.4 Utvecklingsspråk

Under utvecklingen har programmeringsspråket *C#* varit det programmeringsspråk som används flitigast. *C#* [15] är ett objektorienterat programmeringsspråk som är en del av .Net-plattformen från Microsoft. Vid exekvering av kod skriven i *C#* behövs en kompilator för att kompilera kod till kallad Common Intermediate Language (CIL)-kod som sedan körs av Common Language Runtime (CLR).

För att i koden kunna skriva enkla förfrågningar till databasen används *Language-Integrated Query (LINQ)*. LINQ [19] är en del av .NET och har vissa likheter med Structured Query Language (SQL) som också används för anrop till databasen. Ett exempel på hur LINQ går att använda visas i figur 3:1, vilket är koden som anropas när besökare söker efter anställd som ska besökas. Varje gång det sker en förändring i textboxen vid namn *TypeNameTextbox* kommer ett LINQ-

anrop att skickas till databasen där den kollar om det finns någon post med ett för- eller efternamn som innehåller det som angetts i textboxen.

```
private void TypeNameTextbox_TextChanged(object sender, TextChangedEventArgs e)
{
    CollectionViewSource employeeViewSource = ((System.Windows.Data.CollectionViewSource)(this.FindResource("employeeViewSource")));
    var search = TypeNameTextbox.Text;
    var Result = from employee in context.Employees
                 where employee.FirstName.Contains(search) || employee.LastName.Contains(search)
                 select employee;
    employeeViewSource.Source = Result.ToList();
}
```

Figur 3:1 : LINQ-anrop till databas

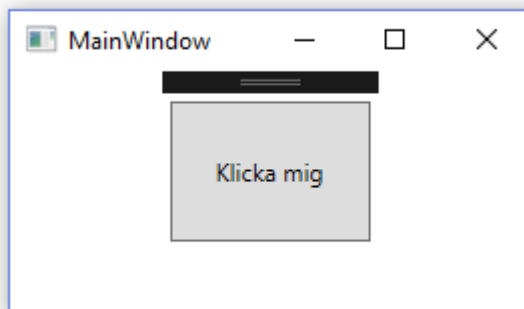
Vid utvecklingen av den administrationsportal (se avsnitt 4.4) som används för att administrera data tillkommer språk som *JavaScript*, *HTML* och *CSS*. Utöver det tillkommer komponentbiblioteket *Bootstrap* [21] som underlättar vid frontend-utveckling i en webbapplikation.

Vid utveckling av en WPF-applikation (se avsnitt 2.3.2) används *Extensible Application Markup Language (XAML)* [8], vilket är ett beskrivande märkspråk som är en del av .NET-ramverkets programmeringsmodell. XAML underlättar vid utveckling av användargränssnitt för applikation som använder sig av .NET. XAML är baserat på XML vilket också är ett märkspråk, vilket betyder att det deklarerar en applikations utseende på ett sätt att det är läsbart för människor.

Dess syntax lyder att en elementdeklaration begränsas med hjälp av "större än" och "mindre än"-tecken, vilket visas i figur 3:2. Inom dessa kommer vilken instanstyp följt av olika attribut för elementet. Resultatet av koden från figur 3:1 visas i figur 3:2.

```
<StackPanel>
    <Button Content="Klicka mig"/>
</StackPanel>
```

Figur 3:2 : Syntax i XAML



Figur 3:3 : Resultat av kod från figur 3:2

3.5 Använda designmönster

Det designmönster som använts heter *Model View Controller (MVC)*, vilket har använts vid utvecklandet av administrationsportalen. MVC är ett designmönster som ofta används vid mjukvaruutveckling samt vid utveckling av användargränssnitt. De tre komponenterna som utgör dess namn används har olika användningsområden men samarbetar. Deras respektive användningsområden [22] är:

- *Model* som utgör kärnan i designmönstret. Används för att hantera data, logik och regler för applikationen utan koppling till användargränssnittet.
- *View* som används för att presentera information för användaren.
- *Controller* har till uppgift att ta emot input från användaren, alternativt validera inputen och sedan anpassa den till olika kommandon till antingen en view eller en model.

3.6 Applikationsgränssnitt

Det applikationsgränssnitt (API) som används under utvecklingen är *Twilio* [23], vilket är ett API för att kunna kommunicera via mobila enheter. De telefonnummer som används måste vara i format e.164 [13], vilket menas med att det börjar med ett + följt av landsnummer och prenumerantnummer och får vara max 15 siffror långt. I Sverige motsvarar detta "+467XXXXXXXX", där varje X ses som en siffra mellan 0 - 9, vilket då bildar ett mobilnummer.

3.7 Directory Services

Directory Services [26] används för att genom kod kunna nå Active Directory. För att göra detta möjligt finns två olika komponentklasser, *DirectoryEntry* och *DirectorySearcher*, vilka använder *Active Directory Services Interface (ADSI)*-teknologi. Med hjälp av ADSI har en användare möjlighet att med rätt behörigheter hantera samt lokalisera resurser på ett relativt enkelt sätt även fast ett nätverk skulle vara väldigt stort.

För att hämta alla anställda från Active Directory används *Lightweight Directory Access Protocol* [14]. LDAP är ett protokoll samt datamodell för hur kommunikation sker gentemot *katalogtjänster* [24] som till exempel en databas eller annat uppslagsverk. Där dess uppgift är att utföra effektiv sökning på katalogservrar. För att koppla upp sig mot en LDAP-server används port 389 som standard [14], även port 636 är möjlig att använda. De olika portarna är portarna för *Transmission Communication Protocol (TCP)* respektive *Transport Layer Security (TLS)*. Via dessa portar är det möjligt för en klient skicka förfrågningar till servern synkront eller asynkront [29], där servern sedan svarar i turordning. Exempel på operationer som en klient kan skicka förfrågningar om är att lägga till, ta bort eller redigera en post, autentisera och specificera LDAP-protokollversion samt att söka och hämta poster från katalogen. LDAP används för att hämta information om de anställda som sedan sparas i Employee-tabellen i databasen.

3.8 NuGet

NuGet [35] är en plattform för utvecklare där det är möjligt att skapa, dela och ta del av andras kod. Detta är mestadels i form av olika paket som innehåller kompilerad kod samt annat innehåll som behövs för att en annan utvecklare ska kunna använda sig av koden.

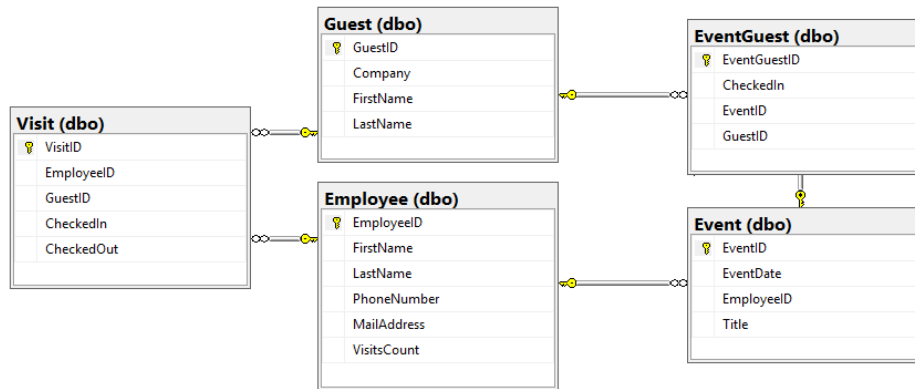
Med hjälp av NuGet Package Manager i Visual Studio är det möjligt att lägga till och installera dessa paket i ens projekt, vilket då gör paketets applikationsgränssnitt (API) tillgängligt i hela projektet.

4. Implementation

I detta kapitel beskrivs hur de olika komponenterna användes för att komma fram till en färdig lösning. Avsnitt 4.1 beskriver utvecklingen av databasen samt hur tabellerna och dess modeller i databasen ser ut. Avsnitt 4.2 ger en mer grundläggande bild över uppbyggnaden för skrivbordsapplikationen. Avsnitt 4.3 beskriver hur webbapplikation är uppbyggd, möjligheten att hantera data samt layout. I avsnitt 4.4 beskrivs utvecklingen av de gästetiketter som ska skrivas ut programmatiskt samt hur dess layout tagits fram. Avsnitt 4.5 beskriver kortfattat den hårdvara som används för att presentera lösningen samt för att skriva ut gästetiketter vid incheckning.

4.1 SQL-databas

Under utvecklingen av databasen används fem databastabeller vilka tagits fram för att kunna presentera relevant information angående ett besök på kontoret. Detta har renderat i fem olika modeller som representeras av fem olika tabeller i databasen (se avsnitt 4.1.1 – 4.1.5). De modeller som tagits fram är Guest, Employee, Visit, Event samt EventGuest. I och med att det i två fall uppstått "många-till-många"-relationer mellan två tabeller behövs det en extra tabell som kopplar ihop dessa. De relationerna är Guest och Employee som kopplas ihop med tabellen Visit, samt Guest och Event som kopplas ihop med tabellen EventGuest. Dessa modeller ska representera den gäst som antingen via ett event eller individuellt besöker en anställd på kontoret, där även viss relevant information om tidpunkten för mötet finns som attribut. För att tydliggöra relationerna mellan de olika tabellerna visas dessa kopplingar i figur 4:1.



Figur 4:1 : Databasdiagram genererat av SQL Server Management Studio

4.1.1 Guest

Guest ska som nämnt ovan representera relevant information om den gäst som besöker kontoret. De attribut som används är hämtade från den pärm som använts tidigare för att registrera besök (se bilaga 1). De attribut som sparas i tabellen är för- och efternamn samt vilket företag gästen kommer ifrån. När en gäst skapas är det krav på att gästen fyller i alla dessa attribut, detta för att motverka att en gäst ska kunna vara anonym.

I denna tabell är *GuestID* primärnyckel, men är även främmandenyckel i tabellerna *Visit* och *EventGuest* för att kunna koppla ihop *Guest* med *Employee* respektive *Event*.

4.1.2 Employee

Employee ska representera den anställda som arbetar på kontoret och ska ta emot gästen. Den information som anses relevant i detta fall är för- och efternamn samt kontaktuppgifter som telefonnummer och mailadress. Dessa uppgifter är relevanta då namnet används för att möjliggöra sökning bland anställda vilket underlättar för gästen vid besök. När en besökare gått igenom arbetsflödet för att checka in skickas en mobilnotifikation i form av ett SMS (se avsnitt 3.6) till den anställda som ska ta emot gästen. I denna tabell är *EmployeeID* primärnyckel men är även främmandenyckel i *Visit*- och *Event*-tabellen.

4.1.3 Visit

Visit ska alltså representera ett besök. Den information som är relevant för ett besök i detta fall är GuestID och EmployeeID, vilka är främmandenycklar och på sätt koppla ihop tabellerna Guest och Employee. Utöver dem kommer tid för in- och utcheckning att anges automatiskt vid incheckning respektive utcheckning.

I denna tabell kommer VisitID att vara primärnyckel, men till skillnad från de andra kommer denna tabell att innehålla de båda andra tabellernas primärnycklar för att koppla ihop tabellerna, men i det här fallet kommer de istället att kallas för främmandenycklar då de refererar till en annan tabell.

4.1.4 Event

Event ska representera relevant information om olika event som Sogeti tillhandahåller på sitt kontor. Ett exempel på sådant event skulle kunna vara ett studiebesök av studenter. De olika attribut som används för att spegla den informationen är dess ID som är dess primärnyckel, vilket datum ett event ska vara samt den anställde som står som ansvarig för ett event. Det är endast möjligt att skapa ett event via administrationsportalen (se avsnitt 4.3), sedan checkar de gäster in som anlänt för ett event via en separat incheckning då de gäster som kommer till ett event ska vara registrerade i förväg.

4.1.5 EventGuest

EventGuest representerar den koppling som är mellan en gäst och ett event. Den information som är intressant i detta fall är vilket event en gäst är kopplad till, vilket gör att ett gäst- samt eventobjekt är de attribut som finns i denna modell.

För att skapa en koppling mellan dessa är det möjligt att lägga till gäster till ett event via en Excel-fil, vilken ska följa mallen som visas i figur 4:18. Detta är möjligt att göra samtidigt som ett Event-objekt skapas i administrationsportalen (se avsnitt 4.3.1).

4.1.6 Från modell till tabell

För att använda dessa modeller i en databas och göra om dem till tabeller med respektive koppling till varandra används *Entity Framework* (EF) (se avsnitt 2.3.5). Till att börja med måste vissa steg utföras, där det första steget är att installera Entity Frameworks *NuGet-paket* från *NuGet Package Manager* (se avsnitt 3.8) eller skriva motsvarande kommando i *Package Manager Console*. Nästa steg är att välja i vilken ände det är tänkt att börja, då det är möjligt att välja *Code First* eller *Database First*. Det förstnämnda är att utvecklaren skapar de modeller som ska presenteras i databasen och sedan skapar EF databasen efter det. Det andra alternativet är att det redan finns en existerande databas som sedan används för att få fram de modeller som ska användas i koden. I detta fall kommer Code First att användas där tidigare nämnda modeller utvecklas först. En Entity Data Model ärver av *DbContext* och de olika modellerna som ska bli tabeller blir attribut av typen *virtual DbSet* av respektive modell, vilket visas i figur 4:2.

```
public virtual DbSet<Employee> Employee { get; set; }
public virtual DbSet<Guest> Guest { get; set; }
public virtual DbSet<Visit> Visit { get; set; }
public virtual DbSet<Event> Event { get; set; }
public virtual DbSet<EventGuest> EventGuest { get; set; }
```

Figur 4:2 : Attribut i databasens Entity Data Model.

När utvecklaren utformat sin *Entity Data Model* på önskat sätt används migrationer. Detta sker genom att köra kommandot *"enable-migrations"* i *Package Manager Console*, då läggs en mapp till i projektet med namnet *"Migrations"* samt en klass vid namn *"Configuration.cs"*. Sedan rekommenderas att en grundläggande migration skapas, vilket möjliggör för att gå tillbaka om det skulle bli fel i framtiden. En migration skapas genom att använda sig av kommandot *"add-migration MigrationsNamn"*, där skapas migrationen och tidigare nämnd mapp vid namn *Migrations*. Varje migration automatgenereras och innehåller de ändringar som skett sedan senaste uppdateringen av databasen. Istället för att ärva av *DbContext* som Entity Data Model, ärver en migration av *DbMigration*.

För att kunna använda sig av funktioner för att kunna lägga till, ta bort eller editera ett objekt i databasen måste en ny instans av den klass som använder sig av *DbContext* att användas. När önskade operationer gjorts mot databasen måste de sparas till databasen med hjälp av "*ContextVariabelNamn.SaveChanges*" eller alternativt "*ContextVariabelNamn.SaveChangesAsync*" om det ska utföras asynkront. Detta visas tydligare i figur 4:3 där ett objekt till tabellen Visit skapas, med hjälp av information som sparats tidigare om en gäst samt den anställde som valts i en Datagrid.

```
using (var context = new SogetiReceptionContext())
{
    var visit = new Visit()
    {
        EmployeeID = id,
        GuestID = tempGuestID,
        CheckedIn = DateTime.Now
    };

    context.Visit.Add(visit);
    context.SaveChanges();
}
```

Figur 4:3 : Kod som används för att skapa ett objekt i tabellen Visit

Vid tillägg av en anställd i tabellen Employee skiljer det sig. Med hjälp av schemaläggning (se avsnitt 4.3.2) används *DirectoryServices* (se avsnitt 3.7) med protokollet LDAP en gång i veckan för att kunna hämta information från önskad domän samt organisationsenhet (kontor). Även här behöver en instans av en Entity Data Model användas för att kunna lägga till eller uppdatera poster i databasen samt en instans av *DirectoryEntry* som innehåller information om varje anställd. Sedan används en foreach-loop för att iterera genom objektet för att hämta information från diverse angivet attribut som i detta fall kallas för "barn" i katalogen. Ett Employee-objekt skapas sedan där informationen från angivna attributen i respektive *DirectoryEntry* läggs till. Informationen jämförs sedan med de poster som finns i tabellen för att se om informationen redan finns lagrad. Skulle det visa sig att den inte är det adderas objektet till databasen med hjälp av "*ContextInstansNamn.Employee.Add(objektnamn);*".

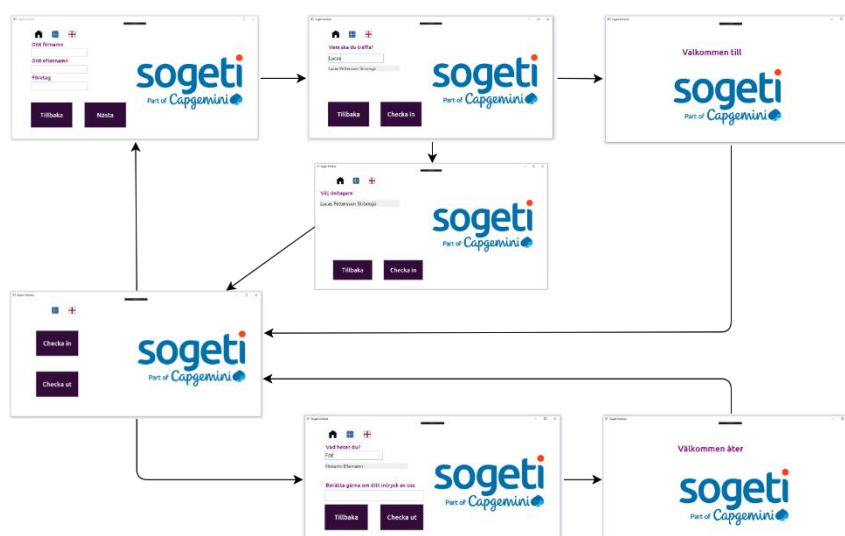
Även för tabellen Event skiljer det sig. Detta är för att Event är till för att kunna lägga till flera besökare samtidigt och att detta ska kunna registreras i förväg. För att kunna göra det på ett smidigt sätt läggs ett Event till via en Excel-fil (se avsnitt 4.3.1).

4.2 Receptionsapplikation för surfplatta

Detta kapitel beskriver hur det gick till vid utvecklingen av skrivbordsapplikationen till surfplattan. I avsnitt 4.2.1 ges en grundläggande beskrivning de olika stegen i arbetsflödet samt en kortare genomgång av den kod som används i respektive steg. Avsnitt 4.2.2 beskriver de olika automatiseringar som gjorts vid utveckling av applikationen till surfplattan med hjälp av schemaläggningar. Avsnitt 4.2.3 ger en översiktlig bild av utvecklingen av funktionalitet för att kunna skifta språk. Avsnitt 4.2.4 beskriver hur gränssnittet till applikationen i surfplattan utvecklats. I avsnitt 4.2.5 beskrivs hur en utvecklare kan gå till väga för att skicka ett SMS på ett programmatiskt sätt.

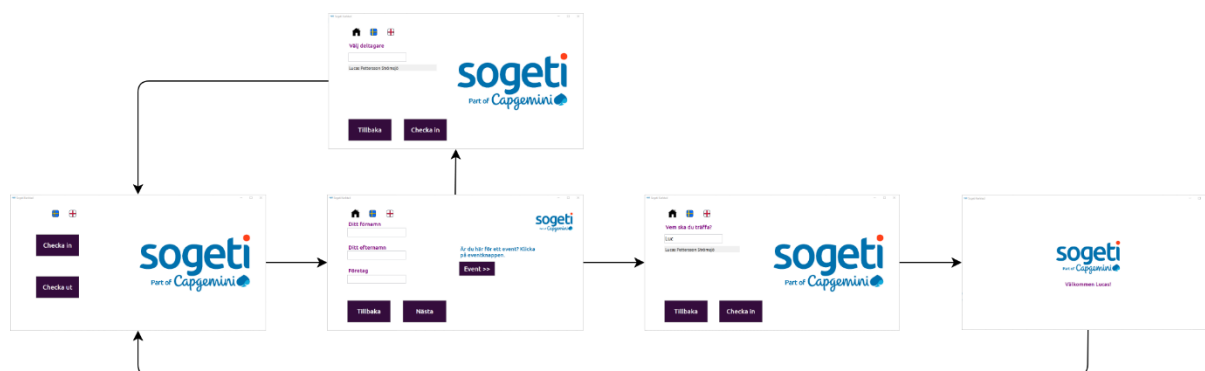
4.2.1 Arbetsflöde

För att få en tydligare och mer översiktlig bild av det arbetsflöde som används i applikationen används diagrammet i figur 4:4 (se även bilaga 2).



Figur 4:4 : Arbetsflödet i Receptionsapplikationen

Figuren ger en översiktlig bild av hur applikationen baserat på val av användaren väljer olika vägar i arbetsflödet. Det första steget är detsamma oberoende av om en gäst använder applikationen för att checka in eller ut. För att få en bättre bild av hur de olika arbetsflöden som används ser ut beroende på om en gäst ska checka in eller ut används figur 4:5 och 4:7.



Figur 4:5 : Arbetsflöde vid incheckning i receptionsapplikationen

Skulle en gäst välja att checka in består det andra steget i figur 4:5 (se även bilaga 3) av att antingen 1) ange sina uppgifter eller 2) checka in som en eventdeltagare. Vid val av att checka in som en eventdeltagare behöver gästen klicka på knappen för att komma vidare till rätt steg. I det steget används en sökruta för att söka bland de gäster som är registrerade på ett event som är planerat under den aktuella dagen. Sedan ska gästen välja gäst att checka in, då registreras besöket och en text som välkomnar gästen att visas. Men applikationen är fortfarande kvar i samma steg av arbetsflödet, detta är för att flera gäster kommer samtidigt och då ska kunna checka in efter varandra på ett smidigt sätt för att minska eventuella risker för köer i entrén. Skulle en gäst istället välja att ange uppgifter är det krav på att ange för- och efternamn samt företag. Detta är för att informationen som anges ska kunna användas vid eventuell evakuering eller vid händelse av brott. Informationen valideras även på sätt att de olika namnen inte får vara längre än 20 tecken samt inte innehålla några symboler eller siffror medan gränsen för företag är 30 tecken. Valideringen för det angivna värdet på företag är inte lika hård angående symboler och siffror då det finns flertalet exempel på företag vars namn innehåller någon form av symbol eller siffror. Utöver det formateras texten att det blir stor bokstav i början på varje angivet värde. Det tredje steget består av att gästen ska ange vilken anställd gästen ska besöka. I detta steg kommer en lista med anställda samt en sökruta att visas. Listan består av de anställda på företaget sorterad efter antalet besök de har fått, samt att om gästen registrerat ett besök tidigare kommer senast angivna anställd att sättas som default. Skulle en gäst välja att söka efter den anställda kommer LINQ-queryn från figur 4:6 att användas.

```
CollectionViewSource employeeViewSource = ((System.Windows.Data.CollectionViewSource)(this.FindResource("employeeViewSource")));
var search = TypeNameTextbox.Text;
var Result = from employee in context.Employee
              where employee.FirstName.Contains(search) || employee.LastName.Contains(search)
              select employee;
employeeViewSource.Source = Result.ToList();
```

Figur 4:6 : LINQ-query som används för att hitta de anställda som gästen sökt efter.

Den jämför värdet i textfältet med namnen på de anställda i databasen och de som matchar värdet visas sedan i listan med hjälp av ViewSource som är kopplad till den DataGrid som visar listan. Denna jämförelse görs varje gång som texten ändras i textboxen.

När en gäst valt anställd och klickar sig vidare checkas gästen in och en instans av ett besök skapas. Där kopplas gästen ihop med den anställda som angivits och datum och tidpunkt för incheckningen registreras och läggs till i databasen. Detta är för att kunna se vilka som har vart på kontoret under dagen samt vem de har träffat. I detta steg dyker en sida upp där gästen välkomnas, sedan används schemaläggning där en schemalagd timer används för att navigera tillbaka till början av applikationen igen (se avsnitt 4.2.2).

Skulle en gäst istället välja att checka ut används arbetsflödet som visas i figur 4:7 (se även bilaga 4) istället.



Figur 4:7 : Arbetsflöde vid utcheckning i receptionsapplikationen.

Som nämnt tidigare är det första steget för utcheckning samma som för incheckning. Det andra steget består av en sökruta och en lista där innehållet i listan även här anpassas efter innehållet i sökrutan, med hjälp av den LINQ-query som visas i figur 4:8 där endast de som checkat in under dagen men inte checkat ut än visas.

```
var Result = from visit in context.Visit
              join guest in context.Guest on visit.GuestID equals guest.GuestID
              where visit.CheckedIn > DateTime.Today && visit.CheckedOut == null && (guest.FirstName.Contains(search))
```

Figur 4:8 : Linq-query för att jämföra sökning med ej utcheckade gäster.

Även här läggs resultatet av queryn till i en ViewSource som är kopplad till den DataGrid som visar listan. Denna jämförelse görs också varje gång som innehållet i sökrutan ändras. Vid val av en gäst för utcheckning uppdateras det Visit-objektet även med en utcheckningstid. Även här används schemaläggning för att via en schemalagd timer kunna navigera applikationen tillbaka till startsidan igen.

4.2.2 Schemaläggningar

För att kunna skriva schemalagda rutiner används *FluentScheduler*. I detta projekt används det bland annat för att efter varje arbetsdags slut leta igenom tabellen för besök och se om det finns några gäster på kontoret som inte har checkat ut sig. Ett exempel på hur det kan se ut i koden syns på figur 4:9, där en funktion triggas vid en viss tidpunkt, i detta fall SignOutGuestsAfterWorkHours som sker vid 17:00.

```
JobManager.AddJob(  
    this.SignOutGuestsAfterWorkHours,  
    schedule => schedule.ToRunEvery(1).Days().At(17, 00));
```

Figur 4:9 : Kod som körs för att kalla på funktionen SignOutGuestsAfterWorkHours.

Samma princip används även som en timer i slutet av arbetsflödet för in- respektive utcheckning. Då en gäst har antingen checkat in eller checkat ut skickas gästen automatiskt till startsidan efter ett visst tidsintervall. Detta liknar koden från bild 4:9 ovan, men skillnaden är att då kallar schedule på *ToRunOnce* med angivet tidsintervall i sekunder som inparameter, där funktionen körs efter det angivna tidsintervallet.

Schemaläggning används även för att leta efter nya anställda, där en funktion triggas en gång i veckan (se avsnitt 4.1.6), där en jämförelse mellan de anställda som finns i organisationsenhet för Karlstadkontoret i Active Directory jämförs med de anställda som finns som poster i databasen. Skulle det vara en nyanställd som inte redan finns i databasen så läggs den nyanställda till.

Utöver detta körs även en jämförelse för de event som eventuellt har vart under dagen, då jämförs vilka gäster som var registrerade med de som checkat in under dagen för att se vilka som skrivit upp sig men inte dykt upp.

4.2.3 Språk

På de sidor i applikationen som inte automatiskt navigeras till startsidan (se avsnitt 4.2.2) på surfplattan är det möjligt att välja vilket språk som önskas. Detta är möjligt med hjälp av resursfiler som skapas och namnges enligt *International Organization of Standardization* (ISO)-standard [30] där två bokstäver beskriver språket följt av ett bindestreck och två bokstäver till för att beskriva vilket land, då vissa länder har flera olika språk eller vissa språk används i flera olika länder. Till exempel skrivs svenska som sv-SE där sv representerar språket svenska (enligt ISO 639-1) [31] och SE representerar landet (enligt ISO 3166-1) [32]. I XAML-koden (se avsnitt 3.4) kopplas den komponent i koden som ska kunna översättas. Resursfilerna som används i detta fall visas i figur 4:10 och har tre olika kolumner för komponentens namn, värdet som ska visas samt en kommentar som anger den sida i applikationen där komponenten finns.

Name	Value	Comment
CheckInButton	Checka in	StartPage
CheckOutButton	Checka ut	StartPage
CheckOutNameLabel	Vad heter du?	CheckOutPage
CompanyLabel	Företag	CheckInPage

Figur 4:10 : Utseendet på en resursfil för språk

Hur namnet på komponenten kan kopplas till den resursfil som används via XAML visas i figur 4:11.

```
<Button Name="CheckInButton" Content="{x:Static p:Resources.CheckInButton}" Click="CheckInButton_Click"/>
<Button Name="CheckOutButton" Content="{x:Static p:Resources.CheckOutButton}" Click="CheckOutButton_Click"/>
```

Figur 4:11 : Hur värdet på Content hämtas från en resursfil.

För att kunna byta språk finns det två flaggor tillgängliga i ena hörnet på gränssnittet där användaren kan välja språk. Den kod som behövs för att byta språk visas i figur 4:12.

```
System.Threading.Thread.CurrentThread.CurrentUICulture =
new System.Globalization.CultureInfo("sv-SE");
```

Figur 4:12 : Kod som krävs för att byta resursfil, i detta fall till svenska.

Utöver den kod som används i figur 4:12 behövs även en ny instans av den sida som är aktiv, där även eventuellt skriven text från textfälten följer med som

inparametrar. Detta gör att en gäst inte behöver skriva om allting vid byte av språk.

4.2.4 Gränssnitt

I en *Windows Presentation Foundation (WPF)*-applikation (se avsnitt 2.3.2) används *XAML* (se avsnitt 3.4). För att underlätta utvecklandet av ett användarvänligt gränssnitt skrivs så mycket kod som möjligt i filen "*app.xaml*". Det är för att sätta standardvärden på olika attribut för olika element som då gäller i hela applikationen. Det gör att varje attribut behöver beskrivas flera gånger om, vilket gör koden lättare att läsa. För att göra detta möjligt anges olika *Style*-element i "*app.xaml*"-filen där först önskad komponent anges, sen sätts de olika attributen med hjälp av "*Setters*" där först önskat attribut anges sedan dess värde, vilket visas i figur 4:13.

```
<Style TargetType="TextBox">
  <Setter Property="FontSize" Value="25"></Setter>
  <Setter Property="Height" Value="40"></Setter>
  <Setter Property="Width" Value="270"></Setter>
  <Setter Property="HorizontalAlignment" Value="Left"></Setter>
  <Setter Property="VerticalAlignment" Value="Top"></Setter>
  <Setter Property="TextWrapping" Value="Nowrap"></Setter>
</Style>
```

Figur 4:13 : Hur gemensamma attribut på ett element kan sättas i *app.xaml*.

För att anpassa applikationen efter företaget har även en grafisk formgivning med anpassade värden på olika attribut tagits fram. Denna är inspirerad av hemsidan där en font vid namn "Ubuntu-medium" används. Även färgerna i applikationen är hämtade från hemsidan samt den logga som under tiden för projektet lanserades.

4.2.5 Mobilnotifikationer

För att uppmärksamma en anställd om att dess gäst har anlänt används mobilnotifikationer i form av SMS. När en gäst slutför sin incheckning via receptionsapplikationen i surfplattan kommer ett SMS att skickas till den anställdes nummer som finns lagrat i databasen.

För att kunna göra detta möjligt används *Twilio* (se avsnitt 3.6), vilket är ett applikationsgränssnitt (API) som används till diverse kommunikation mellan

mobila enheter. Då Twilio inte finns att tillgå från början av ett projekt måste det laddas ner via *NuGet Package Manager* (se avsnitt 3.8). Det som behövs för att i detta fall kunna skicka ett SMS till en anställd är ett konto-id samt en token för autentisering. Sedan hämtas respektive gäst samt anställd för att sedan kunna användas i koden som visas i figur 4:14 senare. För att kunna göra detta används respektive objekts id och sedan används punktnotation för att kunna hämta ut den anställdes telefonnummer ur dess objekt. Numret som används för att skicka SMS är ett nummer som kan hämtas i Twilio och det lagras som en konstant och används sedan när ett meddelande ska skickas.

```
TwilioClient.Init(accountSid, authToken);
Employee employee = GetEmployee(employeeID);
Guest guest = GetGuest(guestID);

var notification = MessageResource.Create(
    body: "Detta är innehållet i ditt SMS.",
    from: new Twilio.Types.PhoneNumber(ReceptionPhoneNumber),
    to: new Twilio.Types.PhoneNumber(employee.PhoneNumber)
);
```

Figur 4:14 : Kod för att skicka ett SMS med Twilio.

För att skicka meddelandet används en meddelanderesurs, vilket består av de tre delar (se figur 4:14 ovan):

- *Body*: Den text som mottagaren av meddelandet får.
- *From*: Det telefonnummer som står som avsändare för meddelandet.
- *To*: Telefonnumret till den telefon är mottagaren av meddelandet.

4.2.6 Asynkron programmering

För att effektivisera applikationerna används asynkron programmering. Det betyder att det är möjligt att utföra flera operationer samtidigt, vilket används för till exempel databasanrop och validering i applikationen. I och med att valideringen utförs asynkront är det möjligt att validera en gästs för- och efternamn samt företag samtidigt som de uppgifterna används för att kolla i databasen om den gästen registrerat sig tidigare eller redan checkat in för dagen.

För att kunna utföra programmering asynkront används nyckelorden *async/await* [34] samt en lista med diverse tasks, som väntar in att alla tasks ska

köras klart innan den går vidare. Utöver det ändras även texten på diverse knapp som tryckts med information om att applikationen laddar.

4.3 Administrationsportal

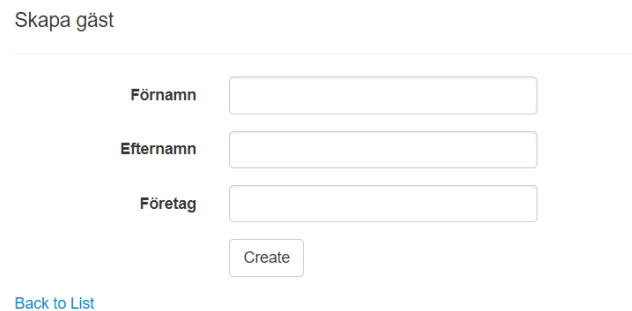
För att underlätta administreringen av data från skrivbordsapplikationen för surfplattan (se avsnitt 4.2) innehåller lösningen i Visual Studio även en webbapplikation. Denna applikation använder designmönstret *MVC* (se avsnitt 3.5). Webbapplikationen underlättar för de som administrerar all data som samlas in vid besök. Den har funktionalitet som gör det möjligt för administratören att lägga till, editera, läsa eller ta bort poster ur databastabellerna på ett enkelt sätt med hjälp av *Entity Framework* (se avsnitt 2.3.5), vilket även används i surfplatteapplikationen. I de följande avsnitten ges beskrivningar över hur administreringen fungerar, hur det är möjligt att söka i realtid i tabellerna samt en kortfattad beskrivning av gränssnittet.

4.3.1 Administrering

För att göra det möjligt att hantera och administrera bland gäster, anställda, besök och event krävs det att applikationen har viss funktionalitet. Till skillnad från applikationen till surfplattan är det möjligt att lägga till Event via administrationsportalen, vilket görs via en Excel-fil (se figur 4:18). Detta möjliggör för att lägga till flera gäster samtidigt som knyts till samma anställda, detta kan vara användbart vid till exempel studiebesök. I administrationsportalen ska det även gå att presentera, sortera data samt kunna ändra eller ta bort poster ur respektive tabell på ett användarvänligt sätt.

För att göra detta möjligt i MVC (se avsnitt 3.5) används en controller med en mängd funktioner av returtypen *ActionResult* som returnerar diverse vy med samma namn som funktionen. Om det ska presenteras data på vyn behöver även en modell alternativt en *ViewModel* skickas med, där *ViewModels* används för att kunna anpassa vilka olika attribut från en modell som ska användas i vyn. Vid händelse av att skapa en ny post i en tabell behövs i första läget inte någon data presenteras, vilket gör att en modell eller *ViewModel* inte behöver skickas med

till vyn. Istället visas flertalet element som utgör det formulär som visas i figur 4:15.



Figur 4:15 : Vy för att skapa gäst

Denna vy nås genom att gå in på fliken "Gäster" i navigationen och sedan välja "Lägg till gäst". Detta gör att det endast returneras en vy i detta steg, vilket visas i figur 4:16.

```
public ActionResult Create()
{
    return View();
}
```

Figur 4:16 : Vy som returneras vid skapande av ny gäst

När formuläret är ifyllt och skickas tas det emot i form av en HTTP-Post och då ser funktionen för att skapa en gäst annorlunda ut. Även här valideras angivna värden och skickas sedan med som inparametrar till funktionen, om allt stämmer betyder det att tillståndet för modellen är godkänt och objektet sparas till databasen, vilket visas i figur 4:17.

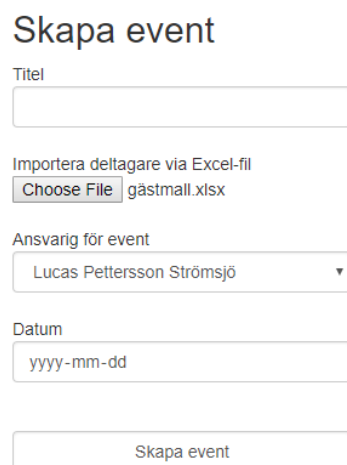
```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "GuestID,Company,FirstName,LastName")] Guest guest)
{
    if (ModelState.IsValid)
    {
        db.Guest.Add(guest);
        db.SaveChanges();
        PrintGuestLabel(guest.FirstName, guest.LastName, guest.Company);
        return RedirectToAction("Index");
    }

    return View(guest);
}
```

Figur 4:17 : Vy som tar emot en HTTP-post

Även här tillkommer en skillnad mellan applikationerna, för när gäster läggs till via administrationsportalen skrivs inga gästetiketter ut.

Om en användare istället skulle vilja lägga till ett event för att kunna koppla flera gäster samtidigt ska istället användaren klicka på "Event" i navigationen och sedan "Skapa Event". På denna vy är det möjligt att fylla i de attribut som finns för Event (se avsnitt 4.1.4) där element för att ange titel, välja dokument, anställd samt datum visas (se figur 4:18).



Skapa event

Titel

Importera deltagare via Excel-fil

Choose File gästmall.xlsx

Ansvarig för event

Lucas Pettersson Strömsjö

Datum

yyyy-mm-dd

Skapa event

Figur 4:18 : Vy för att lägga till gäster via Excel-fil

För att underlätta vid tillägg av gäster via Exceldokument har en mall tagits fram, där de olika kolumnerna motsvarar de attribut som finns i gäst-modellen (se figur 4:19).

	A	B	C
1	Förnamn	Efternamn	Företag
2	Lucas	Pettersson Strömsjö	Sogeti
3	Demo	Demosson	Demo AB

Figur 4:19 : Layout excelmall för tillägg av flera gäster

För att kunna hämta information från en Excel-fil programmatiskt används Interop Excel, vilket kräver att Microsoft.Office.Interop.Excel från NuGet Package Manager (se avsnitt 3.8) installeras och refereras till i klassen. Detta kräver att användaren lägger till en fil som är av xlsx-format, vilket motsvarar ett Exceldokument. Till den separata vyn används en ViewModel där det är möjligt att anpassa innehållet för modellen till en speciell vy. Denna ViewModel består av

ett Exceldokument av datatypen `HttpPostedFileBase`, attribut för att spara allt innehåll från filen, filens filväg samt att det tillkommer attribut från eventet som dess titel, datum, gäster samt ansvarig anställd. I figur 4:17 visades hur vyn för att skapa ett event ser ut. I kontrollern skapas en variabel av tidigare beskriven `ViewModel` följt av en lista som initieras för att kunna användas i en dropdownlista för att välja anställd för eventet. Detta görs genom att ange vilken tabell som data ska hämtas från, det värde som ska visas i dropdownlistan samt det värde som ska tas emot för att lagra valet av anställd. Informationen läggs sedan till i den skapade instansen av beskriven `ViewModel` och skickas tillbaka till kontrollern i form av en HTTP-Post. Tillbaka i kontrollern skapas först en instans av den context som används till databasen, sedan ska filen hanteras där information som filnamn, filväg, filformat lagras för att sedan användas. För att sen hämta information från ett Exceldokument måste flera variabler av olika datatyper kopplade till Excel att behövas. Dessa är `Application`, `Workbook`, `Worksheet` och `Range`. De används för att välja vilket dokument samt blad som ska användas, även vilken mängd celler i dokumentet som används. Efter validering för att se att filformat och innehåll stämmer kollar två olika for-loopar igenom den information som finns i filen vilket begränsas av `Range`. Sedan itereras varje rad i Exceldokumentet och om informationen stämmer läggs gästerna till i databasen. Inte heller här skrivs någon gästetikett ut, utan det sker sedan när en eventdeltagare checkar in via surfplatta.

För att istället ändra en post i databasen ska först önskad flik i navigationen anges. När en användare klickat in på önskad flik visas en tabell med de poster som finns i databasen. Hur en post kan presenteras syns i figur 4:20 nedan, där länkar till diverse funktionalitet finns i anknytning till varje post.

Namn	Företag	
Lucas Pettersson Strömsjö	KAU	Editera Visa Radera

Figur 4:20 : Presentation av en post ur tabellen

Vid val att editera används en vy som liknar den som används vid manuell inmatning av värden, men istället för tomma textfält fylls de med information som hämtats ur tabellen. Detta gör att det är möjligt att editera den text som visas i

textfälten och sedan spara informationen. Istället för att skapa en ny post i tabellen skrivs istället den valda posten över med den nya informationen.

Vid val av "Visa" returneras en vy från diverse post att returneras där attributets namn följt av attributets värde visas med hjälp av HTML-helpers som visas i figur 4:21.

```
<dt>
  @Html.DisplayNameFor(model => model.FirstName)
</dt>

<dd>
  @Html.DisplayFor(model => model.FirstName)
</dd>
```

Figur 4:21 : Html-helpers för att visa displaynamn samt postens värde för förnamn

Detta görs genom att modellen används samt att postens id skickas med som inparameter.

Om en post ur tabellen ska tas bort visas en vy som liknar den för att visa informationen från posten i tabellen, med tillägg att en knapp med texten "Radera" finns vilken gör att posten raderas om användaren skulle trycka på den.

4.3.2 Sök- och sorteringsfunktionalitet

För att kunna sortera de poster som finns i respektive tabell används *Html.ActionLink* där texten på länken, vilken action i dess controller som ska köras samt eventuella värden som ska skickas med till tidigare nämnd action att skickas (se figur 4:22 nedan).

```
@Html.ActionLink("Lägg till anställd", "Create", null, new { @class = "btn btn-default" })
```

Figur 4:22 : Actionlänk som visas i form av en knapp

I detta fall används två olika *ViewBags*, dessa används för att temporärt lagra ett värde. I detta fall lagras de vilken kolumn som ska sorteras och i vilken ordning (stigande eller fallande) som kolumnen ska sorteras. De värden som angetts läggs i en sträng och skickas sedan vidare till önskad action i kontrollern. I kontrollerns action jämförs sedan valet av sortering med en *switch-case-sats* (se figur 4:23) för att sedan ange vilken kolumn som tabellen ska sorteras efter samt ordning.

```

case "Employee":
    visits = visits.OrderBy(v => v.Employee.FirstName, StringComparer.Create(culture, false))
        .ThenBy(v => v.Employee.LastName, StringComparer.Create(culture, false)); break;
case "Employee_desc":
    visits = visits.OrderByDescending(v => v.Employee.FirstName, StringComparer.Create(culture, false))
        .ThenBy(v => v.Employee.LastName, StringComparer.Create(culture, false)); break;

```

Figur 4:23 : Kod för sortering, först förnamn och sedan efternamn samt med svensk kultur.

För att möjliggöra för sortering med svenska alfabetet måste en lista med *Enumerables* av önskad datatyp att skapas, som sedan fylls via ett databasanrop i LINQ. Detta är för att *IQueryable* som är default vid LINQ-query inte kan sorteras med kultur för det svenska språket. Därför används alltså *IEnumerable* istället, där det var möjligt att använda *OrderBy* och sedan välja attribut att sortera på samt välja kultur som inparameter. Då det finns en möjlighet att flera värden i dessa tabeller kan vara lika, till exempel flera på kontoret med samma namn, ordnas anställda först på förnamn och sedan på efternamn vilket visades i figur 4:23.

Det ska även vara möjligt att kunna söka efter information i de olika tabellerna. För att kunna göra detta i realtid används JavaScript. Sökfunktionen anropas varje gång texten i sökfältet ändras vilket medför att användaren inte behöver trycka på någon knapp efter önskat sökvärde angetts. Sökfältet har även attributet *autofocus* (se figur 4:24), vilket medför att en användare inte ska behöva trycka på sökfältet när dess sida öppnats, utan det ska kunna gå att skriva direkt.

```

@using (Html.BeginForm())
{
    <div class="form-group form-inline">
        @Html.TextBox("searchEmployee", null, new { @class = "form-control", @autofocus = "autofocus",
            @placeholder = "Sök på förnamn eller efternamn", @onkeyup = "searchTable()" })
    </div>
}

```

Figur 4:24 : Html-element för sökfält

Sökfunktionen används för att kunna filtrera information för alla tabeller och anpassas för att kunna söka på de mest relevanta attributen i respektive tabell. Den text som angetts i textfältet efter varje anrop till funktion lagras i en variabel där texten sedan anpassas med *ToUpper()*. Vid jämförelse med respektive värde från tabellen använder även dessa värden *ToUpper()* vilket gör att sökningen inte är case sensitive. De andra variablerna i sökfunktionen är tabellens namn, variabel för ett radelement samt en variabel för respektive kolumn i tabellen som

sökfunktionen ska använda till jämförelsen. För att kunna jämföra önskade värden med det angivna värdet från sökfältet används en for-loop, den itererar så många varv som det finns poster i tabellen. Där används if-satser för att se om det angivna värdet i sökfältet matchar något av de värden som finns i de angivna kolumnerna, skulle det inte göra det sätts värdet på *display* på det ej matchande elementet till "none".

4.3.3 Gränssnitt

Vid start av administrationsportalen möts användaren av en startsida, sedan är det möjligt för användaren att navigera sig fram via navigeringen och på så sätt kunna se information från respektive tabell. På var och en av sidorna möts användaren av tabellens titel och en länk till funktionalitet att kunna addera post till tabellen. Utöver det visas de poster som hämtats från tabellen samt att varje post även innehåller länkar till funktionalitet för att kunna se detaljer, editera eller ta bort post. Istället för att visa respektive posts id som hämtats från objekten, visas istället dess gäst alternativt anställdes fulla namn.

Informationen för besök visar inte heller ID:n, istället kommer respektive gäst eller anställds fulla namn att visas för att underlätta för användaren. Även detta för att underlätta läsbarheten av posterna för användaren.

4.4 Gästetiketter

Ett steg mot modernisering av hantering av besökare är att uppgradera de tidigare besöksbrickorna (se figur 4:25).



Figur 4:25 : Tidigare etikett för gäster

Detta har skett genom att beställa en etikettskrivare (se 4.5.2), där en klisterlapp används som kan sättas på kläderna alternativt sättas i en plastbricka med klämma. Genom att använda sig av det Software Development Kit som tillkommer med etikettskrivaren är det möjligt att via receptionsapplikationen skriva ut dessa lappar med information som förnamn, efternamn och företag. Något som är viktigt att tänka på vid utveckling av applikationer som ska använda *bpac* är att de måste använda sig av 64-bitars plattform när projektet byggs. Detta går att ändra på i projektets egenskaper samt att ändra i inställningarna för webbprojekt, där både projektet och IIS måste köras med 64-bitar.

Till att börja med skapades en design för hur en besökslapp ska se ut, vilket kan ses i figur 4:26.



Figur 4:26 : Layout för ny gästetikett

För att sen kunna anpassa texten efter den gäst som checkat in får textfälten med texten "Förnamn Efternamn" och "Företag" olika namn genom att i *P-touch Editor* (se avsnitt 4.5.2) ge olika objektnamn till respektive textfält. Sedan är det möjligt att via kod kunna skriva över den texten med information som hämtats vid incheckning.

För att kunna göra detta rent programmatiskt måste *bpac* (se avsnitt 4.5.2) installeras på datorn samt att en referens till *bpac* måste användas i projektet samt i koden. Efter att informationen gästen angivit validerats körs en asynkron funktion med gästens id som inparameter. Detta ID används i en foreach-loop för att med hjälp av en if-sats kunna hitta rätt gäst i databasen. Sedan skapas ett nytt *bpac*-document, där vissa attribut ska få värden innan det är möjligt att skriva ut en gästetikett rent programmatiskt. Dessa attribut är att välja skrivare, vilken mall

som ska användas, i detta fall mallen från figur 4:26. I figur 4:27 visas den kod som används för att skriva ut etiketten.

```
Document printLabelDoc = new Document();
printLabelDoc.SetPrinter(printerName, true);

if(printLabelDoc.Open(templatePath) != false)
{
    printLabelDoc.GetObject("FullNameText").Text = guestobj.FirstName + " " + guestobj.LastName;
    printLabelDoc.GetObject("CompanyText").Text = guestobj.Company;
    printLabelDoc.StartPrint("", PrintOptionConstants.bpoDefault);
    printLabelDoc.PrintOut(1, PrintOptionConstants.bpoDefault);
    printLabelDoc.EndPrint();
    printLabelDoc.Close();
}
```

Figur 4:27 : Kod för utskrift av etikett

Sedan ska de objektnamn som angavs tidigare användas för att skriva ut den text som nu är angivna i de objekten. Även detta visas i figur 4:27, där en if-sats undersöker om mallen är öppen i något program, för att sedan ange värde på respektive attribut innan en utskrift kan ske. Där ska även antalet etiketter för gästen att anges, vilket görs med hjälp av "dokumentnamn.PrintOut(1, PrintOptionConstants.bpoDefault);" där siffran 1 i detta fall anger ett endast en etikett för gästen ska skrivas ut. Efter det avslutas utskriften med "dokumentnamn.EndPrint();" samt att dokumentet sedan stängs med hjälp av "dokumentnamn.Close();".

4.5 Hårdvara

Detta avsnitt kommer att ge en kortfattad överblick över vilka hårdvarukomponenter som tillhandahålls med hjälp av uppdragsgivaren för att göra lösningen möjlig.

4.5.1 Surfplatta

Den surfplatta som används för att presentera surfplatteapplikationen är en Windows Surface Pro 2. Surfplattan har endast en USB-utgång, vilket används till ett tangentbord då etikettskrivaren ska skriva ut trådlöst. Detta då det inbyggda tangentbordet för surfplattan inte är praktiskt i samband med användning av applikationen. Detta är för att det inbyggda tangentbordet täcker halva skärmen så att en användare av applikationen inte ser de uppgifter som fylls i.

4.5.2 Etikettskrivare

För att göra det möjligt att skriva ut gästetiketter (se avsnitt 4.4) används en etikettskrivare i form av en *Brother QL-810W* [18]. Skrivaren har funktionalitet för att kunna skriva ut etiketter med hjälp av direkta termoutskrifter i färgerna svart, vit och röd.

För att kunna editera mallar för etiketter tillkommer tillbehör som *P-Touch Editor* och *bpac*. Det förstnämnda är en editor där det är möjligt att välja mellan olika föreslagna mallar eller skapa en helt egen och *bpac* är det *Software Development Kit* (SDK) som används för att kunna skriva ut etiketter även programmatiskt.

Skrivaren har funktionalitet för att kunna skriva ut både via en USB-kabel eller trådlöst.

5. Resultat

Detta kapitel redogör för resultatet av det arbete som gjorts. I avsnitt 5.1 beskrivs den hur den slutgiltiga lösningen för slutprodukten ser ut samt vissa skillnader mellan de olika applikationerna i lösningen. Avsnitt 5.2 beskriver hur målsättningen med arbetet följts samt hur funktionaliteten skiljer sig gentemot projektspecifikationen. Avsnitt 5.3 beskriver kortfattat de problem som uppstått under arbetet.

5.1 Slutprodukten

Utvecklingen av projektet har renderat i dels en skrivbordsapplikation som ska köras på en surfplatta där det är möjligt att checka in som antingen 1) ny gäst där gästen får ange sina uppgifter eller 2) checka in som eventdeltagare. Incheckningen för eventdeltagare fungerar annorlunda, där det ska vara möjligt att visa de deltagare som är registrerade för ett event under den dagen och sedan välja deltagare och sedan checka in. Detta är för att det inte ska bli kö i entrén då det är möjligt att flera deltagare anländer till kontoret samtidigt. Vid incheckningen kommer även gästen att få en gästetikett utskriven, där information som förnamn, efternamn och företag finns angivet. Denna etikett är möjlig att sätta på kläder för att sedan tas bort efter besök. Oavsett om en gäst besöker kontoret för att delta på ett event eller inte kommer utcheckningen att fungera på samma sätt, där gästen söker på de gäster som har checkat in under dagen men ej checkat ut och sedan välja vilken gäst som ska checkas ut. Efter arbetsdagens slut checkas de gäster ut som inte gjort det under dagen, vilket sker automatiskt. Utöver skrivbordsapplikationen har även en webbapplikation utvecklats för administrering av den data som registreras i skrivbordsapplikationen eller via webbapplikationen själv. Till skillnad från skrivbordsapplikationen är det möjligt att registrera ett event i webbapplikationen via en Excel-fil, dock kan inte en gäst checka in via webbapplikationen. Det är även möjligt att editera, visa eller radera information i webbapplikationen. Båda dessa applikationer är kopplade till en databas i Azure. För en mer översiktlig av hur komponenterna hänger ihop, se figur 2.1.

5.2 Målsättning

Målet med projektet var att modernisera det bemötande en gäst får vid besök av kontoret, vilket gör att projektet får anses lyckat. Under tiden för projektet har dock den funktionalitet som önskats varierat och vissa tekniker samt funktionalitet har prioriterats bort. Exempel på funktionalitet som valts bort är att en gäst välkomnas via ett röstmeddelande samt att kunna använda kameran på surfplattan för att kunna känna igen gäster som varit där tidigare eller en option som gällde ansiktsigenkänning för anställda, detta gjorde att inte Cortana Cognitive Services inte används.

5.3 Problem

Under utvecklingen av projektet har även en del problem uppstått. Vid uppstart av projektet användes en Azure-prenumeration från författarens privata mailadress, vilket endast var en tidsbegränsad lösning vilket senare gav problem då prenumerationen gick ut. Det renderade i att författarens mailadress från Sogeti fick göras tillgänglig för att senare kunna användas i Azure. Detta gjorde att en ny databas fick skapas för att sedan i koden kopplas till projektet. Ett annat problem som uppstod var hur surfplattan skulle kunna användas på nätverket, då det nätverk som används för mobila nätverk samt gästnätverket har vissa begränsningar som blockerar vissa portar som behöver användas vid databasanrop.

6. Slutsats

Detta kapitel redogör för utvärdering av projektet ur en icke-teknisk synvinkel. I avsnitt 6.1 beskrivs kortfattat hur uppdragsgivaren bidragit till arbetet. Avsnitt 6.2 beskriver hur det är möjligt att använda ett agilt arbetssätt vid mjukvaruutveckling. Avsnitt 6.3 ger en beskrivning av författarens tidigare relation till de ramverk och verktyg som användes samt de erfarenheter författaren tar med sig i framtiden. I avsnitt 6.4 beskrivs förslag på vidareutveckling av systemet.

6.1 Summering

Arbetet med projektet har gått över förväntan då arbetet fick en tuff start när den andra medlemmen några dagar innan projektstart inte längre var tillgänglig för att få vara en del av projektet. Trots de ändrade förutsättningarna nära inpå projektstart har samarbetet med företaget fungerat mycket bra redan från första början. För att stötta författaren fanns två handledare att tillgå för teknisk hjälp samt en projektledare som hjälpte till med hur arbetet skulle kunna läggas upp. Utöver detta har företaget även bidragit med arbetsdator, företagskonto, tillgång till Azure-konto samt Team Foundation Server för versionshantering samt backlog.

6.2 Arbetssätt

Under utveckling av projektet användes det agila arbetssättet SCRUM där perioden för arbetet delades upp i flera sprintar fördelat på tre veckor styck. För att hålla koll på vad som skulle göras användes tidigare nämnd backlog i Team Foundation Server. Där är det möjligt att lägga till olika "tasks" som är tänkt att implementeras under den sprinten. I slutet av varje sprint skulle sprintens arbete redovisas i form av en demonstration av de tasks som utförts.

Vid sidan av utvecklingen har uppsatsen uppdaterats kontinuerligt för att underlätta arbetsbördan mot slutet av projektet, vilket har visat sig vara en stor fördel.

6.3 Erfarenheter

Då arbetet utförts på egen hand har ett större ansvar fått tagits, vilket har vart utvecklande och en bra erfarenhet inför arbetslivet som väntar. Under utvecklingen

uppstod även chansen att testa och ta del av verktyg och tekniker som varit relativt nya samt helt nya för författaren. Detta då designmönstret för webbapplikationen, att utveckla en skrivbordsapplikation samt att använda sig av Azure var något som inte testats alltför mycket under studietiden. Detta har varit lärorikt och bra erfarenheter för framtiden då dessa tekniker används flitigt på diverse hemsidor och arbetsplatser.

Det finns även möjligheter till förbättring, vilka i detta fall är kopplade till att arbetet utfördes på egen hand. De retrospektiv som ska ske efter varje sprint blev inte lika givande då det inte fanns flera medlemmar i gruppen att diskutera med angående vad som gått bra eller dåligt under sprinten. Då en större arbetsbörda fått tagits av författaren har även arbetet under tidsperioden delvis varit mer än halvfart som det är tänkt. Författaren skulle därför rekommendera framtida studenter som gör examensarbeten att utföra arbetet i grupp, då det är möjligt att dela på den arbetsbörda som uppstår. Det möjliggör även för att diskutera och via samarbete kunna lösa problem på ett sätt som är svårare för en student som utför ett arbete på egen hand.

6.4 Framtida utveckling av systemet

För att utveckla systemet vidare i framtiden så finns det flera exempel på vad som skulle kunna göras. Under arbetets gång har fokus varit att systemet ska ha en fungerande backend, vilket gjort att designen på framförallt administrationsportalen skulle kunna förbättras. Utöver att skifta fokus till frontend skulle även mer funktionalitet kunna läggas till. Då funktionalitet under arbetets gång prioriterats bort skulle det vara möjligt att implementera vissa av dessa. Till exempel skulle kameran på surfplattan kunna användas som en sensor för att känna av om det står någon framför surfplattan. Sedan när en gäst checkat in skulle ett välkomstmeddelande kunna spelas upp. För att göra lösningen än mer energisnål skulle möjligheten att stänga av och sätta på etikettskrivaren programmatiskt kunna ses över, då etikettskrivaren endast behöver vara på då en gästetikett ska skrivas ut.

Referenser

- [1] Sogeti Sverige, Microsoft, <https://www.sogeti.se/varfor-sogeti/strategiska-allianser--partners/microsoft/> [Använd 2018-09-05]
- [2] Microsoft, Visual Studio IDE, <https://visualstudio.microsoft.com/vs/> [Använd 2018-09-05]
- [3] Microsoft, What is Face API?, <https://docs.microsoft.com/en-us/azure/cognitive-services/face/overview> [Använd 2018-09-05]
- [4] Wikimedia Commons, File: Cloud computing layers, https://commons.wikimedia.org/wiki/File:Cloud_computing_layers.png [Använd 2019-01-04]
- [5] Microsoft, Kognitiva tjänster, <https://azure.microsoft.com/sv-se/services/cognitive-services/> [Använd 2018-09-05]
- [6] Microsoft, Vad är Azure? , <https://azure.microsoft.com/sv-se/overview/what-is-azure/> [Använd 2018-09-06]
- [7] Microsoft, Team Foundation Server, <https://visualstudio.microsoft.com/tfs/> [Använd 2018-09-06]
- [8] Microsoft, XAML Overview (WPF), <https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/xaml-overview-wpf> [Använd 2018-09-10]
- [9] Microsoft, Data Binding Overview, [https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/ms752347\(v%3dvs.100\)](https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/ms752347(v%3dvs.100)) [Använd 2018-09-11]
- [10] Microsoft, WPF overview, <https://docs.microsoft.com/en-us/visualstudio/designers/introduction-to-wpf?view=vs-2017> [Använd 2018-09-11]
- [11] Microsoft, What is Version Control? , <https://docs.microsoft.com/en-us/azure/devops/learn/git/what-is-version-control> [Använd 2018-09-11]
- [12] Delaney, K. (2018). *Microsoft Azure Options for SQL Server Relational Databases*. Microsoft Corporation

- [13] Twilio, E.164, <https://www.twilio.com/docs/glossary/what-e164>
[Använd 2018-12-19]
- [14] IETF, Lightweight Data Access Protocol: The Protocol,
<https://tools.ietf.org/html/rfc4511> [Använd 2018-12-12]
- [15] Microsoft, Introduction to the C# Language and the .NET Framework,
<https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>
[Använd 2018-12-10]
- [16] National Institute of Standards and Technology (NIST), The NIST
Definition of Cloud Computing,
<https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf> [Använd 2018-09-24]
- [17] Microsoft, Entity Framework, <https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/overview> [Använd 2018-12-12]
- [18] Brother, QL 810W Specifikation,
<https://support.brother.com/g/b/spec.aspx?c=se&lang=sv&prod=lpql810weuk> [Använd 2018-12-04] [19] Microsoft Developer Network, LINQ,
<https://msdn.microsoft.com/en-us/library/bb308959.aspx> [Använd 2018-10-01]
- [20] Microsoft, SQL Server Management Studio,
<https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-2017> [Använd 2018-12-18]
- [21] Bootstrap, Bootstrap, <http://getbootstrap.com/> [Använd 2018-10-05]
- [22] Tutorialspoint, Design Patterns – MVC Pattern,
https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm
[Använd 2018-12-18]
- [23] Twilio, Programmable SMS, <https://www.twilio.com/sms> [Använd 2018-12-19]
- [24] Microsoft, Översikt över Active Directory Domain Services,
<https://docs.microsoft.com/sv-se/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview> [Använd 2018-12-18]

- [25] Microsoft, ASP.NET overview, <https://docs.microsoft.com/en-us/aspnet/overview>
[2018-10-08]
- [26] Microsoft, System.DirectoryServices Namespace,
<https://docs.microsoft.com/en-us/dotnet/api/system.directoryservices?view=netframework-4.7.2>
[Använd 2018-10-22]
- [27] Microsoft, ADO.NET overview, <https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ado-net-overview>
[Använd 2018-10-22]
- [28] Microsoft, What is .NET,
<https://www.microsoft.com/net/learn/dotnet/what-is-dotnet> [Använd 2018-10-22]
- [29] Howes, T. Smith, M (1997) LDAP Programming Directory-Enabled Applications with Lightweight Data Access Protocol. *Que Publishing*.
- [30] International Organization of Standards, International Organization of Standards, <https://www.iso.org/home.html> [Använd 2018-10-23]
- [31] Library of Congress, Codes for Representation of Names of Languages, https://www.loc.gov/standards/iso639-2/php/code_list.php [Använd 2018-12-12]
- [32] Organisation Internationale de Normalisation (ISO), ISO 3166-1 alpha-2 Country Codes, <https://www.iso.org/obp/ui/> [Använd 2018-12-12]
- [33] Beth Massi, Understanding .NET,
<https://blogs.msdn.microsoft.com/bethmassi/2015/02/25/understanding-net-2015/>
[Använd 2018-11-08]
- [34] Microsoft, Asynchronous programming, <https://docs.microsoft.com/en-us/dotnet/csharp/async> [Använd 2018-12-04]
- [35] Microsoft, What is NuGet, <https://docs.microsoft.com/en-us/nuget/what-is-nuget> [Använd 2018-12-04]

Bilagor

Bilaga 1: Lapp för besökare från pärm



Besökslista

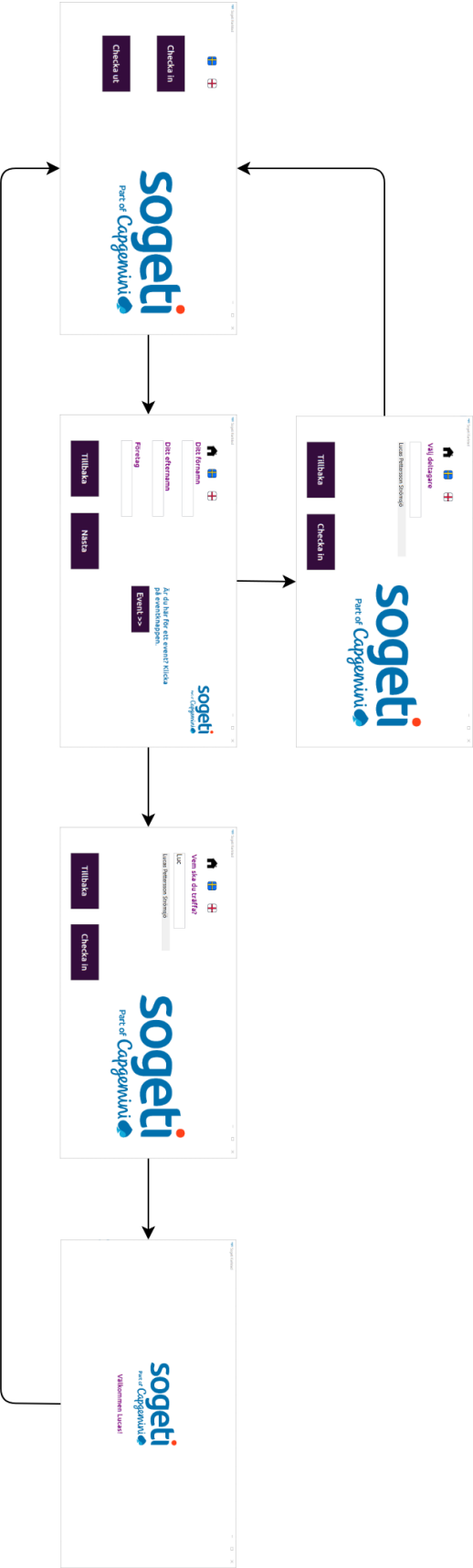
Välkommen till Sogeti. Du som besökare, vänligen skriv in dig och ta en besöksbricka.

Namn	Företag	Datum	Kom klockan	Gick klockan

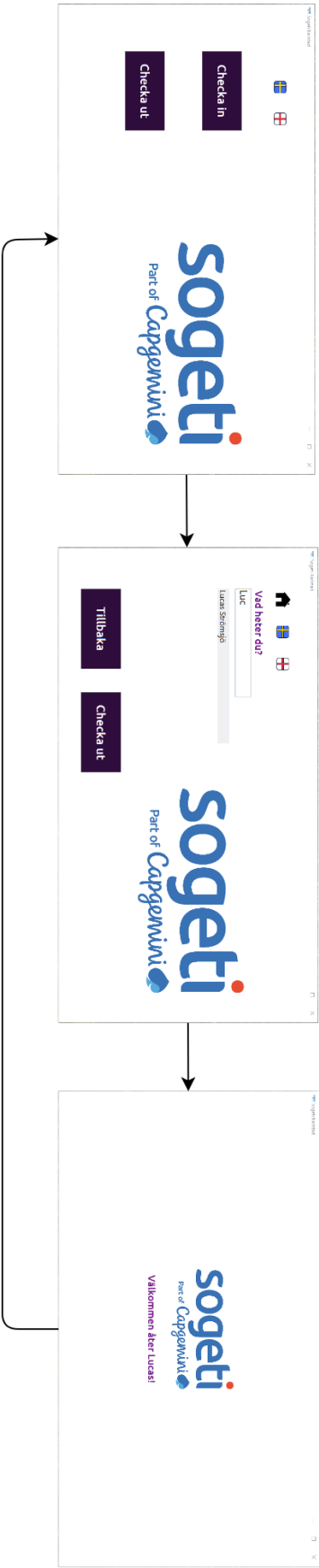
Bilaga 2: Arbetsflöde Receptionsapplikation



Bilaga 3: Arbetsflöde incheckning



Bilaga 4: Arbetsflöde utcheckning





Projektspecifikation

Receptionisten

Version 1.1

Befattning	Bolag/ enhet	Namn	Ät gärd	I nfo.
Student	KaU	Lucas Pettersson Strömsjö		
Sr Teknisk handledare	Sogeti	Thomas Heder		
Jr Teknisk handledare	Sogeti	Johan Olofsson		
Konsultchef	Sogeti	Åsa Maspers		
Projektleddarmentor	Sogeti	Magnus Karlsson		

Innehållsförteckning

1. Sogeti	3
1.1 Receptionisten – beskrivning	3
1.2 Tekniska krav	3
1.3 Optioner	3
1.4 Option – Ansiktsigenkänning för anställda	4
2. Genomförande/arbetssätt	4
2.1 Rutiner	4
2.2 Genomförande	4
2.3 Tidsplan	4
2.4 Arbetsmoment	5
3. Stöd/kvalitetssäkring	5
3.1 Granskningar	5
3.2 Testarbete	5
4. Leveranser	6
4.1 Dokumentation	6
5. Konfigurationsstyrning	6
6. Miljö	6
7. Uppföljning och Rapportering	6
7.1 Rapportering internt/externt	6
7.1.1 Statusrapportering	6
7.1.2 Möten	6
7.1.3 Slutrapportering	6

1. Sogeti

Sogeti är Sveriges ledande leverantör av tjänster och lösningar när det gäller Azure och övriga

Microsoft-produkter, och vi innehar högsta partnerstatus med Microsoft med Guldstatus inom 14 olika kompetenser. För mer info in Sogeti, se www.sogeti.se.

1.1 Receptionisten – beskrivning

Uppdraget går i första hand ut på att bygga en digital reception, där gästen skall kunna skriva in diverse kontaktinformation och vem ska besökas, systemet skall kunna skicka en notifiering till den som skall besökas. Applikationen skall lämpligen köras på någon form av surfplatta som står i anslutning till entrén.

Vi vill att receptionen skall byggas i form av en "virtuell receptionist" som välkomnar besökare, snarare än en statisk, "tråkig" websida. Man kan tänka sig att receptionisten känner av när en människa står framför den, och då med hjälp av tal välkomnar, och till viss del förstår röstmeddelanden som besökaren säger.

Som option tänker vi oss att receptionisten med hjälp av kamera kan komma ihåg och känna igen besökare som återkommer.

1.2 Tekniska krav

Följande är tekniska krav som finns på applikationen:

- Applikationen ska byggas med Microsoft-teknologi, endera som webb med MVC som back-end eller som Windows applikation med WPF eller som en Windows 10 Store App. Väljer man att bygga det som webb är det valfritt om man vill använda standard MVC eller blanda in ramverk som t.ex. React eller Angular för front-end.
- Applikationen ska så långt det går använda tjänster från Cortana Cognitive Services <https://azure.microsoft.com/en-us/services/cognitive-services/>

1.3 Optioner

Följande är förslag på vidareutveckling av detta som uppdragstagarna själv får plocka från om tid finns. Innan uppdragstagarna börjar med någon option så måste ett möte till för att prioritera dessa.

1.4 Option – Ansiktsigenkänning för anställda

Applikation ska kunna kopplas till en kamera och genom att använda t.ex Cortana Cognitive Services ska besökaren kunna analyseras.

Applikationen skall kunna analysera om besökaren är just en besökare eller anställd.

Som en grund för uppdraget räcker det med om applikationen klarar av att detektera en person i taget som passerar framför kameran. Men applikationen skall inte analysera samma person flera gånger när hen passerar kameran. Även kunna undersöka hur detta är möjligt kontra GDPR samt om det skett några överträdelser under utvecklingen och i så fall varför.

2. Genomförande/arbetssätt

2.1 Rutiner

Sogeti tillhandahåller arbetsplatser, datorer, hårdvara samt erforderliga utvecklingsverktyg.

Uppdragstagarna kommer att ha access till Sogetis nätverk och förväntas nyttja vår TFSserver för versionshantering.

2.2 Genomförande

Uppdragstagarna planerar själva genomförandet och Sogeti tillhandahåller stöttning både projektstyrningsmässigt och rent implementationstekniskt. Sogeti tillhandahåller all programvara och hårdvara som behövs.

Under utvecklingen kommer en variant av Scrum att användas, där det kommer att vara demonstration av funktionalitet som utvecklats under sprinten. Dessa möten kommer att vara var tredje vecka och är planerade att vara i en timme per gång där uppdragstagare, rekryteringschef, projektledare samt två handledare kommer att delta. Utöver detta så kommer även en timme retrospectives att användas.

2.3 Tidsplan

Under sprintarna kommer halva tiden för examensarbetet att användas för att skriva på rapporten och den andra halvan kommer att gå till utvecklingen av projektet. Under den

första sprinten så kommer mycket tid att läggas på att komma igång med projektet, med allt från vilken sorts applikation som ska användas till att få tillgång till Azure samt att lägga upp en databas i molnet.

Under de följande sprintarna kommer mer fokus att kunna lägga på utveckling av projektet där en vidareutveckling på tidigare sprintar görs. Under de sprintarna ska bland annat notifieringar till anställda som fått besök, checka ut personer automatiskt om de glömt bort, jämföra hur applikationen står sig lagligt med GDPR samt i mån av tid finns även en option med där det ska vara möjligt att identifiera anställda samt återkommande gäster vid användande av applikationen.

Vid senare sprintar så kommer än mer fokus att läggas på att få till skrivandet av uppsatsen samt analysera hur arbetet har fortlöpt under examensarbetet.

2.4 Moment

I början av projektet kommer fokus att ligga på grundläggande saker som att få tillgång till Azure, där en databas ska läggas till och sedan designas efter behov för uppdragsgivaren. Nästa steg är att börja utvecklingen av applikationen samt den grundfunktionalitet som kommer att krävas för att till exempel en gäst ska kunna checka in och ut.

Applikation kommer att vara en Windows Presentation Foundation (WPF)-applikation där större delen av utveckling kommer att ske i C#, men delar av SQL kan även förekomma.

3. Stöd/kvalitetssäkring

3.1 Granskningar

Vid behov genomförs granskning som kan initieras av både handledare och uppdragstagare.

Lämpligen definieras några granskningspunkter vid planeringen av projektet.

3.2 Testarbete

Funktions-, system- och integrationstest görs av uppdragstagarna.

4. Leveranser

4.1 Dokumentation

Systemdokumentation görs av uppdragstagarna. Dokumenten lagras i projektarkiv hos Sogeti.

5. Konfigurationsstyrning

All programkod och tillhörande specifikationer och andra utvecklingsdokument ska versionshanteras med hjälp av Microsoft TFS.

6. Miljö

Utvecklingsmiljö kan lämpligen vara Visual Studio 2017.

7. Uppföljning och Rapportering

7.1 Rapportering internt/externt

7.1.1 Statusrapportering

Rapportering av status och framskridande i utvecklingen beslutas i samråd vid projektuppstart.

7.1.2 Möten

Möten hålls vid behov. Vid uppstart läggs lämpligt antal avstämningsmöten in i projektplanen.

7.1.3 Slutrapportering

Arbetet presenteras för Sogeti i samband med lämpligt månadsmöte alternativt lunchmöte.

