RADIX

En administrativ webbapplikation för VACS

RADIX
An administrative web application for VACS

Benjamin Memic
Adam Payne

Fakulteten för hälsa, natur- och teknikvetenskap

Datavetenskap

Examensarbete 15 hp

Handledare: Donald F. Ross

Examinator: Thijs Jan Holleboom

Opositionsdatum: 2016-01-21

Sammanfattning

Företaget Evry Forest & Logistics Solutions AB i Karlstad har utvecklat systemet VACS (Virkes Administrativ Client/Server) som administrerar virkesaffärer för skogsindustrin. Eftersom kundbasen ständigt växer behöver Evry ett system där alla kunders produktkonfigurationer finns samlade på ett system och kan nås via ett web-baserat interface.

Denna uppsats beskriver projektet Radix. Projektets huvudsakliga syfte var att skapa en administrativ plattform som hanterar de kunder som använder VACS.

Projektet har resulterat i en användarbaserad webbapplikation med ett tillhörande program som ska fungera som ett hjälpmedel för att hitta en kunds produktkonfiguration i VACS källkod.

Abstract

The company Evry Forest & Logistics Solutions AB in Karlstad has developed the VACS system (Lumber Administrative Client/Server) that manages the timber business for the forest industry. As the customer base is constantly growing Evry needs a system where all customer’s product configurations are gathered in one system and can be accessed via a web-based interface.

This dissertation describes the Radix project. The project's main aim was to create an administrative platform that manages customers who use VACS.

The project has resulted in a user-based web application with an associated program that will serve as a tool to find a client's product configuration in VACS source code.

Förord

Denna uppsats är skriven vid Karlstad Universitet under höstterminen 2015 samt första två veckorna av 2016. Projektet har utförts hos Evry Forest & Logistic Solutions AB i Karlstad. Det har varit ett intressant, lärorikt, roligt och intensivt arbete. För att genomföra arbetet har vi varit beroende av flera personer vilka vi vill uttrycka vår tacksamhet till.

Främst vill vi rikta ett stort tack till två personer som betytt extra mycket för vårt arbete under höstterminen. Dessa är Torbjörn Stake på Evry som varit vår handledare för projektet samt vår handledare för uppsatsskrivandet, Donald F. Ross.

Vi vill även tacka personal på Evry Forest som gett oss goda tips under projektets gång.

Innehållsförteckning

Figurer

1 Introduktion

Sverige är världens tredje största exportör av sågade trävaror. Idag uppgår skogsindustrins andel av Sveriges varuexport till tolv procent och förväntas öka markant som följd av global konkurrens [1].

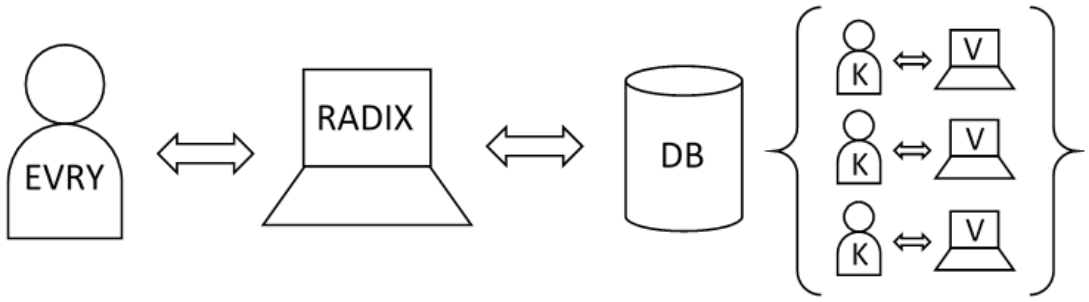
Evry [1] bidrar till att öka skogsindustrins konkurrenskraft med sitt administrativa system VACS (Virkes Administrativ Client/Server) [2]. Med över 30 kunder är VACS det ledande systemet för skogsråvara. Systemet hanterar allt från värdering och anbudslämnande till kontraktering och planering av verkningsuppdrag.

I takt med att kundbasen ökar vill Evry hitta en lösning för att hantera och hämta information om deras kunders produktkonfiguration. I denna uppsats behandlas projektet Radix som är en administrativ webbapplikation för att hantera denna information.

1.1 Syfte

I dagens läge görs för ekonomiskt syfte en årlig genomgång av VACS kundbas. Information om varje kunds produktkonfiguration sammanställs genom att göra ingående analyser av kravspecifikationer, så kallade product backlog's [3], samt källkoden för VACS. Informationen sammanställs sedan i Excel [4].

Syftet med detta projekt har varit att skapa ett informationssystem där en användare ska kunna logga in och snabbt kunna bilda sig en uppfattning om en kunds produktkonfiguration. En användare ska även kunna hantera information om kunder samt andra viktiga produktdelar. Detta ska göras med en webbapplikation som är fullt möjlig att använda från mobila verktyg oavsett vart personalen befinner sig. Webbapplikationen ska vara användarbaserad och på så sätt hjälpa Evry att minska personberoenden, där nyckelpersoner är ensamma att besitta viss information.



Figur 1.1 Systemet Radix. K = Kund, V = VACS.

Eftersom webbapplikationen fortfarande kräver manuell inventering av VACS källkod för att kunna skapa informationen är syftet även att skapa ett extraheringsprogram som automatiskt läser igenom källkoden för VACS och hämtar informationen om kundernas produktkonfiguration. Detta ska hjälpa till att minska tiden för inventeringen.

Projektets mål har varit att skapa en webbapplikation som kan tas i bruk vid projektets slut samt att skapa en prototyp av extraheringsprogrammet.

1.2 Disposition

Nedan följer en disposition av uppsatsen.

Kapitel 1 - Introduktion

I detta kapitel introduceras företaget Evry Forest & Logistics Solutions AB samt syftet med projektet Radix.

Kapitel 2 – Bakgrund

I detta kapitel beskrivs Evrys administrativa system VACS. Kapitlet innehåller även en beskrivning av problemformuleringen samt en detaljerad beskrivning av kravspecifikationen för projektet Radix. Slutligen beskrivs tekniker och verktyg som har använts under projektet.

Kapitel 3 – Design

I detta kapitel beskrivs designen av webbapplikationen, extraheringsprogrammet samt tillhörande databas.

Kapitel 4 – Implementation

I detta kapitel förklaras implementationen av de centrala delarna i webbapplikationen samt extraheringsprogrammet.

Kapitel 5 – Resultat och utvärdering av verktyg

I detta kapitel presenteras resultatet samt kravuppfyllelsen av projektet. I kapitlet ges även en förklaring av testning. Slutet av kapitlet innehåller en utvärdering av tekniker och verktyg som har använts under projektets gång.

Kapitel 6 – Utvärdering av projekt

I detta kapitel redogörs projektets tidsåtgång, arbetsmiljö, kundkontakt, kravspecifikation, lärdomar samt möjligheter till vidareutveckling. I detta kapitel presenteras även en slutats för hela projektet.

2 Bakgrund

I detta kapitel beskrivs projektet Radix samt de tekniker som har använts för att utveckla webbapplikationen och extraheringsprogrammet. För att få en bättre förståelse om varför projektet skapats beskrivs företaget Evry Forest & Logistic Solutions och deras produkt VACS.

2.1 EVRY Forest and Logistic Solutions

Evry Forest & Logistic Solutions AB tillhör koncernen Evry, Nordens näst största IT-tjänsteföretag med ca 10000 anställda utspridda på 50 kontor runtom i Norden [1].
På kontoret i Karlstad arbetar Evry Forest & Logistic Solutions för att utveckla effektiva verksamhetsprocesser och IT-lösningar åt skogsindustrin [5].

2.2 VACS

VACS [2] står för Virkes Administrativ Client/Server. Det är ett administrativt system för skogsråvara och är basen i EVRYs skogliga affär. VACS erbjuder alla former av administrativ behandling av virkesaffärer. Kunderna erbjuds information i form av en prisräkning som stöd för fakturering och avräkning vid köp och försäljning av skog.

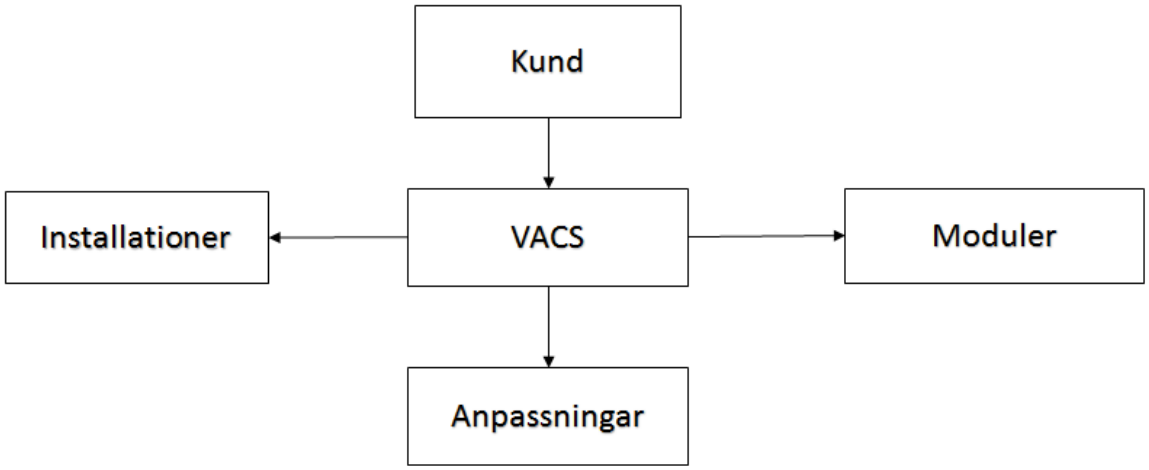
VACS uppdateras kontinuerligt och det släpps nya versioner allt eftersom, vanligtvis två gånger om året. Om kunden väljer att uppgradera till en ny version har den möjligheten att köpa till funktionalitet som är tillgänglig för den nya versionen. Det finns inget krav på kunden att uppgradera till en nyare version. Kunden får dock bara underhållstöd för två versioner tillbaka.

VACS systemet består av flera installationer. En installation är en konfiguration av VACS där olika installationer används till olika typer av uträkningar på volymen avverkad skog. Som exempel räknar installationen Rundvirke ut hur mycket virke man kan få ut av stamvolymen i en avverkning medan installationen Energi istället räknar ut hur många Megawattimmar energi man får under avverkningen. Kunden använder sig vanligtvis av en enda installation men kan i vissa fall ha två. En kund måste dock använda minst en installation.

En kund kan också behöva olika anpassningar som är skräddarsydda för kundens ändamål. En anpassning kan vara exempelvis att kunden vill ha en annan typ av räknefunktion utöver den standardfunktion som systemet redan innehåller. Det kan även vara att kunden vill ha en extra

räknefunktion. En anpassning kan ses som en mindre typ av modifiering av de standardfunktioner som redan finns i systemet samt ett mindre tillägg. Nya anpassningar som kunder har köpt finns tillgängligt i nästa version av VACS och kan endast användas av de kunder som köpt anpassningarna.

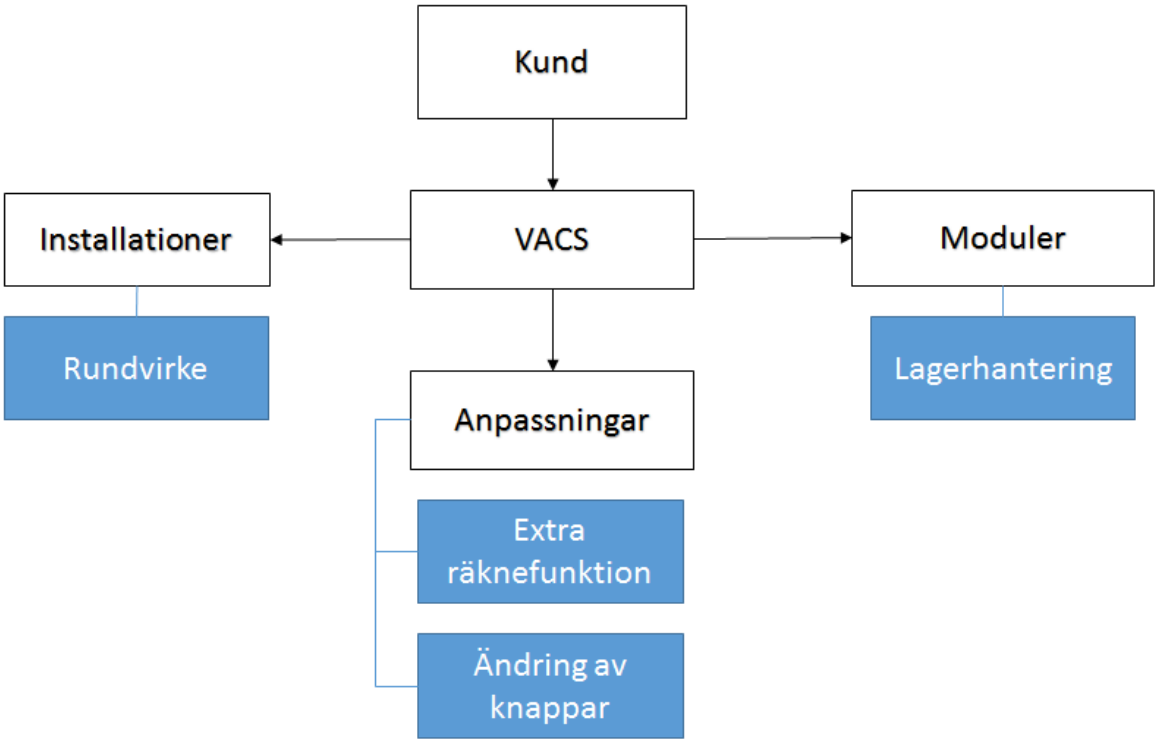
Förutom anpassningar har VACS fått ett antal delsystem eller moduler utvecklade under systemets livscykel. Dessa moduler är mer omfångsrika och kapslar lite större arbetsflöden eller funktionsgrupper. Modulerna är fristående från standardsystemet. Beroende på kundens behov kan denne ha en eller flera moduler kopplade till systemet, men det är inget krav.



Figur 2.1 VACS relationer

En typisk kund till VACS använder ett standardsystem med tillhörande installationer, moduler och anpassningar som är behovsanpassade till den specifika kunden, se Figur 2.1.

En kund kan exempelvis använda sig av VACS version 4.6 med tillhörande installation rundvirke och ha en modul för lagerhantering. Till detta har kunden också ett antal anpassningar på systemet, bl.a. en extra räknefunktion för installationen rundvirke som inte ingår i grundversionen samt några små ändringar på färg och storlek för knappar. Se Figur 2.2 nedan för en illustration av den typiske kund som beskrivits ovan.



Figur 2.2 En typisk kund, VACS

Här har kunden valt att använda sig av den senaste versionen av VACS för att ha tillgång till alla de senaste anpassningarna, installationerna och modulerna samt bibehålla support på systemet. Kunden har också valt installationen rundvirke eftersom det är den virkestyp som används. Till detta har kunden lagt till en del anpassningar som gör att programmet blir mer skräddarsytt för kundens önskemål och användningsområden.

2.3 Problemformulering

Evry Forest har en nuvarande kundbas på ca 30 kunder för sitt system VACS. Eftersom antalet kunder ständigt ökar krävs det att man håller ordning på dessa. Detta är viktigt ur flera perspektiv, se bilaga A.2.2.

1. Att veta vilken kund som har installerat anpassningar och vilka de är, är viktigt för att faktureringen av den årliga underhållsavgiften ska bli rätt.
2. Att veta vilken kund som köpt vilka moduler är viktigt för att faktureringen av den årliga underhållsavgiften ska bli rätt.
3. Att veta vilken kund som har anpassningar, moduler och vilken version av VACS är viktigt för utvecklare och produktägare eftersom det får konsekvenser i utvecklingscykeln av VACS och kvalitetssäkrande åtgärder.

Varje år sammanställs denna information manuellt genom att gå igenom en product backlog och manuellt leta i VACS källkod för att identifiera anpassningar för en specifik kund. Detta tar oftast ett par månader att göra. Informationen sammanställs sedan på lokala Excel listor. För att underlätta detta arbete krävs ett bättre verktygsstöd. Det är detta som har utvecklats i projekt Radix.

2.4 Radix

Som tekniskt krav begärde uppdragsgivaren att Radix skulle skapas som en webbapplikation för att undgå en programinstallation på PC eller mobil enhet, se bilaga A.4.1.

De grundläggande funktionella kraven för Radix var att hämta och hantera information om VACS kunder och produktkonfigurationer. Med produktkonfiguration menas vilken version av VACS kunden använder, vilken installation som används samt vilka anpassningar och moduler som kunden har köpt utöver standardsystemet, se Figur 2.1.

I uppdraget fanns det en förfrågan om att denna information endast ska kunna användas av en autentiserad användare, se bilaga A.4.1. Det ska också finnas olika roller för en användare, en roll med administratörsrättigheter och en roll där man inte kan hantera andra användare. Användaren ska efter inloggning tas direkt till en instrumentpanel där vidare navigering till någon av systemets dimensioner kan ske. Dimensioner i detta fall är kund, VACS version, installation, modul och anpassning.

Efter valet av dimension är det första man ska mötas av en tabell som visar alla instanser av dimensionen. Användaren ska sedan kunna skapa/ändra eller ta bort en instans.

Vid skapande av kund ska man kunna välja namn på kunden samt vilken VACS-version kunden använder. Här ska man också kunna välja vilken installation kunden använder, samt de anpassningar och moduler som kunden köpt. Relationerna mellan dessa ska automatiskt kopplas i databasen.

Vid skapande av anpassning ska man kunna välja namn på anpassningen samt ange en beskrivning av den. Här ska man även kunna ange i vilken VACS-version anpassningen skapades i samt i vilka VACS-versioner anpassningen finns tillgänglig i. Utöver det ska man kunna ange hur mycket anpassningen kostade att utveckla samt i vilka klasser koden för anpassningen finns i VACS källkod.

För att skapa en VACS version ska man ange namn och datum skapad. Versionen ska sedan kunna kopplas till en kund eller en anpassning när en ny skapas eller en befintlig redigeras. Uppdragsgivaren har under projektets gång specificerat att det inte ska finnas några relationer mellan en installation och version samt modul och version i Radix, eftersom detta kan medföra en onödigt komplicerad datamodell.

För de resterade dimensionerna installation och modul skapar man en instans genom att endast ange ett namn. Denna instans kopplas sedan till en kund när en ny kund skapas eller en befintlig kund redigeras.

Under tiden projektet har utvecklats har nya förfrågningar om funktionalitet utöver det som ursprungligen fanns i kravspecifikationen tillkommit från uppdragsgivaren. En autentiserad användare ska kunna söka efter t.ex. en specifik version av VACS. Sökning utan begrepp ska visa alla versioner. Sökning med helt eller delvis namn t.ex. ”4.” ska ge träff på ”4.1 – 4.9”.

Uppdragsgivaren har även ställt ett antal förfrågningar på användarvänlighet. Ett av dem är att instanserna ska vara klickbara så att relationerna till vald instans fälls ut i direkt anslutning till raden. För varje relation skall hyperlänkar finnas mellan systemets dimensioner. Vart man än befinner sig i systemet ska man kunna navigera till en annan dimension av systemet utan att behöva gå tillbaka till dashboard, d.v.s. instrumentpanelen. Man ska alltså kunna klicka på de utfällda relationerna och då skickas till den dimensionen av webbapplikationen där relationen är en instans i relevant tabell.

Eftersom antalet instanser av en dimension kan bli många till antalet bör samtliga tabeller delas upp i flera sidor för att minimera scrollande.

Utöver de grafiska kraven om instrumentpanel ska Radix utvecklas tile-baserat för att likna webbsidor så som Visual Studio Online [6] eller Dyn AX [7]. Radix bör även stödja responsivitet [8] och kunna köras på ett användarvänligt sätt även från en mobil enhet, se bilaga A.4.1.

I en utökning till uppdraget fanns också ett önskemål att en användare ska kunna få all denna information inladdad i en databas per automatik genom att söka igenom källkoden för VACS och identifiera entiteterna kund och anpassning, se bilaga A.5. Till detta skulle det skapas ett separat system och inte nödvändigtvis behöva nås genom Radix.

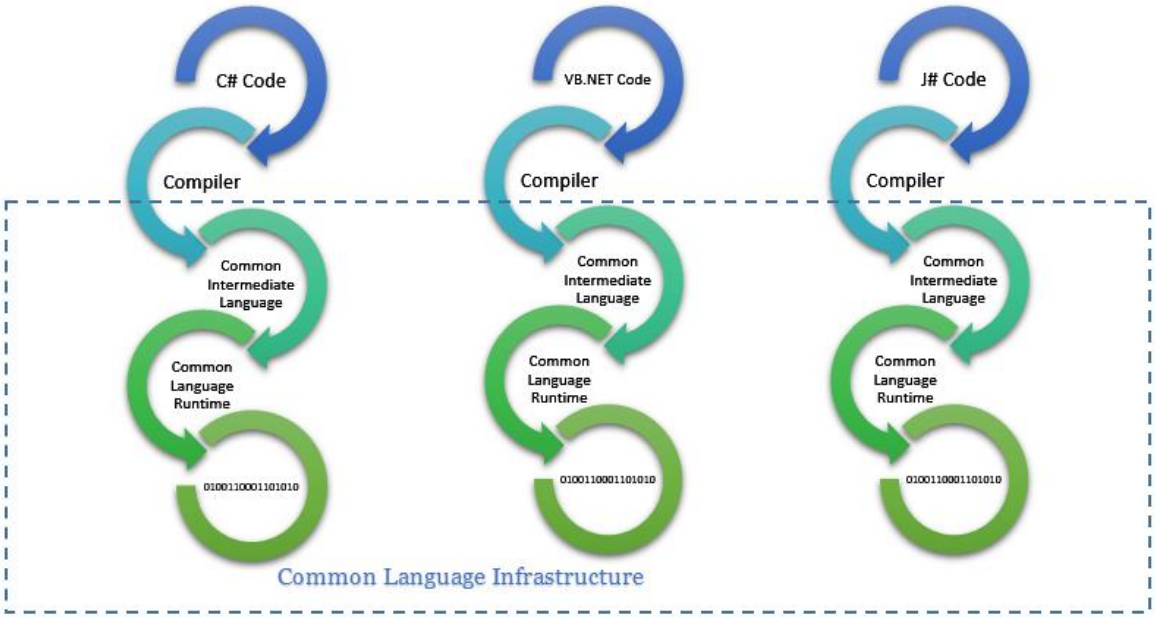
2.5 Tekniker

I detta avsnitt presenteras tekniker som har använts för att skapa projektet. Eftersom produktägaren rekommenderade att använda Microsofts plattform har denna också använts.

2.5.1 NET Framework

.NET [9] är ett ramverk som hanterar applikationer skrivna specifikt för .NET ramverket. De två centrala beståndsdelarna är CLR (Common Language Runtime) [10] och ett omfattande klassbibliotek [11]. CLR är en run-time miljö som exekverar koden och tillhandahåller minneshantering, garbage collection samt andra systemtjänster. Miljön har även stöd för cross-language integration, som betyder att man kan skapa projekt med flera olika språk som kan kommunicera med varandra. Detta kan göras eftersom .NET språk först kompileras till ett plattform-neutralt CIL (Common Intermediate Language) [12] språk och sedan kompileras CIL koden till maskin kod av CLR.

CLR är en implementation av CLI (Common Language Infrastructure) standarden som är specifikation på hur man implementerar en exekverings miljö [13].



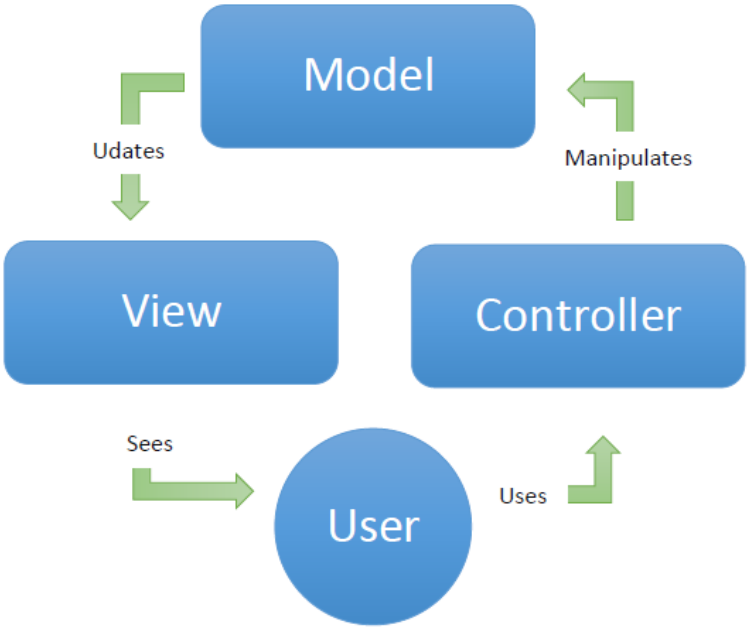
Figur 2.3 CLR

2.5.2 ASP.NET

ASP.NET [14] är ramverk utvecklat av Microsoft och är baserat på .NET Framework. ASP.NET används för att skapa dynamiska webbapplikationer, webbsidor samt webbtjänster. Visual Studio [15] skapar kärnan av alla projekt-filer som behövs för att köra applikationen [16] direkt. Därefter kan man modifiera och utöka applikationen utefter önskemål.

2.5.3 MVC

Model-View-Controller [17] är ett designmönster inom systemutveckling som separerar presentationslagret från datahanteringen och lägger till ett Controller lager som fungerar som en mellanhand till View och Model. Detta görs för att inte datahanteringen skall påverkas av förändringar i presentationslagret och att data kan hanteras utan att behöva göra ändringar i presentationslagret [18].



Figur 2.4 MVC

Model innehåller all logik och data för systemdomänet. Ofta används Model-objekt till att hämta, ändra och skriva data från och till databasen.

View är oftast ett användargränssnitt. Det är till för att representera funktionaliteten i Model på ett användarvänligt sätt.

Controller hanterar och svarar på användar-inputs och interaktioner, samt framkallar ändringar i Model och View.

2.5.4 ASP.NET MVC Framework

ASP.NET MVC Framework [19] bygger på ASP.NET men använder sig av designmönstret Model View Controller som beskrivits ovan där ingen logik är bunden till det grafiska gränssnittet. Ramverket används vid skapande av dynamiska webbsidor där innehållet på sidan kan förändras.

Fördelar med ASP.NET MVC till skillnad från ASP.NET är att man som utvecklare får full kontroll över den genererade HTML-koden [20] samt att det blir lättare att testa och återanvända arkitekturen.

2.5.5 Azure

Azure [21] är Microsofts molnplattform för att bygga och hosta webbapplikationer via ett datacenter. Plattformen innehåller en stor samling integrerade tjänster, för analys, databearbetning, databas, mobil, lagring och webb.

Azure stöder också nästan alla operativsystem, programmeringsspråk, ramverk, verktyg, databaser och enheter vilket gör det lättare för utvecklare då de kan använda sig av samma tekniker som innan [22].

2.5.6 Entity Framework

Entity Framework [23] är en Object Relational Mapping [24] som ger utvecklare en möjlighet att automatisera åtkomst och lagring av data i databasen. Ramverket kan användas på tre olika sätt.

Om en databas redan existerar eller om man vill designa databasen före andra delar av applikationen, skapar ramverket automatiskt klasser för den existerande databasen och kopplar ihop dem med databastabeller.

Vill man istället fokusera på klasser först och sedan skapa databasen från klasserna använder man sig av Code First tekniken. Med hjälp av Code First skapas en databas och tabeller som motsvarar klasserna. Finns databasen redan så skapas endast tabellerna.

Sista utförandeformen är att designa databasschemat med Entity Data Model Designer och sedan skapa databaser och klasser [25].

2.5.7 Bootstrap

Bootstrap [26] är det mest populära HTML, CSS [27] och Javascript [28] ramverket för att utveckla responsiva mobile-first projekt på webben. Med Bootstrap går front-end utveckling snabbt och enkelt oavsett kompetensnivå, skärmstorlek och omfattning av projektet. Ramverket innehåller många skräddarsydda HTML, CSS komponenter och Jquery plugins [29].

Bootstrap är installerat i Visual Studio och ASP.NET MVC som standard [30].

2.6 Implementationsverktyg

I denna del beskriver vi de implementationsverktyg vi har använt oss av under projektets gång.

2.6.1 Visual Studio 2015

Visual Studio 2015 [15] är en utvecklingsmiljö utvecklad av Microsoft. Med Visual Studio kan man skapa program, webbapplikationer, webbsidor samt webbtjänster. Utvecklingsmiljön har stöd för de inbyggda programmeringsspråken C [31], C++ [32], C# [33], VB.NET [34] samt F# [35] . Stöd för andra språk tillsätts med hjälp av ett tillägg som kallas för Languange Service.

2.6.2 Visual Studio Online

Visual Studio Online är en samling molnbaserade tjänster som är kopplade till Visual Studio. Visual Studio Online använder sig av metodiken Scrum [3] där får man en överblick av arbetet som skall göras. Några av tjänsterna är att teamet skapar en digital planeringstavla med backlog items, delar upp projektet i sprintar, skriver in hur många timmar per vecka dem är tillgängliga att arbeta med projektet mm.

Utvecklarna kan även lagra kod i privata förvaringsplatser som andra i teamet kan hämta. Mellan förvaringsplatsen och Visual Studio används olika versionshanteringsprogram, såsom Git [36] och Team Foundation Server [37], för att få ut det mesta av samarbetet.

2.6.3 Git

Git är ett distribuerat versionshanteringsprogram som hanterar såväl små som väldigt stora projekt [38]. Git kan anslutas till Visual Studio vilket gör att man kan versionshantera från samma program som man arbetar med projektet i. I Git sker alla operationer lokalt och behöver inte kommunicera med någon server [39].

2.7 Sammanfattning

Detta kapitel har gett en bakgrundsbeskrivning till företaget Evry som är Nordens näst största IT-tjänsteföretag. Kapitlet innehåller även en beskrivning av VACS-systemet och alla systemets delar såsom installation, modul samt anpassning. För att få en bättre förståelse för uppdraget har även problemformuleringen till projektet framförts. Efter problemformuleringen har uppdraget Radix beskrivits. Slutligen beskrivs också de tekniker och verktyg som använts.

3 Design

I detta kapitel introduceras designen av Radix. Delkapitel 3.1 beskriver databasdesignen och ger en överblick av databasen. Senare i delkapitlet beskrivs även de tabeller som databasen består av. Delkapitel 3.2 presenterar användargränssnittet för Radix, hur det är uppbyggt, hur det hänger ihop samt vilka likheter och olikheter som finns i webbapplikationen. Delkapitel 3.3 redogör för dynamiken av användargränssnittet. I slutet av kapitlet beskrivs designen av extraheringsprogrammet VACS Extract.

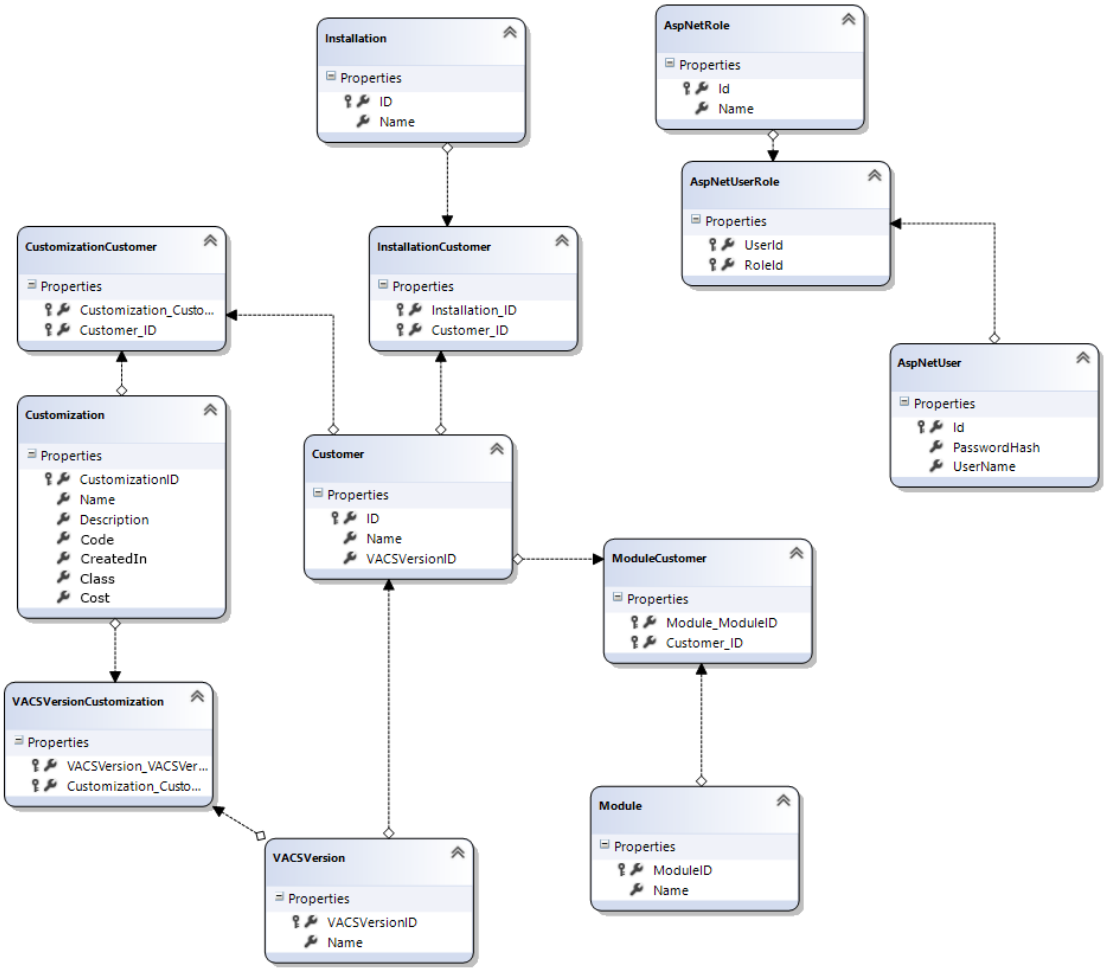
3.1 Databasen

En databasmodell har tagits fram med hjälp av uppdragsgivaren för att få med alla relationer mellan kund och VACS. Databasen används både av webbapplikationen och det externa programmet för extrahering av anpassningar ur källkoden för VACS.

Databasen innehåller tolv tabeller och implementerades med hjälp av Entity Framework [23]. Entity Framework hjälper till att automatiskt skapa tabeller utefter en datamodell. Datamodellen består av ett antal entiteter som motsvarar tabeller i databasen. Dessa entiteter är kund, installation, modul, anpassning, version samt användare.

3.1.1 Design

Nedan visas en överblick av databasen i Radix i form av ett UML-diagram. Här presenteras alla tabeller i databasen och relationerna mellan dem.



Figur 3.1 Databas design

3.1.2 Databastabeller

Nedan följer en beskrivning av alla tabeller i databasen.

3.1.2.1 AspNetUser

Tabellen AspNetUser innehåller uppgifter om användare som har tillgång till Radix. Där finns id, användarnamn och lösenord sparat för varje användare.

3.1.2.2 AspNetRole

Tabellen AspNetRole innehåller de olika roller som en användare av Radix kan ha, d.v.s. Admin för administratör och User för vanlig användare.

3.1.2.3 Customer

Tabellen Customer innehåller uppgifter om alla kunder som använder VACS. Där finns namn och främmandenyckel till VACS-version.

3.1.2.4 Installation

Tabellen Installation innehåller namnen på de olika installationerna.

3.1.2.5 Customization

Tabellen Customization innehåller information om de olika anpassningarna. Där finns information som namn, beskrivning, kod, vilken version av VACS den skapades i, vilken klass den finns med i, samt kostnaden för anpassningen.

3.1.2.6 Module

Tabellen Module innehåller namn på de olika modulerna.

3.1.2.7 VACSVersion

Tabellen VACSVersion innehåller namn på de olika VACS-versionerna.

3.1.2.8 InstallationCustomer

Tabellen InstallationCustomer innehåller kopplingen mellan kunder och installationer. Denna tabell finns med för att skapa en många till många relation. En kund kan ha flera installationer och en installation kan användas av flera kunder.

3.1.2.9 CustomizationCustomer

Tabellen CustomizationCustomer innehåller kopplingen mellan kunder och anpassningar. Denna tabell finns med för att skapa en många till många relation. En kund kan ha flera anpassningar och en anpassning kan användas av flera kunder.

3.1.2.10 ModuleCustomer

Tabellen ModuleCustomer innehåller kopplingen mellan kunder och moduler. Denna tabell finns med för att skapa en många till många relation. En kund kan ha flera moduler och en modul kan användas av flera kunder.

3.1.2.11 VACSVersionCustomization

Tabellen VACSVersionCustomization innehåller kopplingen mellan VACS-versioner och anpassningar. Denna tabell finns med för att skapa en många till många relation. En VACS-version kan ha flera anpassningar och en anpassning kan användas av flera VACS-versioner.

3.1.2.12 AspNetUserRole

Tabellen AspNetUserRole innehåller kopplingen mellan användare och roller. Denna tabell finns med för att skapa en många till många relation. En användare kan ha flera roller och en roll kan användas av flera användare.

3.2 Användargränssnittet

En vy är ett användargränssnitt som representerar data och relationer i databasen utefter en modell.

De grundläggande funktionella kraven för Radix är att hämta och hantera information om VACS kunder och deras produktkonfigurationer. Denna information har valts att presenteras med hjälp av tabeller i ett antal vyer.

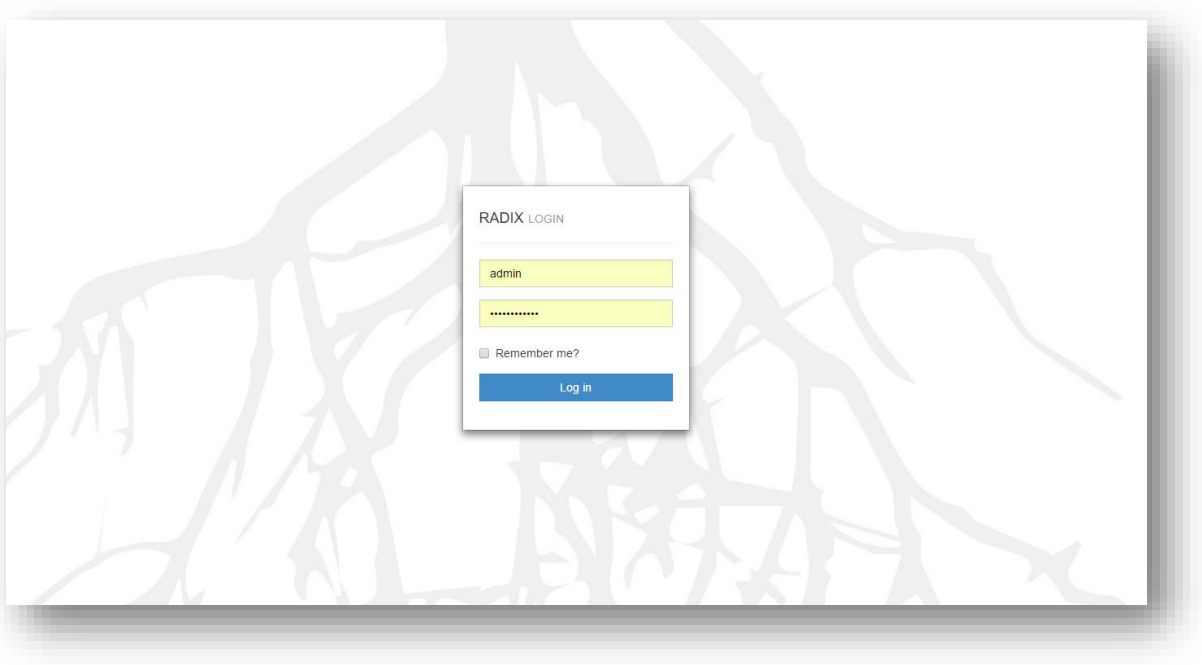
Utöver vyerna med informationstabeller finns också vyer för inloggning, överblick och användare.

3.2.1 Vyer utan informationstabell

Innan användaren kommer åt vyerna med informationstabeller möts denne av vyer för inloggning och överblick av systemet. Nedan finns en beskrivning av vad dessa vyer innehåller och hur de ser ut. Det finns också en beskrivning till vyn för hantering av användare som har en striktare autentisering än de andra vyerna.

3.2.1.1 Inloggning

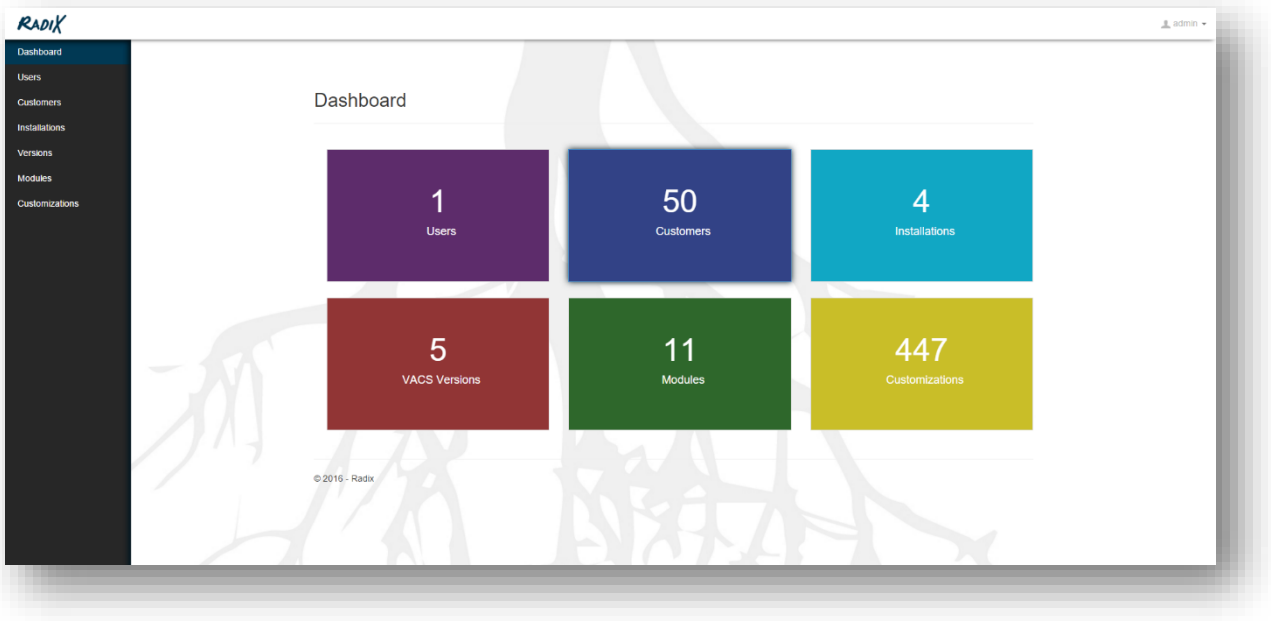
På inloggningssidan finns en ruta där användaren loggar in för att komma åt webbapplikationen. Här finns valideringskontroller för användarnamn och lösenord så att ingen obehörig kan logga in.



Figur 3.2 Loginsida

3.2.1.2 Dashboard

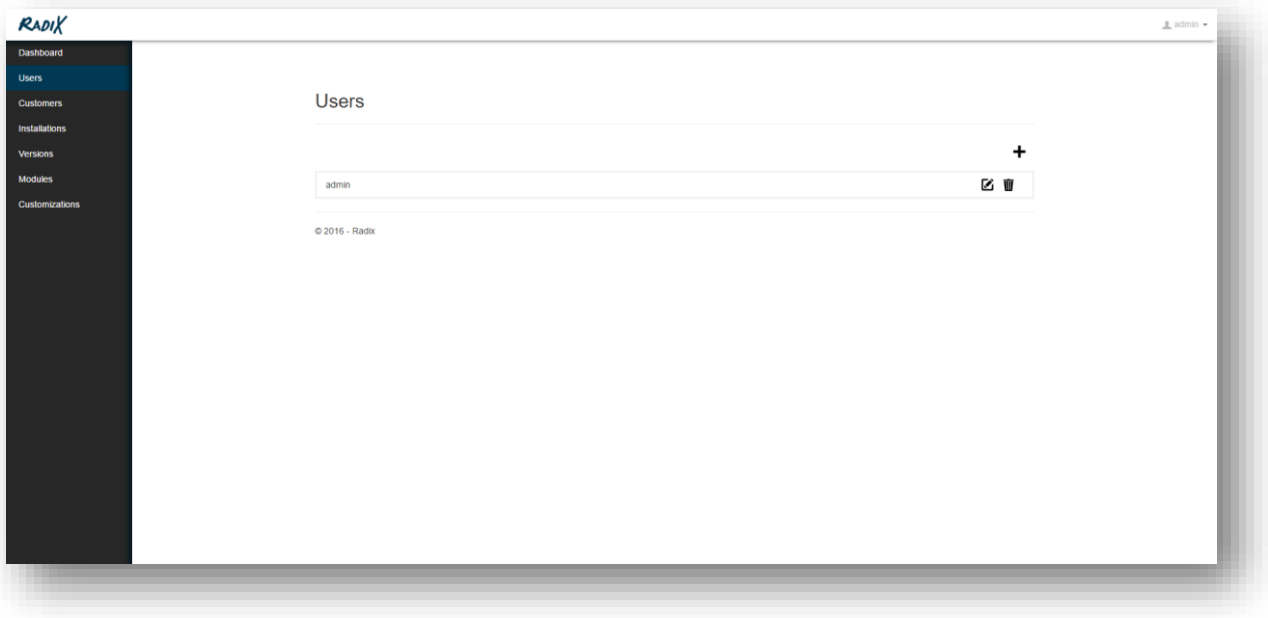
Dashboard är den första vyn som användaren möts av efter denne loggat in. Här finns en överblick av webbapplikationen och länkar till olika tabellsidor i form av brickor. I varje bricka visas statistik för antalet kunder, installationer, anpassningar, moduler och VACS-versioner. Om man håller pekaren över en bricka blir den tydligt markerad genom att den lyser upp och får en skugga bakom sig.



Figur 3.3 Radix Dashboard

3.2.1.3 Users

Vyn Users innehåller en tabell på alla användare som finns i systemet. Här kan man även skapa nya användare samt redigera och radera befintliga. Vid skapande av användare väljer man användarnamn, lösenord och roll. Vid redigering kan man ändra namn och roll. Vyn Users kan endast nås om användaren har en roll med administratörsrättigheter.



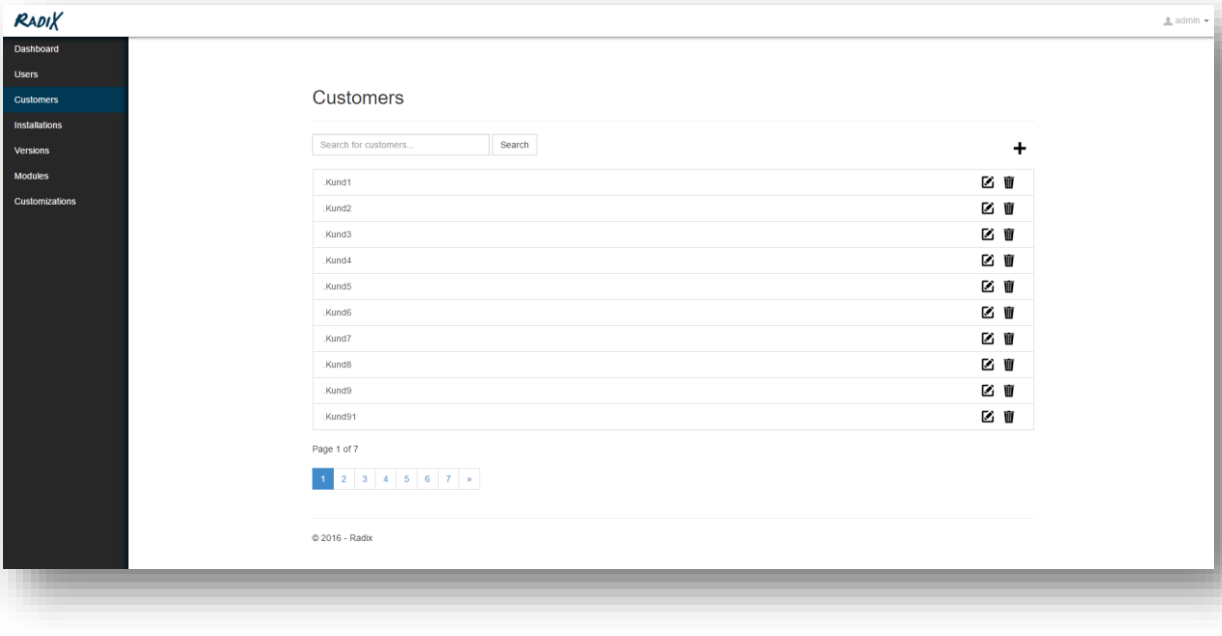
Figur 3.4 Radix Användare

3.2.2 Vyer med informationstabell

Alla vyer med informationstabeller är uppbyggda på samma vis och innehåller en presentation av de instanser som är relevanta till den specifika vyn man valt att visa. Utöver detta finns funktionalitet för skapa nya instanser samt att hantera och söka efter befintliga. Nedan följer en beskrivning av vad som återkommer på respektive vyer.

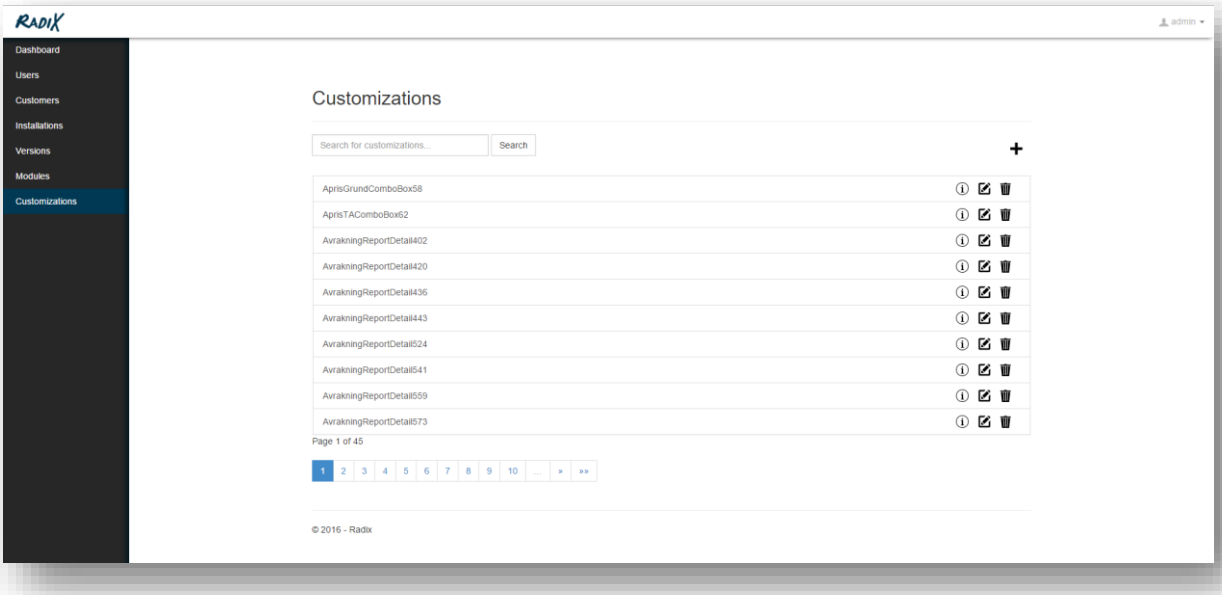
3.2.2.1 Presentation av data

Data för varje vy representeras i en tabell. Varje rad i tabellen representerar en instans som är relevant till vald vy. Tabellerna delas även upp i tabellsidor av tio instanser per sida. Nedan är ett exempel på vyn ”Customers” där alla kunder listas i en tabell.



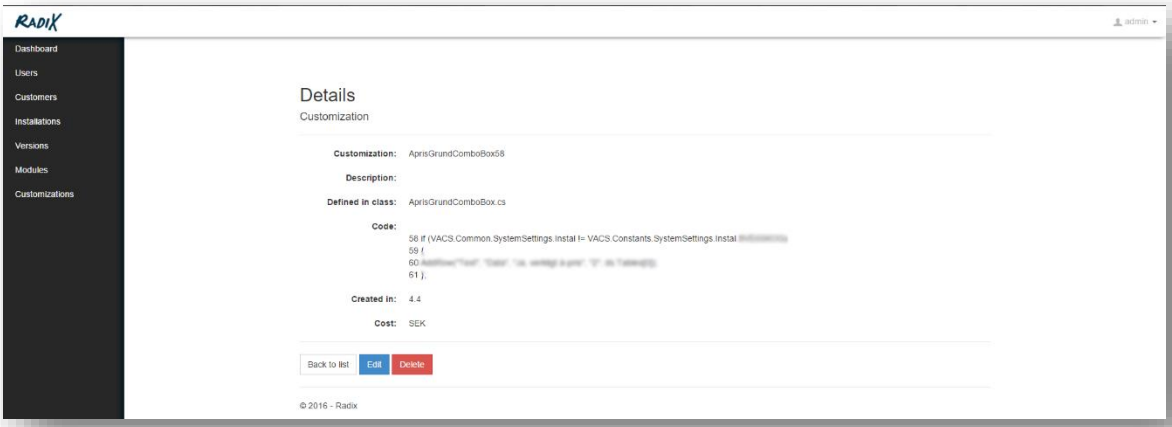
Figur 3.5 Tabell, kunder

I vissa fall finns även en knapp för att visa mer detaljerad information om instansen, se knappen med ett ”i” i Figur 3.6 nedan.



Figur 3.6 Anpassningar

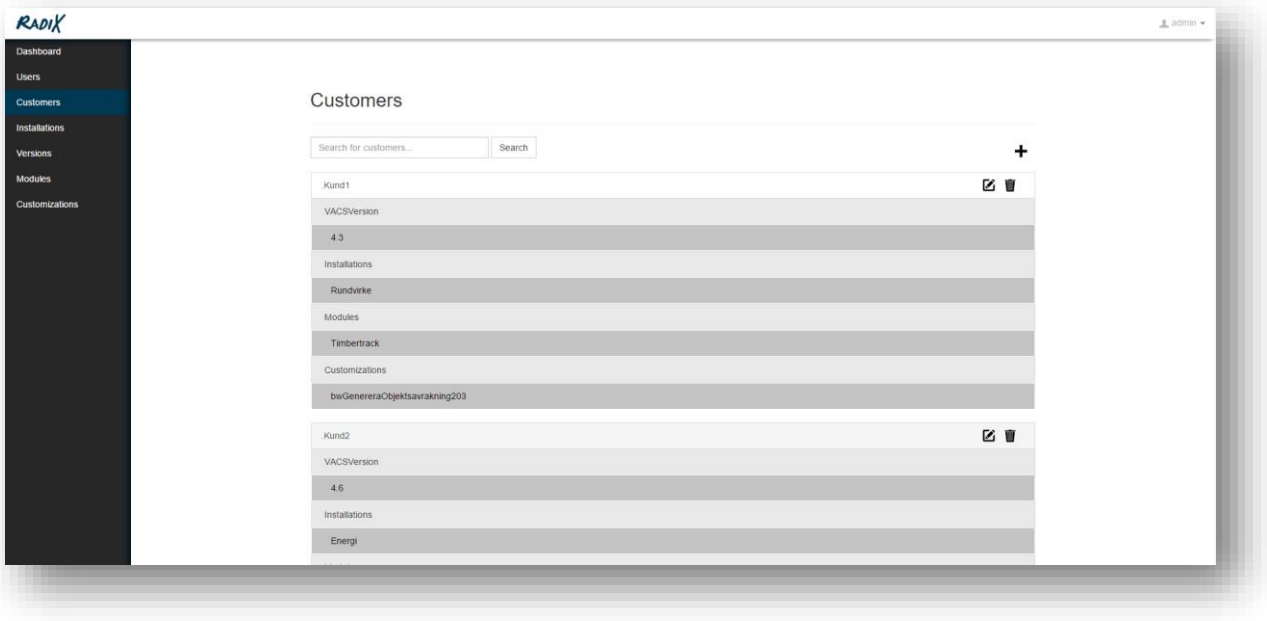
Om man klickar på detalj-knappen skickas man vidare till en vy med ytterligare information om instansen. Vilken information som visas beror på vilken vy man navigerat från. Nedan, i Figur 3.7, visas ett exempel på en vy med detaljer för en anpassning.



Figur 3.7 Detaljer, anpassning

3.2.2.2 Expanderad rad

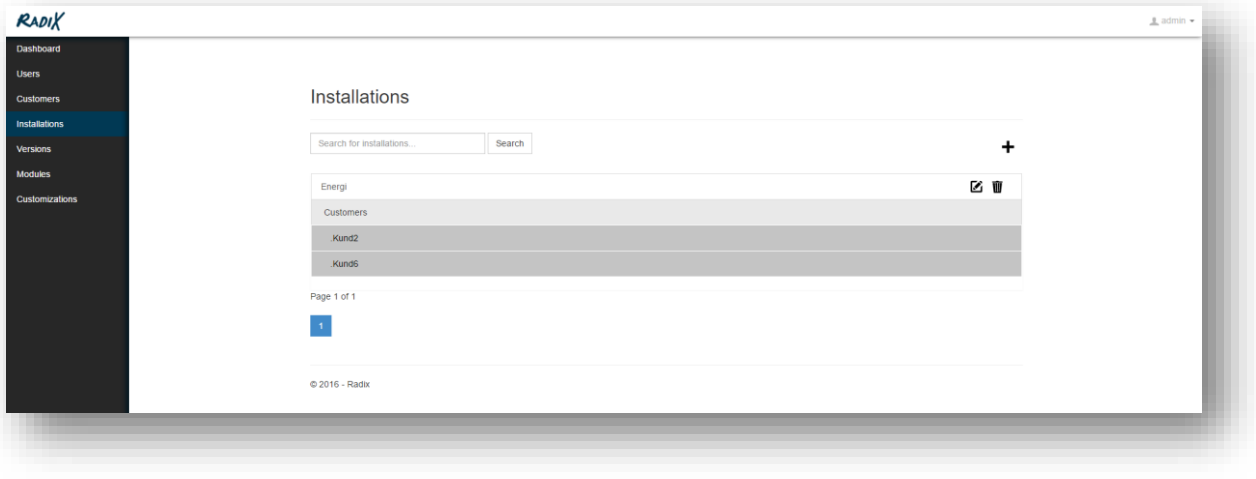
Om man klickar på en tabellrad expanderas raden och visar relationer som är relevanta till vald instans. Nedan, se Figur 3.8, visas ett exempel från Figur 3.5 där man vidare klickat på tabellraden för kunden ”Kund1” och raden har expanderats för att visa relationer till denne kund. I detta fall är relationerna vilken VACS-version, vilka installationer, vilka moduler och vilka anpassningar kunden använder sig av.



Figur 3.8 Expanderad tabellrad, kunder

3.2.2.3 Navigering

Klickar man på en relation i expanderad tabellrad skickas man vidare till relevant vy utefter den valda relationen. På så vis kan man navigera runt i Radix och få hämta information utifrån olika perspektiv. Om man exempelvis klickar på raden för installationen ”Energi” i vyn ”Customers”, se Figur 3.8, skickas man vidare till vyn ”Installations” i Figur 3.9 och ges relevant information utifrån vald installation. I detta fall listas vilka kunder som använder ”Energi” som installation.



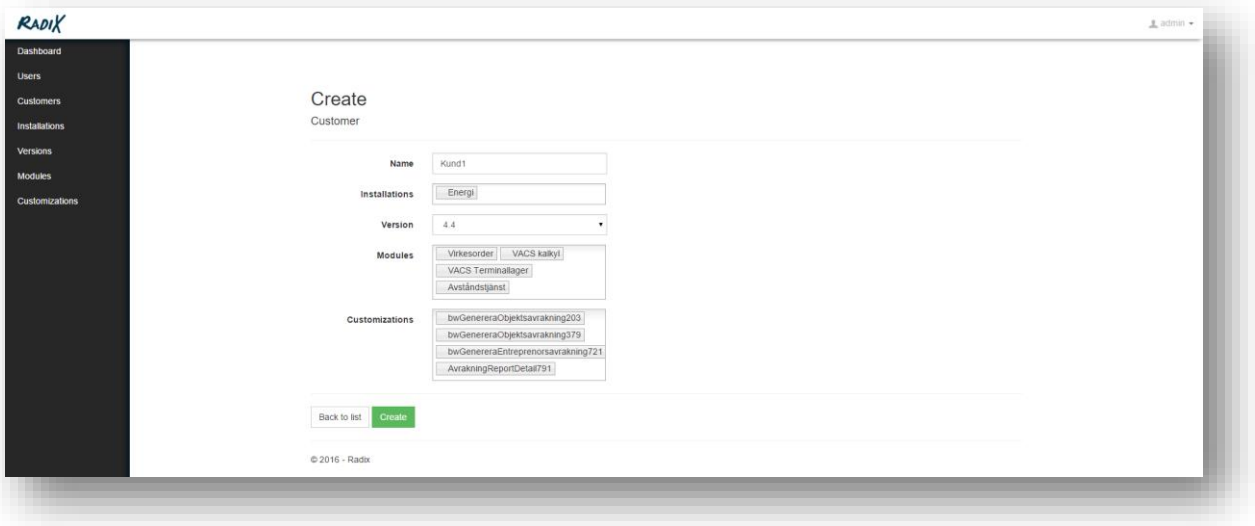
Figur 3.9 Navigering Installation

3.2.2.4 Hantering

Varje tabellrad innehåller knappar för att hantera respektive instans, se Figur 3.5. Dessa är knappar för att redigera och ta bort instansen. Över tabellen finns en knapp för att lägga till en instans. Hantering av instanser sker på samma vis men innehållet ser olika ut beroende på vilken vy man hanterar informationen på.

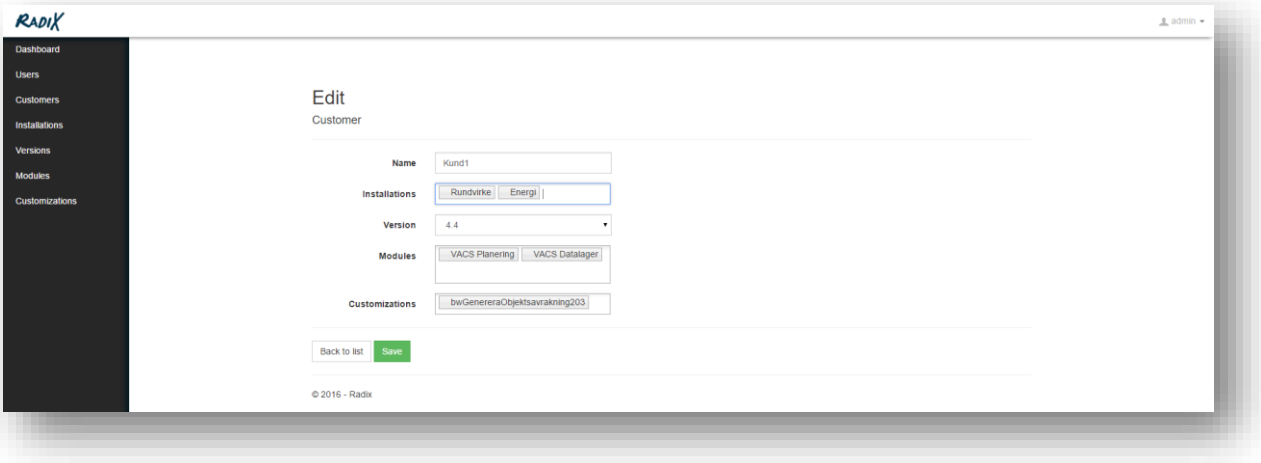
Vid skapande och redigering av en instans väljer man namn på instansen samt tillhörande information som är relevant till den kategorin av instanser. Vid borttagning av en instans skickas man till en vy för att bekräfta borttagningen, se Figur 3.12. Nedan följer exempel på hantering av en kund.

Vid skapande av kund väljer man namn på kund samt vilka relationer kunden har till systemet, se Figur 3.10.



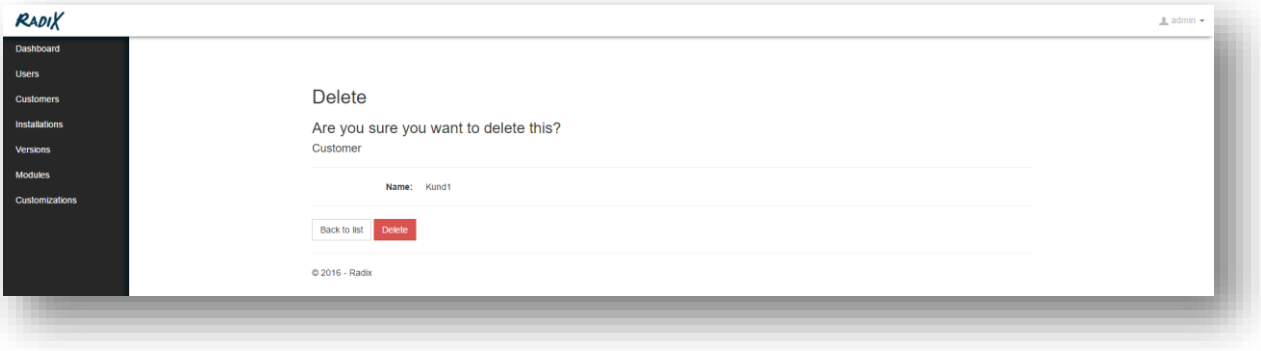
Figur 3.10 Radix Skapa kund

Vid redigering av kund får man upp samma alternativ som vid skapande av kund förutom att fälten redan är ifyllda med tidigare vald information, se Figur 3.11. Här kan man ta bort och lägga till relevant information för kunden.



Figur 3.11 Redigera, kund

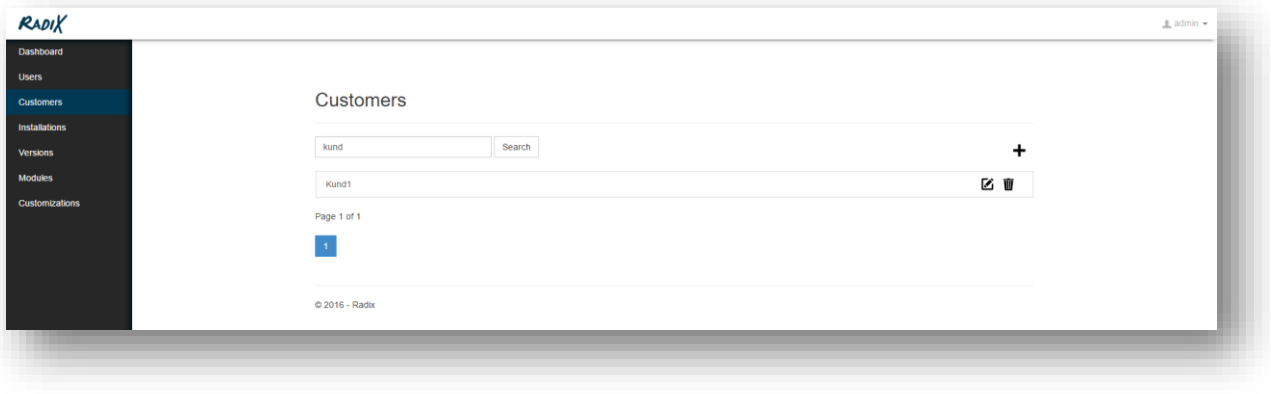
När man vill ta bort en kund får man en fråga om man verkligen vill ta bort kunden, se Figur 3.12. Trycker man då på knappen ”Delete” accepterar man att kunden tas bort från databasen. Därefter skickas man tillbaka till vyn med kundtabellen.



Figur 3.12 Ta bort, kund

3.2.2.5 Sök

Över tabellen finns en sökruta där man kan söka efter en specifik instans i tabellen. Om man i vyn ”Customers”, se Figur 3.5, skriver in namnet ”Kund 1” i sökfältet och klickar på sök, listas alla kunder med det namnet, se Figur 3.13.



Figur 3.13 Sök, kund

3.2.2.6 Innehåll

Var och en av vyerna med informationstabeller är uppbyggda på samma sätt men innehåller olika data beroende på vad för information man söker. Nedan följer en beskrivning till de olika vyerna, vad de innehåller för data och vad som skiljer dem åt.

3.2.2.6.1 Customers

Vyn Customers innehåller en tabell av kunder, se Figur 3.5. Här kan man söka i kundtabellen efter en specifik kund eller VACS-version samt lägga till, ta bort och redigera kunder. Om man klickar på en kund får man dessutom upp mer information. Denna information innefattar tillhörande VACS-version för den specifika kunden samt vilka anpassningar, installationer och moduler som kunden använder till sitt VACS-system.

Vid skapande och redigering av kund väljer man namn på kund, vilken VACS-version kunden har, samt vilka installationer, moduler och anpassningar kunden använder. Det sistnämnda representeras i form av separata flervalslistor, se Figur 3.11.

3.2.2.6.2 Installations

Vyn Installations innehåller en tabell med alla installationer som finns. Här kan man söka i tabellen efter en specifik installation samt lägga till, ta bort och redigera installationer. Om

man klickar på en installation får man upp vilka kunder som använder den installationen. Vid skapande och redigering skriver man in namnet på installationen.

3.2.2.6.3 Versions

Vyn Versions innehåller en tabell med alla VACS-versioner som finns. Här kan man söka i tabellen efter en specifik version samt lägga till, ta bort och redigera versioner. Om man klickar på en version får man upp vilka kunder som använder den versionen samt vilka anpassningar som finns tillgängliga i versionen. Det finns också en knapp för detaljer om versioner. Om man klickar på knappen för detaljer ser man ytterligare information om versionen som inte visas i tabellen. I detta fall visas datumet då VACS-versionen var skapad. Vid skapande och redigering skriver man in namnet på versionen samt datumet då versionen var skapad.

3.2.2.6.4 Modules

Vyn Modules innehåller en tabell med alla moduler som finns. Här kan man söka i tabellen efter en specifik modul samt lägga till, ta bort och redigera moduler. Om man klickar på en modul får man upp vilka kunder som använder den modulen. Vid skapande och redigering skriver man in namnet på modulen.

3.2.2.6.5 Customizations

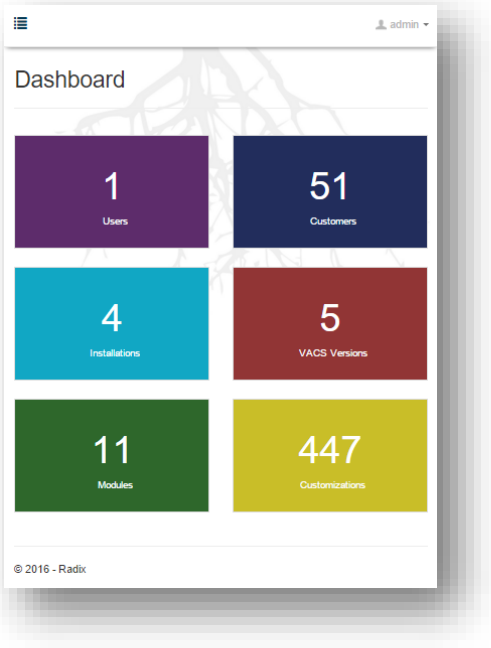
Vyn Customizations innehåller en tabell med alla anpassningar som finns. Här kan man söka i tabellen efter en specifik anpassning samt lägga till, ta bort och redigera anpassningar. Om man klickar på en anpassning får man upp vilka kunder som använder den anpassningen samt i vilka VACS-versioner den finns med i. Det finns också en knapp för detaljer om anpassningen. Vid skapande och redigering av en anpassning väljer man namn, beskrivning, kostnad, klassen den är kodad i samt vilka VACS-versioner den finns i.

Om man klickar på knappen för detaljer ser man ytterligare information om anpassningen som inte visas i tabellen, se Figur 3.7. Denna information är namn på anpassning, beskrivning av anpassning, koden som avser anpassningen, vilken VACS-version anpassningen var skapad i samt kostnaden för anpassningen.

3.3 Responsivitet

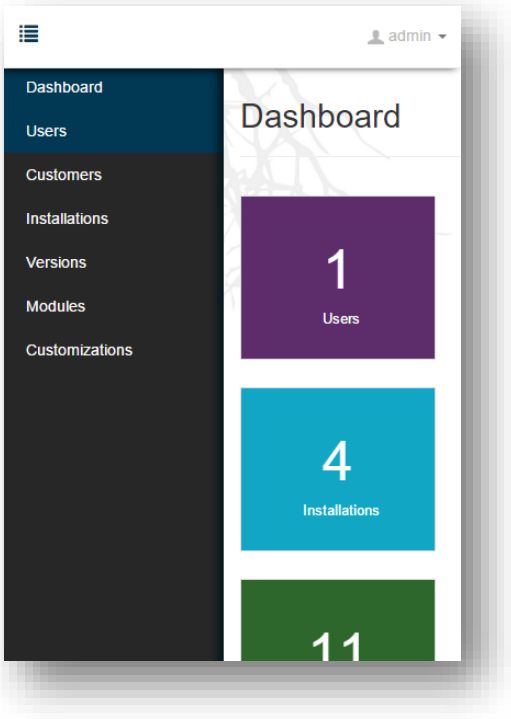
Radix är en responsiv webbapplikation, d.v.s. att den fungerar på alla typer av enheter, oberoende av skärmstorlek. Det var ett krav från uppdragsgivaren att slippa programinstallation på PC eller mobil enhet. Detta har åstadkommits med hjälp av Radix dynamiska gränssnitt som skalar ned allt innehåll när man minskar skärmstorleken. Nedan illustreras exempel på hur webbapplikationen kan se ut på en mobil enhet.

Figur 3.14 illustrerar hur vyn ”Dashboard”, från Figur 3.3, presenteras på en mobil enhet. Här har rutorna för överblick skalats ner för att likna en mobilapplikation. Även menyn har fällts in för att inte ta upp för mycket plats på skärmen.



Figur 3.14 Dashboard, mobil enhet

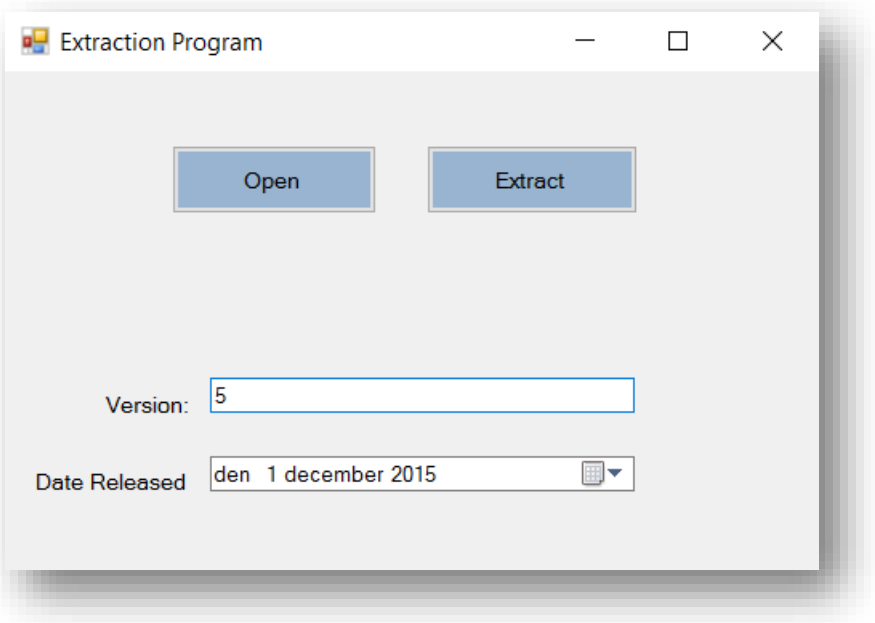
Nedan, i Figur 3.15, illustreras samma vy som ovan med menyn utfälld. Den ser ut på samma sätt som tidigare men allt material på sidan är pressat åt sidan. När man klickat på en länk och navigerat till vald sida återgår menyn till sitt infällda läge.



Figur 3.15 Meny, mobil enhet

3.4 VACS Extract

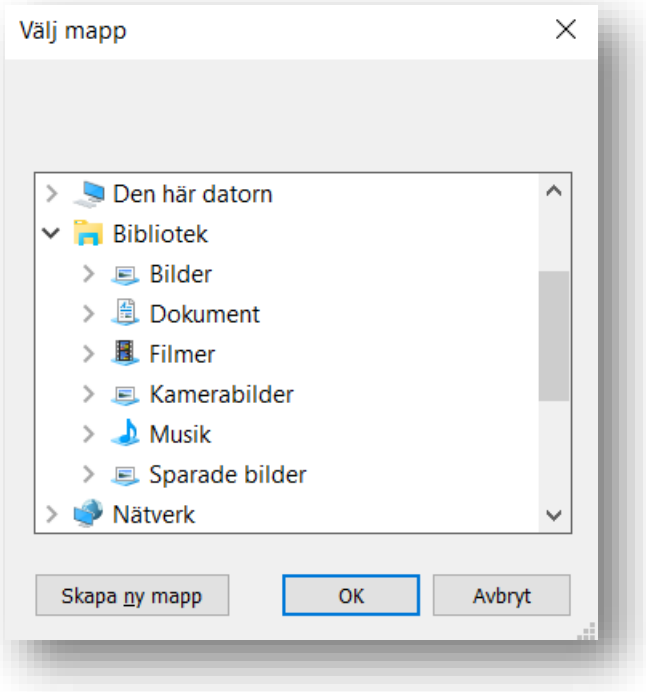
I Figur 3.16 nedan visas det externa extraheringsprogrammet till Radix som extraherar kopplingar mellan anpassningar och kunder ur källkoden för VACS. Här finns en knapp för att välja VACS-version, ”open”, samt en knapp för att extrahera, ”extract”. Utöver dessa knappar finns ett fält för att skriva in vilken version av VACS man vill extrahera från samt ett datumfält där man väljer vilket datum VACS-versionen släpptes.



Figur 3.16 Extraheringsprogrammet

3.4.1 Välj mapp

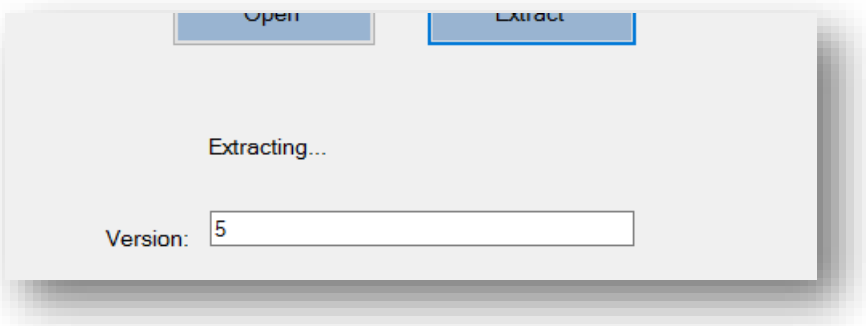
För att välja vilken VACS-version man vill extrahera från klickar man på knappen ”open”. Det öppnas då en ny ruta där man väljer i vilken mapp på datorn man vill söka i, se Figur 3.17 nedan.



Figur 3.17 Välj mapp

3.4.2 Extrahera

När man valt mapp, skrivit in VACS-version och klickat i vilket datum den skapades, klickar man på knappen Extract. En textrad visar i vilket tillstånd programmet befinner sig i, se ”Extracting…” i Figur 3.18 nedan.



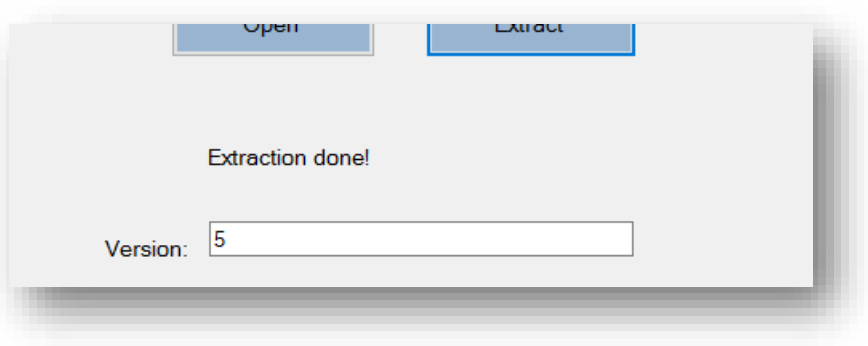
Figur 3.18 Extraherar

3.4.3 Felhantering

Om något går fel under extraheringen finns inbyggd felhantering som meddelar användaren vad felet är och vart i programmet felet har uppstått. Eftersom programmet endast används av personal på Evry skrivs felet ut i klartext för att underlätta felsökning i källkoden.

3.4.4 Resultat

När programmet är klar med extraheringen meddelas användaren av programmet, se Figur 3.19 nedan.



Figur 3.19 Extrahering klar

I webbapplikationen Radix kan man nu granska kopplingarna mellan kunder och anpassningar. Även kodrader för anpassningar extraheras ur källkoden för VACS och kan nås via webbapplikationen då man visar detaljer för en anpassning. Eftersom man skriver in VACS-version och släppdatum i extraheringsprogrammet följer även dessa värden med till webbapplikationen.

3.5 Sammanfattning

Detta kapitel beskriver designen av databasen och de olika vyerna för webbapplikationen Radix. Här ges en inblick i de tabeller som databasen innehåller samt relationerna mellan dem. Det ges också en ingående förklaring till de olika vyerna, hur de ser ut, hur de hänger ihop och hur de används. I vyerna ser man också tydligt kopplingen till databasen. Utöver webbapplikationen ges också en inblick i det externa extraheringsprogrammet samt en beskrivning till hur det används.

4 Implementation

Detta kapitel beskriver implementationen av Radix. Kapitlet börjar med en beskrivning av de metoder som hanterar de olika entiteterna i Radix. Därefter följer en beskrivning av vy klasserna som presenterar modellen. Slutligen beskrivs även de centrala delarna av extraheringsprogrammet.

4.1 Uppbyggnad

För varje entitet finns det en modell-klass som definierar entiteten. För varje modell-klass finns det en controller som hanterar den specifika modellen. I en controller finns följande metoder.

- Metoder för grundläggande hantering av entitetsinstanser. I dessa metoder finns det även kod som hanterar användarvänligheten i Radix.
- Metoder för att fylla i alla flervalslistorna som används i Radix.
- Uppdateringsmetoder som används vid redigering av en befintlig instans vilket är en utökning av den grundläggande redigeringsmetoden.

4.1.1 Hanteringsmetoder

Varje controller är uppbyggd med fyra centrala CRUD metoder [40] som utgör kärnan i varje controller. Nedan följer en mer detaljerad beskrivning av dessa metoder.

4.1.1.1 Index

Index metodens främsta uppgift är att hämta en lista av instanser och skicka dessa vidare till vyn. Metoden ser likadan ut i alla controller-klasser, den enda avvikelsen är att den hämtar olika instanser i databasen beroende på vilken controller man använder, t.ex. om man är i controllern som hanterar kunder, kommer index metoden då att hämta alla instanser av kunderna i databasen.

Inuti denna metod finns kod som hanterar sökfunktionen, expanderad rad samt sidnumrering. Eftersom dessa är i direkt koppling till vad som ska presenteras i vyn har ingen egen metod skapats för dessa funktioner. Detta är beskrivet mer i detalj i 4.2.2 Presentation.

4.1.1.2 Create

Denna metod skapar en ny entitet med hjälp av en model binder [41] för den entiteten och sparar den i databasen. En model binder är funktionalitet i ASP.NET som gör det enklare att jobba med data inskickad av ett formulär. Model binder omvandlar formulärvärden som skickats från vyn till CLR-typer och skickar dessa till metodens parametrar och på så sätt instansierar en ny entitet [41].

```
public ActionResult Create([Bind(Include = "Name,Description,Cost")] Customization
customization, CustomizationIndexData model)
```

Det som skiljer create-metoderna i de olika controllers är vad för relationer de ska lagra för varje entitet. När man t.ex. skapar en ny kund skapas tre olika listor för varje relationstyp. Dessa listor fylls sedan på med värden som valts i create-vyn med hjälp av foreach-loopar.

```
customer.Installations = new List<Installation>();
customer.Modules = new List<Module>();
customer.Customizations = new List<Customization>();

if (model.ModuleIds != null)
{
    foreach (var m in model.ModuleIds)
    {
        var moduleToAdd = db.Modules.Find(m);
        customer.Modules.Add(moduleToAdd);
    }
}
```

4.1.1.3 Edit

Edit metoden redigerar en befintlig entitetsinstans med hjälp av två inparametrar. Inparametrarna som metoden använder sig av är id och model. Id är ett nummer som används för att hitta den specifika entitetsinstansen i databasen och hämta all information om den för att sedan spara det i en ny entitetsinstans.

```
Customer customerToUpdate = db.Customers
                .Include(i => i.Installations)
                .Include(i => i.Modules)
                .Include(i => i.VACSVersion)
                .Include(i => i.Customizations)
                .Where(i => i.ID == id)
                .Single();
```

Model är av typen CustomerViewModel och innehåller de värden man har redigerat i vyn och vill spara för en specifik entitetsinstans, i detta fall en kund. Denna model skickas sedan ihop med customerToUpdate till en uppdateringsmetod som kopierar över värdena från model till customerToUpdate. Uppdateringsmetoden beskrivs i detalj i delkapitel 4.1.4.

4.1.1.4 Delete

Delete-metoden har en inparameter id. Id:t som skickas med tillhör entitetsinstansen som man vill ta bort. Id används för att hämta entitetsinstansen från databasen. Entitetsinstansen används sedan för att utföra en kontroll på om det finns några kunder kopplade till den. Om så är fallet finns det kod som kommer att generera ett felmeddelande på sidan då detta inte är tillåtet. Denna kod finns i alla controllers som innehåller en delete-metod förutom kund controllern. Om det inte finns några kunder kopplade till entitetsinstansen fortsätter metoden med att ta bort entitetsinstansen från databasen.

```
Installation installation = db.Installations.Find(id);

if (installation.Customers.Count() != 0)
{
            TempData["DeleteError"] = "Not allowed to delete. This installation has
            one or more customers";
            return RedirectToAction("Index");
}
```

4.1.2 Metoder som hanterar användarvänlighet

Metoderna som hanterar instanserna innehåller även kod för användarvänligheten på sidan eftersom dessa är i direkt relation till varandra.

4.1.2.1 Expanderad rad

För att en rad i en informationstabell skall kunna fällas ut när man klickar på den, se delkapitel 3.2.2.2, används ett javascript plugin ur ramverket Bootstrap. Detta plugin används vanligtvis till att fälla ut ett textfält när man klickar på en rad. I projektet Radix används pluginet istället till att fälla ut nya rader med information när man klickar på en rad eller när man navigerat till raden från en annan vy. Nedan visas kod från index-vyn för ”Customers”. Här illustreras hur informationstabellen är uppbyggd med hjälp av pluginet för att kunna fälla ut rader när man klickar på en kund-instans. När man klickar på en instans, se kod nedan med id:t ”header”, kallar länken på ”collapse” i javascript pluginet vilket gör att information om kunden fälls ut. Alla rader i tabellen, både de för kunderna och de för informationen om kunderna, är uppbyggda av ”list-group”-klasser som gör att pluginet tolkar dem som rader

som skall kunna fällas ut. I exemplet nedan listas alla installationer som är kopplade till vald kund.

```
<div class="panel list-group" ... >

   @foreach (var item in Model)
   {
      <a id="header" ... data-toggle="collapse" ... >
         @item.Name
         <img ... />
         <img ... />
         <img ... />
      </a>
      <div id="@item.CustomerID" class="panel-collapse collapse" ... >

         ...
         <div class="list-group">
            @if (item.Installations != null)
            {
               foreach (var installation in item.Installations)
               {
                  <a class="list-group-item small" ... > @installation.Name </a>
               }
            }
         </div>

         ...
      </div>

   }

</div>
```

I varje index-metod finns också en koll om man navigerat från en annan vy och endast vill få upp information om vald instans. Om så är fallet har det skickats med en navigationssträng från föregående vy. Utifrån denna navigationssträng hämtas data från databasen som endast är relevant till den valda instansen. Koden nedan har hämtats ur index-metoden för vyn ”Customers”. Denna kod hämtar data ur databasen där namnet är lika med navigationssträngen.

```
else if (!String.IsNullOrEmpty(navigationString))
{
   viewModel.Customers = db.Customers
   .Where(i => i.Name.Equals(navigationString))
   .Include(i => i.VACSVersion)
   .Include(i => i.Installations)
   .OrderBy(i => i.Name);
}
```

För att programmet skall fälla ut raden automatiskt krävs att ett id skickats med till index-metoden i controllern för aktuell vy. Detta görs endast om man navigerat från en annan vy genom att klicka på en instans i en informationstabell. Koden nedan är hämtad ur vyn "Customizations" och illustrerar en kund-instans i tabellen för anpassningar. Om användaren klickar på denna länk skickas id:t och namnet för kunden med till vyn "Customers".

```
<a class="list-group-item small" href=@Url.Action("Index", "Customer", new {
navigationString = customer.Name , id = customer.CustomerID})>@customer.Name </a>
```

Om en användare klickat på länken som beskrivits ovan har ett namn och ett id för vald kund skickats med från tidigare vy och kundens id skickas därför med till vyn, se kod nedan.

```
ViewBag.CustomerID = id;
```

Raden för kunden fälls sedan ut och presenterar information om kunden med hjälp av ett javascript i index-vyn för "Customers". Javascript-koden nedan använder id:t som skickats med i form av en ViewBag för att fälla ut information om kunden.

```
<script>
    $( document ).ready(function() {
        $('#@ViewBag.CustomerID').collapse()
    });
</script>
```

Om användaren däremot inte har klickat sig vidare via länken finns inget namn eller id medskickat och id:t kommer därför anta värdet null, se kod nedan. Om id inte har något värde kommer raden inte fällas ut med hjälp av javascriptet.

```
if (searchString != null || String.IsNullOrEmpty(navigationString))
{
   id = null;
}
```

4.1.2.2 Navigering

I vyerna som genererar tabellerna finns det kod som anropar index metoden när man klickar på den raden man vill navigera till. I kodexemplet nedan visas hur tabellrader skapas för varje kund i databasen med hjälp av en loop. Ett href attribut används för att specificera vad som

ska hända när man klickar på en av tabellraderna. Det som ska hända i detta fall är att metoden Url.Action ska anropas. Url.Action är en inbyggd funktion i .NET Framework som genererar en URL [42] till en action-metod. En action-metod är en metod i controllern som returnerar en vy, så att denna kan visas på användarens dator. De första två parametrarna specificerar action-metodens namn och controller-namnet därefter specificeras parametrarna man vill skicka till action-metoden. I detta exempel är det index-metoden i customer-controllern som ska anropas med parametrarna navigationString = customer.Name och id = customer.CustomerID.

```
foreach (var customer in item.Customers)
{
 <a class="list-group-item small"
href=@Url.Action("Index", "Customer", new { navigationString = customer.Name,
 id = customer.CustomerID })> @customer.Name </a>
}
```

Index-metoden använder parametern navigationString för att leta fram kunden i databasen och parametern id för att expandera kunden information se Figur 3.9.

4.1.2.3 Sökfunktion

Sökboxen skapas i koden för vyn. När man har tryckt på search skickas det ett anrop till index-metoden med en söksträng.

```
@Html.TextBox("SearchString", null, new { @class = "form-control",
placeholder = "Search for installations..." })
<input type="submit" value="Search" class="btn btn-default" />
```

Söksträngen används sedan för att hitta entitetsinstansen eller entitetsinstanserna i databasen. Värdena som hittas skickas tillbaka till vyn.

I exemplet nedan visas hur viewModel.Installations som används för att visa tabellen installationer sätts till alla värden som hittas i databasen som innehåller söksträngen.

```
if (!String.IsNullOrEmpty(searchString))
{
     viewModel.Installations = db.Installations
     .Where(i => i.Name.Contains(searchString))
     .Include(i => i.Customers)
     .OrderBy(i => i.Name);
```

Vymodellen skickas sedan tillbaka till vyn och visar endast värderna som har tagits fram ur databasen se Figur 3.13.

4.1.2.4 Paging

Siduppdelningen för tabellerna görs i controller klasserna för respektive entitet med hjälp av ett NuGet-paket [43] som heter PagedList.Mvc. Paketet installerar en PagedList-samlingstyp och förlängningsmetoder för IQueryable [44] och IEnumerable [45] samlingar. Förlängningsmetoderna använder IQueryable eller IEnumerable-samlingarna för att skapa en PagedList-samling som innehåller flertalet egenskaper och metoder för paging.

PagedList.Mvc installerar också en sidhjälpare som visar sidnumreringen för tabellsidorna, se Figur 4.1 nedan.

Figur 4.1 Sidhjälpare

I index metoden i controller klassen hanteras paging. Vid första anropet av metoden antar page-parametern värdet null. Klickar man på en sidnumrering antar page-parametern värdet för numret på tabellsidan.

```
public ActionResult Index(int? id, string searchString, string navigationString, int?
page)
```

Om söksträngen ändras måste page-parametern återställas till värdet 1.

```
if (!String.IsNullOrEmpty(searchString))
{
            page = 1;
}
```

För att bestämma antalet instanser per tabellsida sätts värdet pageSize till 10. De instanser som visas per tabellsida är då tio till antalet.

```
int pageSize = 10;
```

Om page-parametern har värdet null vill man sätta värdet på page-parametern till 1 för att motsvara första tabellsidan. Nedan visas koden som sätter värdet på page-parametern till 1 om det tidigare är null.

```
int pageNumber = (page ?? 1);
```

I slutet av metoden konverteras kundobjekten som är samlade i en IEnumerable-samling till en sida av kunder i en samlingstyp som stödjer tabellsidor dvs. till en PagedList-samling med hjälp av förlängningsmetoden ToPagedList, se kod nedan. Sidan av kunder skickas sedan till vyn för att presenteras på webbapplikationen. ToPagedList metoden får även in parametrar för storleken på sidorna (pageSize) och aktuellt sidnummer (pageNumber).

```
return View(viewModel.Customers.ToPagedList(pageNumber, pageSize));
```

4.1.2.5 Sidomeny

Sidomenyn som syns i Figur 3.3 är skapad med hjälp av ”simple sidebar” som är en css-mall till Bootstrap. Denna mall hjälper till att skapa en meny som stänger sig vid mindre skärmstorlekar. Nedan finns kod hämtad ur layout-filen för Radix. För att css-mallen skall kunna användas flyttas hela layouten av sidan till en div med id:t ”wrapper”. Länkarna som skall finnas i sidomenyn flyttas till en div med id:t ”sidebar-wrapper”. Resten av innehållet på sidan flyttas till en div med id:t ”page-content-wrapper”. För var och en av dessa id:n finns tillhörande css-kod i css-mallen som skapar en sidomeny och anpassar innehållet på sidan efter denna.

```
<div id="wrapper">

   <div id="sidebar-wrapper">
      <ul ... >
         <li> ... </li>
         <li> ... </li>
         <li> ... </li>
         <li> ... </li>
         <li> ... </li>
      </ul>
   </div>

   <div id="page-content-wrapper">
      ...
   </div>

</div>
```

4.1.3 Listmetoder

Webbapplikationen innehåller flertalet listor som hjälper till att välja relationer mellan de olika entiteterna. Vid exempelvis skapande av ny kund finns det fyra dropdown-listor (rullgardinslistor), tre av dessa är multi-select listor (flervalslistor) och en är en single-select lista (envalslistor). Multi-select listorna används där man ska kunna göra flera val samtidigt, t.ex. en kund kan ha flera anpassningar. Single-select listorna används där man endast ska kunna välja en entitetsinstans, t.ex. en kund kan endast ha en version av VACS.

Listorna fylls i med hjälp av separata metoder för varje lista som anropas av de metoder där dropdown-listor ska finnas t.ex. skapa en ny kund eller editera en befintlig kund metoderna.

Metoden för att fylla en lista av moduler får in en CustomerViewModel parameter som heter model. Denna parameter är en tom vymodel för en kund. Vymodellen innehåller en IEnumerable lista av SelectListItem [46].

```
public virtual IEnumerable<SelectListItem> Modules { get; set; }
```

SelectListItem är ett valbart objekt i en instans av SelectList [47], som är en lista där en användare kan välja ett eller flera objekt ur den listan.

Listan Modules fylls genom att hämta alla modul objekt från databasen och skapa en ny SelectListItem för varje sådant objekt. Varje SelectListItem-objekts värde tilldelas ett modul-id och text värdet tilldelas namnet på modulen. Text värdet blir det som visas i dropdown-listan för varje modul.

```
private void PopulateModuleListBox(CustomerViewModel model)
        {
            model.Modules = db.Modules.Select(m => new SelectListItem
            {
                Value = m.ModuleID.ToString(),
                Text = m.Name
            });
        }
```

I vyn skapas sedan listan med hjälp av HTMLHelpers [48] som är ett förenklat och renare sätt att skiva html kod. En listbox-metod som tar in tre parametrar som hjälp för att skapa html koden anropas.

Parametern Model.Modules är SelectListItem listan som fylldes med Modul objekt i controllern och används här som parameter till ListBoxFor metoden.

```
@Html.ListBoxFor(m => m.ModuleIds, Model.Modules, new { @class = "listbox" })
```

Efter att denna drodown-lista har renderats för användaren och denna har valt moduler till kunden sparas dessa moduler undan i ModuleIds för att sedan skickas tillbaka till create metoden i controllern. I Create metoden används dessa id:n för att skapa relationen mellan kund och modul.

```
public ActionResult Create([Bind(Include = "Name,VACSVersionID")] Customer customer,
CustomerViewModel model)
        {

            …

            if (model.ModuleIds != null)
            {
                foreach (var moduleId in model.ModuleIds)
                {
                    var moduleToAdd = db.Modules.Find(moduleId);
                    customer.Modules.Add(moduleToAdd);
                }
            }

            …

}
```

4.1.4 Uppdateringsmetoder

Uppdateringsmetoderna används när man redigerar en kundinstans eller en anpassningsinstans eftersom det är endast på dessa entitet man kan redigerar relationer mellan entiteten. Man kan t.ex. endast redigera relationen mellan kund och installation när man redigerar en kund. När man redigerar en installation kan man bara ändra namnet på den. Alla uppdateringsmetoder är uppbyggda på samma vis.

Ett exempel på en uppdateringsmetod är UpdateCustomerInstallations. I edit metoden i kund controllern anropas metoden UpdateCustomerInstallations. Denna metod tar emot två parametrar som heter model och customerToUpdate. Model parametern är av typen CustomerViewModel och CustomerToUpdate är en Customer typ. Parametern customerToUpdate är till för att veta vilken kund som ska uppdateras. Parametern model är vymodellen som har skickats från vyn som innehåller de nya värden som kunden ska uppdateras med.

```
private void UpdateCustomerInstallations(CustomerViewModel model, Customer
customerToUpdate)
```

Om vymodellen inte innehåller några InstallationsId:n sätts kundens installationer till en ny tom lista.

```
if (model.InstallationIds == null)
{
   customerToUpdate.Installations = new List<Installation>();
   return;
}
```

Den första if satsen kontollerar om en installation som loopats fram från databasen är vald i vyn och därmed finns i selectedInstallationsHS. Om den är vald så görs ytterligare en kontroll i form av en till if sats som kontrollerar om kundens lista av installationer customerInstallation inte innehåller den valda installationen. Om så är fallet läggs installationen in i kundens installationslista.

```
var selectedInstallationHS = new HashSet<int>(model.InstallationIds);
var customerInstallation = new HashSet<int>
(customerToUpdate.Installations.Select(m => m.InstallationID).ToList());

foreach (var installation in db.Installations)
{
   if (selectedInstallationHS.Contains(installation.InstallationID))
   {
      if (!customerInstallation.Contains(installation.InstallationID))
      {
         customerToUpdate.Installations.Add(installation);
      }
   }
```

Om installationen inte är vald och därmed inte finns i selectedInstallationsHS görs en kontroll om installationen finns i kundens installationslista. Om så är fallet kommer denna installation att tas bort.

```
   else
   {
      if (customerInstallation.Contains(installation.InstallationID))
      {
         customerToUpdate.Installations.Remove(installation);
      }
   }
}
```

4.2 Autentisering

Radix autentiseringshantering har skapats med hjälp av ASP.NET Identity [49]. ASP.NET Identity används i Visual Studios projektmallar för ASP.NET MVC, webbformulär [50], Web API [51] och SPA (Single Page Application) [52] och tillhandahåller funktioner som gör det möjligt för utvecklare att skapa validering av användare i form av unika användare med lösenord och tillhörande roller.

Följande funktioner av ASP.NET Identity har används i Radix:

- Skapande av nya användare och lösenord
- Autentisering av besökare på sidan
- Lösenordshantering
- Rollhantering

4.3 VACS Extract

Programmet börjar med att hämta sökvägen till VACS mappen. Efter att sökvägen till mappen har specificerats hämtas alla filsökvägar som finns i mappen och sparas undan i en sträng-array. Sökvägarna används senare i programmet av en StreamReader [53] som öppnar filen med hjälp av filsökvägen och läser igenom den.

4.3.1 Extrahering av kunder

Efter undersökning av filerna och genom att samtala med utvecklingsteamet kom det fram att kunder som använder VACS finns definierade i en fil som heter Companies.cs och att detta gäller för alla VACS versioner. Därav har det kunnat utvecklas en metod för att hitta denna fil genom att söka igenom alla filer i arrayen med filsökvägar som skapats i början av programmet. När filen har hittats av programmet används en StreamReader för att läsa igenom filen och extrahera ut namnen på kunderna.

Filen Companies.cs innehåller endast sträng-variabler som representerar kunder och ingen annan kod. Nedan visas ett exempel på hur koden kan se ut i filen Companies.cs.

```
public class Instal : Object
{
            public const string KundNamn1 = "KUNDNAMN1";
            public const string KundNamn2 = "KUNDNAMN2";
            public const string KundNamn3 = "KUNDNAMN3";
            public const string KundNamn4 = "KUNDNAMN4";
            public const string KundNamn5 = "KUNDNAMN5";

}
```

Koden i metoden kontrollerar varje rad i filen om den innehåller koden ”public cons string”. Om så är fallet hämtas kundnamnet ur koden med hjälp av två variabler, beginning och end. Beginning specificerar vart namnet börjar och end specificerar vart kundnamnet slutar. Beginning specificeras till efter ”string ” slutar och end specificeras till innan ” =” börjar. Variablerna används sedan för att anropa Substring [54] metoden. Substring metoden tar emot två parametrar, en som specificerar vart delsträngen börjar och en som specificerar hur lång den är och returnerar en delsträng. Delsträgen som genereras är kundnamnet. Kundnamnsträngen skickas sedan vidare till en metod som sparar kunden till Radix databas.

```
if (line.Contains("public const string"))
{
      int beginning = line.IndexOf("string ") + 7;
      int end = line.LastIndexOf(" =");
      customerName = line.Substring(beginning, end - beginning);
      addCustomerToDatabase(customerName);
}
```

Metoden som sparar kunden till databasen skapar ett kund objekt och använder sig av kundnamnsträngen som skickats in som en inparameter för att deklarerar namnet på den nyskapade kund objektet. Metoden skapar sedan en lista och fyller den med kunder som redan finns i databasen och därefter kontrollerar om kunden redan existerar i databasen. Om kunden existerar sparas den inte till databasen och på så sätt förhindrar att dubbletter skapas.

4.3.2 Extrahering av anpassningar

Efter att programmet extraherat alla kunder så fortsätter den med att extrahera anpassningar. Men hjälp av en for-each loop skickas varje filsökväg från sträng-arrayen med filsökvägar som skapats i början av programmet till en metod som ska läsa av filen och kontrollerar om filen innehåller en specifik kod-del som indikerar att filen innehåller en anpassning.

Metoden börjar med att skapa två listor. En lista av kundobjekt som fylls med alla kunder som finns i databasen. Den andra listan heter CustomizationCustomersList och sparar undan alla kunder som använder den aktuella anpassningen.

```
List<Customer> CustomerList = db.Customer.ToList();
List<Customer> CustomizationCustomersList = new List<Customer>();
```

Varje anpassning är specifik för en eller flera kunder som har köpt den och koden måste därför innehålla en specifik kod-del som indikerar att kunden har tillgång till anpassningen. Det finns inget specificerat i VACS källkod att en specifik metod innehåller en anpassning utan programmet måste leta efter den specifika kod-del som indikerar en anpassning. Det som indikerar en anpassning är en if-sats som kontrollerar om det är en specifik kund som kör VACS.

För att få en bättre förståelse för hur en anpassning kan se ut och vad programmet ska leta efter så följer ett exempel på en metod nedan. Metoden Sorteringsordning har till uppgift att en kund ska kunna ändra sorteringsordning. Kundnamn1 har dock valt att på sitt VACS system att inte kunna ändra sorteringsordning. Koden inom fetstil indikerar en anpassning och den kursiva koden indikerar en specifik kund som anpassningen gäller för.

```
void Sorteringsordning(object sender, EventArgs e)
{
              //KundNamn1 ska inte kunna ändra sorteringsordning.
              if(VACS.Common.SystemSettings.Instal.Equals(VACS.Constants.SystemSettings.Inst
              al.KundNamn1))
              {
                   Sortering.Enabled = false;
                   return;
              }

              if (Specificerad.Checked)
                  Sortering.Enabled = true;
                  else
                  Sortering.Enabled = false;
}
```

För att extrahera en anpassning måste StreamReader i metoden hitta en rad som innehåller koden ”VACS.Constants.SystemSettings.Instal”. Koden fortsätter sedan med en for-each loop som går igenom kundlistan som skapats i början av metoden och för varje kund kontrollera om kundens namn förekommer i raden. Om kundens namn finns med så läggs kunden in i CustomizationCustomer listan.

När en anpassning hittats i källkoden skall också koden för anpassningen sparas undan. Detta görs genom att räkna antalet klammerparenteser efter raden för anpassningen som hittats. När antalet höger – och vänsterparenteser tar ut varandra i antalet är kodsnutten för anpassningen slut. All kod mellan första och sista klammerparentes sparas undan i databasen med koppling till anpassningen som hittades.

Slutligen skickas namnet på filen som kommer vara anpassningsnamnet, koden samt CustomizationCustomer listan till en metod som sparar anpassningen till databasen.

4.3.3 Spara till databas

När en anpassning hittats behöver den sparas undan till databasen innan programmet letar vidare efter nästa anpassning. Programmet kallar därför på en metod som tar emot ett antal inparametrar och sparar dessa till databasen. Dessa är namn på anpassningen, koden för anpassningen samt VACS-version och namnet på klassen anpassningen hittats i. En koll görs först om anpassningen redan finns med i databasen. Om anpassningen inte finns med skapas ett nytt objekt av in-parametrarna som sedan sparas till databasen, se kod nedan.

```
if (!customizationList.Contains(customizationName.ToUpper()))
{
   Customization customization = new Customization
   {
      Name = customizationName,
      Code = code,
      VACSClass = className,
      VACSVersion = version
    };
    foreach(Customer customer in customerList)
    {
       customization.Customer.Add(customer);
    }

    db.Customization.Add(customization);
    db.SaveChanges();
}
```

Om anpassningen däremot redan finns med i databasen uppdateras den befintliga anpassningen i databasen med värdena från inparametrarna, se kod nedan.

```
else
{
   Customization customizationToUpdate = db.Customization.First(i => i.Name.ToUpper()
   == customizationName.ToUpper());
   customizationToUpdate.VACSVersion.Add(db.VACSVersion.Where(y => y.Name.ToUpper() ==
   _VACSVersionName.ToUpper()).SingleOrDefault());
   customizationToUpdate.Code = code;
   foreach (Customer customer in customerList)
   {
      customizationToUpdate.Customer.Add(customer);
   }
   db.SaveChanges();
}
```

4.4 Sammanfattning

Detta kapitel tar upp och förklarar implementationen av Radix utifrån de delar som tagits upp i design-kapitlet. Här ges exempel på ett informationsflöde i applikationen samt en detaljerad beskrivning av källkoden och hur de olika delarna hänger ihop. Det ges också en beskrivning till hur det externa extraheringsprogrammet till Radix fungerar och hur det är kopplat till databasen.

5 Resultat och utvärdering av verktyg

Detta kapitel börjar med en kravuppfyllelse. Här beskrivs de krav som uppfyllts och inte uppfyllts. Därefter följer en utvärdering av testning på systemet följt av resultatet av projektet Radix. I resultatet beskrivs hur Radix underlättar processen för Evry att sammanställa information om sina kunder till systemet VACS. Slutligen görs en utvärdering på de tekniker och verktyg som använts under projektets gång.

5.1 Kravuppfyllelse

I början av projektet gavs en tydlig kravspecifikation från uppdragsgivaren (se bilaga A). Därefter satte uppdragsgivaren upp en product backlog utifrån kravspecifikationen med prioritering på i vilken ordning de olika delarna skulle implementeras. Kraven diskuterades tillsammans med uppdragsgivaren och en plan sattes upp efter att tidsestimeringen gjorts.

Nedan listas de krav som uppfyllts och inte uppfyllts tillsammans med en beskrivning till hur de uppfyllts eller varför de inte uppfyllts.

Primära roller och övergripande funktionella krav

Säljchef

Säljchefen behöver ett systemstöd för att snabbt kunna bilda sig en uppfattning om:

▪ Vilka kunder kör VACS?	✓
▪ Vilka anpassningar har en kund köpt till VACS?	✓
▪ Vilka moduler har en kund köpt till VACS?	✓
▪ Vilka kunder har en specifik modul?	✓
▪ Vilken version av VACS körs hos vilka kunder?	✓
▪ (Pris på modul och pris på anpassning önskvärt)	✓

Säljchefens krav har uppfyllts genom att skapa tabeller med kunder, VACS-versioner, anpassningar och moduler som är kopplade till varandra utefter säljchefens frågor. Kravet av priset på modul har tagits bort under projektets gång men pris på anpassning har implementerats.

Produktchef

Produktchefen behöver ett systemstöd för att snabbt kunna bilda sig en uppfattning om:

▪ Vilka versioner av VACS finns?	✓
▪ Vilka kunder har en specifik version av systemet?	✓
▪ Vilka moduler finns till VACS?	✓
▪ Vilka beroenden finns mellan olika moduler?	✗
▪ Vilka moduler är kompatibla med respektive VACS-version?	✗

Produktchefens krav har uppfyllts med hjälp av tabeller som är kopplade till varandra. De sista två kraven har tagits bort av produktägaren under projektets gång för att förenkla datamodellen.

Övergripande tekniska krav

Skallkrav

▪ Systemet ska inte kräva en programinstallation på PC eller mobil enhet utan konsumeras förslagsvis som en webapplikation.	✓
▪ Systemet ska endast kunna användas av en användare som finns upplagd i applikationen.	✓
▪ Data ska lagras i en relationsdatabas.	✓
▪ Kommunikation mellan grafiskt gränssnitt och databas ska ske genom ett affärslogiskt lager med tjänster.	✓
▪ Affärslogiktjänster ska exponeras på ett vis som möjliggör att en mobil applikation kan konsumera och tillföra data i databasen i ett senare skede (utanför projektets scope).	✓

Skallkraven har uppfyllts genom att det skapades en användarbaserad webbapplikation där all data lagras i en relationsdatabas. Webbapplikationen har utvecklats med hjälp av MVC Framework. Det sista skallkravet togs bort av produktägaren då kravet lösts indirekt genom att webbapplikationen utformats till att vara responsiv på alla skärmstorlekar.

Börkrav

▪ Systemet bör designas med hjälp av moderna molntjänster för att maximera utvecklingsinsatsen.	✓
▪ Grafiskt gränssnitt bör stödja responsivitet och kunna köras på ett användarvänligt sätt även från en mobil enhet (tablet)	✓
▪ Systemet bör designas med hjälp av Microsoftteknologi i så många led det går då det stöder EVRY Forest & Logistics utvalda strategiska plattform	✓

Börkraven har uppfyllts eftersom molntjänster som Azure och Visual Studio Online har använts samt att systemet endast har utvecklats med hjälp av Microsoft plattform. Systemet har utformats som en responsiv webbapplikation vilket gör att den kan användas både på mobila enheter och på vanlig dator.

Övriga funktionella krav

▪ En högst önskvärd funktion i det tänkta systemet är att med automatik gå igenom källkoden för VACS och identifiera dimensionerna *kund* och *anpassning*. Dessa dimensioner bör då kunna laddas i Radix Databas utan att en användare manuellt lägger in dessa.	✓

En protoyp har tagits fram för att lösa ovanstående krav. Prototypen söker igenom källkoden för VACS och identifierar dimensionerna kund och anpassning och extraherar dessa dimensioner till Radix databas. Detta är endast en prototyp då systemet inte söker igenom hela källkoden.

Ett grafiskt gränssnitt där en användare (autentiserad) kan underhålla data ska möjliggöra följande funktioner:

▪ Skapa/ändra och ta bort kund	✓
▪ Skapa/ändra och ta bort VACS-version	✓
▪ Skapa/ändra och ta bort modul	✓
▪ Skapa/ändra och ta bort anpassning	✓
▪ Skapa/ändra och ta bort relation mellan moduler/anpassningar/VACS	✓

Kraven ovan har genomförts genom att webbapplikationen tillåter hantering av entiteterna kund, VACS-version, modul och anpassning.

Uppskattningsvis har projektet slutförts till ca 95%.

5.2 Resultat

Projektet Radix har resulterat i en webbapplikation som hanterar VACS kundbas och underlättar arbetet för produkt- och säljchef både vid fakturering och i utvecklingscykeln. Arbetet som tidigare har tagit veckor genom att manuellt sammanställa information i Excel och leta i VACS källkod kan nu göras automatiskt i ett enkelt och användarvänligt gränssnitt. Detta har vi löst genom att skapa en webbapplikation där man kan snabbt bilda sig en uppfattning om kunder och VACS alla systemdelar.

I vyn dashboard, se delkapitel 3.2.1.2, får man direkt se antalet kunder och omfattningen av olika systemdelarna i VACS. Här får man tydliga valmöjligheter för vidare navigering på sidan med interaktiva klickbara tiles.

I de vyerna med informationstabeller har användarvänligheten prioriterats. Här finns alla instanser samlade i tydliga och stilrena tabeller. När man klickar på en specifik instans i tabellen expanderas den och man kan snabbt hitta den sökta informationen.

Är man speciellt intresserad av en viss del av denna information kan man klicka på informationen och snabbt navigera vidare till den specifika informationens sida. Eftersom antalet instanser i några av tabellerna kan vara flera hundra är dessa uppdelade i sidor med en restriktion på max tio instanser per sida. För att en användare snabbt ska kunna leta fram en specifik instans har även en sökfunktion implementerats.

Anpassningarna har detaljerad information med kod och klassnamn där koden finns i VACS programmet. Nu behöver inte man inte längre leta manuellt i VACS källkod för att hitta anpassningskällkod och försöka lista ut vilka kunder som använder den.

Nedan följer ett citat från produktägaren för att sammanfatta resultatet av projektet.

”*Genom Radix har EVRY Forest & Logistics nu fått en ordentlig och konsekvent plattform i vilket vi kan stämma av vår produktkonfiguration hos varje kund vid varje enskilt tillfälle. Informationen har flyttats från lokala Excel listor ut till en tjänst som är fullt möjlig att använda från mobila verktyg och oavsett var vår personal befinner sig. Säljare kan enkelt gå in och skaffa sig en uppfattning om vilka moduler/anpassningar och versioner som finns ute hos en kund. Förvaltnings- och faktureringsprocessen har fått ett systemstöd som hjälper oss i våra årliga avstämningar. Produktchef och produktägare har fått ett inventeringssystem av vilka ”hot spots” vi har i systemet avseende anpassningar, något som kommer att vara ovärderligt för att kunna skriva om vårt system för framtiden med god kvalitet och transparens mot våra kunder. Vi kommer att spara mycket tid på att slippa göra manuella inventeringar av moduler och anpassningar, men framför allt har vi fått en enhetlig plats för denna information och verktyg för att kunna upprätthålla kvaliteten i denna information oavsett om vi befinner oss på kontoret eller ej. Kvaliteten i informationen kommer också att vara bättre i Radix än i tidigare Excel listor eftersom vi nu har automatiserat och konsekvent stöd för att göra inventering av anpassningar på ett repetitivt vis utan manuell sammanställning. Vi kan också genom att erbjuda access till systemet minska personberoende oss emellan, där nyckelpersoner är ensamma om att sitta på viss information.*

Vi är jättenöjda med utfallet av projekt Radix, men den ***kvantifierade*** *nyttoeffekten i termer av besparad tid eller förbättrad datakvalitet kvarstår för oss att göra.* ”

5.3 Testning

Något som uppdragsgivaren såg positivt på var testning på systemet. Detta har inte använts i Radix vilket är något som man i efterhand sett som en stor miss i utvecklingsprocessen. Testning har hela tiden funnits med som en uppgift att implementera men har hela tiden prioriterats bort för att istället fokusera på att få fram en färdig prototyp.

När grundversionen av systemet var färdig började arbetet direkt på nästa uppgift, extraheringsprogrammet. Här var tanken att testning också skulle användas men samma sak hände här. Testning prioriterades bort framför att leverera en fungerande prototyp.

Testning användes inte parallellt med implementationen på grund av bristande kunskaper om detta, och skulle därför tagit tid från utvecklingen att lära sig. Testning är viktigt för vidareutveckling av system och skulle definitivt prioriteras i nästa projekt.

5.4 Tekniker och verktyg

Redan innan projektet startade rekommenderade uppdragsgivaren verktyg och tekniker som också används i EVRY Forest & Logistics strategiska plattform. Alla de tekniker och verktyg som rekommenderats har använts under utvecklingen av Radix. Microsoft utvecklingsmiljö har varit lätt att använda och har underlättat att snabbt få upp en plattform att bygga vidare på. Versionshanteringen med Git har varit felfri under hela projektet. Databasen som användes låg i Azure-molnet vilket medförde att överföringstiderna mellan databasen och programmet förlängdes jämfört med att arbeta med en lokal databas.

5.5 Sammanfattning

Detta kapitel har påvisat de krav som uppfyllts och inte uppfyllts samt beskrivit resultatet av projektet Radix. Här gavs också en förklaring till hur användningen av olika tekniker och verktyg fungerat och vad de bidragit med i projektets utvecklingscykel. Utöver detta finns också en utvärdering av testning och varför det inte används.

6 Utvärdering av projektet

I detta kapitel görs en utvärdering av projektet. Här redogörs bland annat tidsåtgång för projektet, samt vad man lärt sig under projektets gång. Det ges också en utvärdering till vidareutveckling av systemet.

6.1 Tidsåtgång

I början av projektet lades all tid på implementering vilket var tre arbetsdagar per vecka. Detta var för att lära sig mer om uppgiften så att det fanns tillräckligt med information för att ha något att skriva om. Efter två veckor påbörjades rapportskrivandet. Här lades två arbetsdagar på implementering och en dag på rapportskrivande. Efter ytterligare en månad behövdes det ännu mer tid att lägga på rapportskrivande och implementering. Därför utökades arbetsdagarna till fyra dagar per vecka. Tiden som lades på implementering respektive rapportskrivande ändrades från vecka till vecka utefter vad som behövde läggas mest tid på. Denna tidsuppdelning användes under resten av projektet. Under första sprintplaneringen i början av projektet åtogs för lite uppgifter åt sprintens tid. Detta berodde på osäkerhet i vad man skulle klara av att genomföra på den planerade tiden. Efter varje ny sprint ökade kunskaperna i att uppskatta tidsåtgången för olika typer av uppgifter. Detta ledde till mer korrekta tidsestimeringar och bättre diskussioner om nya uppgifter på sprintplaneringarna.

6.2 Arbetsmiljö

All implementering i projektet har gjorts på Evry Forest kontor i Karlstad. De första två månaderna av projektet skrevs rapporten hemifrån med kommunikation via Skype. Därefter skrevs rapporten på Evrys kontor för att kunna sitta tillsammans samtidigt som man är mer fokuserad i ett arbetsklimat än när man sitter hemma samt att man snabbare får feedback från handledaren. En annan anledning till detta var att lära känna andra på företaget vilket medfört att man lättare kunnat ställa frågor.

6.3 Kundkontakt

Kunden i projektet har varit uppdragsgivaren. Detta har medfört en god kontakt med kunden då denne befunnit sig på samma plats och man har därför enkelt kunnat ställa frågor angående oklarheter med projektet. Efter varje sprint gavs det direkt feedback från kunden i form av beröm eller ändringar i projektet.

6.4 Kravspecifikation

Uppdragsgivaren skapade från början en skriftlig kravspecifikation på projektet i sin helhet samt en product backlog utifrån kravspecifikationen med specifika uppgifter på vad systemet skulle innehålla. Under implementeringen av projektet har det arbetats utifrån uppgifterna i product backlog samtidigt som den skriftliga kravspecifikationen har använts under skrivandet av rapporten. Det har också tillkommit uppgifter i product backlog under utvecklingscykeln vilket har varit en del av den agila utvecklingsprocessen. Uppgifterna har lagts till efter diskussion med uppdragsgivaren.

6.5 Lärdomar

Innan projektet fanns det tidigare erfarenheter om projekt som också bygger på tekniker som ASP.NET och MVC med Entity Framework. Dessa erfarenheter har dock varit andras projekt som man sedan fått bygga vidare på. Projektet Radix har istället skapats från grunden vilket har medfört en bättre förståelse för hur ett projekt är uppbyggt samt utökat kunskapen och erfarenheten om hur man använder dessa tekniker.

6.6 Möjligheter till vidareutveckling

Redan innan projektet startade var det tydligt att systemet skulle användas utav personal på Evry Forest. När grundversionen av systemet började bli färdig i början av november nämnde uppdragsgivaren att de skulle vilja börja använda systemet inom kort.

Under projektet har det hela tiden funnits i åtanke att systemet skall kunna vidareutvecklas av andra.

6.7 Slutsats

Vi är väldigt nöjda med vårt projekt hos EVRY Forest. Överlag har vi inte stött på några större problem, varken med rapportskrivandet eller med implementeringen. Vi har fått god hjälp av både uppdragsgivare och handledare. Uppdragsgivaren är mycket nöjd med vår insats och systemet vi skapat.

Referenser

[1] EVRY, ”Om EVRY,” [Online]. Available: https://www.evry.se/evry/. [Använd 25 09 2015].

[2] EVRY, ”VACS,” [Online]. Available: https://www.evry.se/globalassets/produktblad_vacs_lr.pdf. [Använd 25 09 2015].

[3] Wikipedia, ”Scrum (software development),” 07 01 2016. [Online]. Available: https://en.wikipedia.org/wiki/Scrum_%28software_development%29#Product_backlog. [Använd 09 01 2016].

[4] Wikipedia, ”Microsoft Excel,” [Online]. Available: https://sv.wikipedia.org/wiki/Microsoft_Excel. [Använd 04 10 2015].

[5] EVRY, ”Skogsindustri,” [Online]. Available: http://www.evry.se/. [Använd 29 09 2015].

[6] Microsoft, ”Visual Studio Online,” [Online]. Available: https://www.visualstudio.com/en-us/products/what-is-visual-studio-online-vs. [Använd 27 09 2015].

[7] Microsoft, ”Microsoft Dynamics AX at a glance,” [Online]. Available: https://www.microsoft.com/en-us/dynamics/erp-ax-overview.aspx. [Använd 12 11 2015].

[8] Wikipedia, ”Responsiv webbdesign,” [Online]. Available: https://sv.wikipedia.org/wiki/Responsiv_webbdesign. [Använd 03 12 2015].

[9] Wikipedia, ”.NET Framework,” [Online]. Available: https://sv.wikipedia.org/wiki/.NET_Framework. [Använd 04 10 2015].

[10] Microsoft, ”Common Language Runtime (CLR),” [Online]. Available: https://msdn.microsoft.com/en-us/library/8bs2ecf4(v=vs.110).aspx. [Använd 25 09 2015].

[11] Microsoft, ”Getting started with the .NET Framework,” [Online]. Available: https://msdn.microsoft.com/en-us/library/hh425099(v=vs.110).aspx. [Använd 25 09 2015].

[12] Wikipedia, ”Common Intermediate Language,” [Online]. Available: https://sv.wikipedia.org/wiki/Common_Intermediate_Language. [Använd 04 10 2015].

[13] Wikipedia, ”Common Language Infrastructure,” [Online]. Available: https://sv.wikipedia.org/wiki/Common_Language_Infrastructure. [Använd 25 09 2015].

[14] Wikipedia, ”ASP.NET,” [Online]. Available: https://sv.wikipedia.org/wiki/ASP.NET. [Använd 25 09 2015].

[15] Wikipedia, ”Microsoft Visual Studio,” [Online]. Available: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio#Visual_Studio_2015. [Använd 27 09 2015].

[16] Microsoft, ”Web Application Projects versus Web Site Projects in Visual Studio,” [Online]. Available: https://msdn.microsoft.com/en-us/library/dd547590(v=vs.110).aspx. [Använd 25 09 2015].

[17] Wikipedia, ”Model-View-Controller,” [Online]. Available: https://sv.wikipedia.org/wiki/Model-View-Controller. [Använd 25 09 2015].

[18] Microsoft, ”ASP.NET MVC Overview,” [Online]. Available: https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx. [Använd 25 09 2015].

[19] Wikipedia, ”ASP.NET MVC Framework,” [Online]. Available: https://sv.wikipedia.org/wiki/ASP.NET_MVC_Framework. [Använd 25 09 2015].

[20] Wikipedia, ”HTML,” [Online]. Available: https://sv.wikipedia.org/wiki/HTML. [Använd 04 10 2015].

[21] Wikipedia, ”Windows Azure Platform,” [Online]. Available: https://sv.wikipedia.org/wiki/Windows_Azure_Platform. [Använd 25 09 2015].

[22] Microsoft, ”Microsoft Azure,” [Online]. Available: http://azure.microsoft.com/sv-se/overview/what-is-azure/. [Använd 26 09 2015].

[23] EntityFrameworkTutorial, ”What is Entity Framework?,” [Online]. Available: http://www.entityframeworktutorial.net/what-is-entityframework.aspx. [Använd 26 09 2015].

[24] Wikipedia, ”ORM,” [Online]. Available: https://sv.wikipedia.org/wiki/ORM. [Använd 05 09 2015].

[25] Microsoft, ”Code First to a New Database,” [Online]. Available: https://msdn.microsoft.com/en-us/data/jj193542.aspx. [Använd 26 09 2015].

[26] Bootstrap, ”Bootstrap,” [Online]. Available: http://getbootstrap.com/. [Använd 26 09 2015].

[27] Wikipedia, ”Cascading Style Sheets,” [Online]. Available: https://sv.wikipedia.org/wiki/Cascading_Style_Sheets. [Använd 04 10 2015].

[28] Wikipedia, ”Javascript,” [Online]. Available: https://sv.wikipedia.org/wiki/Javascript. [Använd 04 10 2015].

[29] Wikipedia, ”Jquery,” [Online]. Available: https://sv.wikipedia.org/wiki/Jquery. [Använd 04 10 2015].

[30] E. Charbeneau, ”Bootstrap for ASP.NET MVC,” [Online]. Available: http://responsivemvc.azurewebsites.net/Bootstrap. [Använd 26 09 2015].

[31] Wikipedia, ”C (programspråk),” [Online]. Available: https://sv.wikipedia.org/wiki/C_(programspr%C3%A5k). [Använd 04 10 2015].

[32] Wikipedia, ”C++,” [Online]. Available: https://sv.wikipedia.org/wiki/C%2B%2B. [Använd 04 10 2015].

[33] Wikipedia, ”C-sharp,” [Online]. Available: https://sv.wikipedia.org/wiki/C-sharp. [Använd 04 10 2015].

[34] Wikipedia, ”Visual Basic,” [Online]. Available: https://sv.wikipedia.org/wiki/Visual_Basic_.NET. [Använd 04 10 2015].

[35] Wikipedia, ”F Sharp,” [Online]. Available: https://sv.wikipedia.org/wiki/F_Sharp. [Använd 04 10 2015].

[36] Wikipedia, ”Git,” [Online]. Available: https://sv.wikipedia.org/wiki/Git. [Använd 27 09 2015].

[37] Wikipedia, ”Team Foundation Server,” [Online]. Available: https://en.wikipedia.org/wiki/Team_Foundation_Server. [Använd 04 10 2015].

[38] git, ”git,” [Online]. Available: https://git-scm.com/. [Använd 28 09 2015].

[39] git, ”About,” [Online]. Available: https://git-scm.com/about/. [Använd 28 09 2015].

[40] Wikipedia, ”CRUD,” [Online]. Available: https://en.wikipedia.org/wiki/Create,_read,_update_and_delete. [Använd 25 11 2015].

[41] T. Dykstra, ”ASP.NET,” Microsoft, 14 02 2014. [Online]. Available: https://www.asp.net/mvc/overview/getting-started/getting-started-with-ef-using-mvc/implementing-basic-crud-functionality-with-the-entity-framework-in-asp-net-mvc-application. [Använd 24 11 2015].

[42] Wikipedia, ”Uniform Resource Locator,” [Online]. Available: https://sv.wikipedia.org/wiki/Uniform_Resource_Locator. [Använd 26 11 2015].

[43] Wikipedia, ”NuGet,” [Online]. Available: https://en.wikipedia.org/wiki/NuGet. [Använd 01 12 2015].

[44] Microsoft, ”IQueryable,” [Online]. Available: https://msdn.microsoft.com/en-us/library/system.linq.iqueryable(v=vs.100).aspx. [Använd 01 12 2015].

[45] Microsoft, ”IEnumerable,” [Online]. Available: https://msdn.microsoft.com/en-us/library/system.collections.ienumerable(v=vs.110).aspx. [Använd 01 12 2015].

[46] Microsoft, ”SelectListItem Class,” [Online]. Available: https://msdn.microsoft.com/en-us/library/system.web.mvc.selectlistitem(v=vs.118).aspx. [Använd 06 01 2016].

[47] Microsoft, ”SelectList Class,” [Online]. Available: https://msdn.microsoft.com/en-us/library/system.web.mvc.selectlist(v=vs.118).aspx. [Använd 06 01 2016].

[48] w3schools.com, ”ASP.NET MVC - HTML Helpers,” [Online]. Available: http://www.w3schools.com/aspnet/mvc_htmlhelpers.asp. [Använd 06 01 2016].

[49] R. A. T. D. J. G. Pranav Rastogi, ”Introduction to ASP.NET Identity,” Microsoft, 17 10 2013. [Online]. Available: http://www.asp.net/identity/overview/getting-started/introduction-to-aspnet-identity. [Använd 02 12 2015].

[50] O. Hallberg, ”Skapa webbformulär,” Karlstad Universitet, 30 05 2013. [Online]. Available: http://www.kau.se/drupal/skapa-webbformular. [Använd 08 12 2015].

[51] Microsoft, ”ASP.NET Web API,” Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/hh833994(v=vs.108).aspx. [Använd 08 12 2015].

[52] M. Wasson, ”Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET,” 11 2013. [Online]. Available: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx?f=255&MSPPError=-2147217396. [Använd 08 12 2015].

[53] Microsoft, ”StreamReader Class,” [Online]. Available: https://msdn.microsoft.com/en-us/library/system.io.streamreader(v=vs.110).aspx. [Använd 07 01 2016].

[54] Microsoft, ”Substring Method,” Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/aka44szs(v=vs.110).aspx. [Använd 07 01 2016].

[55] Microsoft, ”MSDN Microsoft,” 2015. [Online]. Available: https://msdn.microsoft.com/en-us/library/system.web.mvc.controllerbase.viewbag%28v=vs.118%29.aspx?f=255&MSPPError=-2147217396. [Använd 26 11 2015].

[56] Microsoft, ”Introduction to Membership,” Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/ms731049(v=vs.110).aspx. [Använd 03 12 2015].

A Bilaga

Kravspecifikation av Torbjörn Stake.

A.1 Inledning

Detta dokument är den överordnade kravspecifikationen för *projekt Radix*, EVRY Forest & Logistics system för produkt-och modulsammanställning.

A.2 Bakgrund

A.2.1 Produktutveckling och systemet VACS

Basen i EVRY:s skogliga affär är systemet VACS. VACS är ett system som utvecklats och byggts ut under 25 års tid. Det har genomgått ett antal teknikskiften genom åren och skapades ursprungligen för att köras i AS/400. Runt år 2000 lyftes systemet till SQL Server och klient i Centura. Då introducerades också namnet VACS (VirkesAdministrativ Client/Server). År 2006 tekniklyftes VACS klient från Centura till .NET och där befinner sig systemet idag, rent tekniskt.

Under de 25 år som systemet existerat har kundbasen vuxit från ca 5 kunder till ca 30 kunder. Under dessa år har också antalet anpassningar i systemet eskalerat. Varje anpassning betalas av den kund som köpt den, både i införandeavgift och därefter en årlig underhållsavgift.

Förutom anpassningar har VACS fått ett antal delsystem eller moduler utvecklade under systemets livscykel. Dessa moduler är mer omfångsrika och kapslar lite större arbetsflöden eller funktionsgrupper. Modulerna är i högre eller lägre grad fristående från kärnsystemet och köps enligt licensavgift och underhållsavgift.

A.2.2 Problemformulering

Att ha ordning på alla de anpassningar som utvecklats för alla kunder, och alla de moduler som finns i och runt VACS är viktigt ur flera perspektiv.

4. Att veta vilken kund som har installerat anpassningar och vilka de är, är viktigt för att faktureringen av den årliga underhållsavgiften ska bli rätt.
5. Att veta vilken kund som köpt vilka moduler är viktigt för att faktureringen av den årliga underhållsavgiften ska bli rätt.
6. Att veta vilken kund som har anpassningar, moduler och *vilken version* är viktigt för utvecklare och produktägare eftersom det får konsekvenser i utvecklingscykeln av VACS och kvalitetssäkrande åtgärder.

Idag görs för ekonomiskt syfte en årlig genomgång av kunderna, deras anpassningar och köpta moduler. Informationen sammanställs med hjälp av manuell inventering och i Excel. Informationen utgör faktureringsgrundande underlag.

Utvecklare och produktägare måste vid givna tillfällen göra djupgående analyser för att få reda på detaljer om en specifik kunds installation. Detta görs behovsstyrt och när applikationens livscykel så kräver (ny release, ny installation av programkod hos kund, Change Request från kund etc)

EVRY Forest & Logistic Solutions AB söker ett bättre verktygsstöd för att hålla ordning på relationerna mellan kund, system, version, anpassning och modul, vilket avses lösas inom ramen för projekt Radix.

A.3 Primära roller och övergripande funktionella krav

A.3.1 Säljchef

Säljchefen behöver ett systemstöd för att snabbt kunna bilda sig en uppfattning om:

- Vilka kunder kör VACS?
- Vilka anpassningar har en kund köpt till VACS?
- Vilka moduler har en kund köpt till VACS?
- Vilka kunder har en specifik modul?
- Vilken version av VACS körs hos vilka kunder?
- (Pris på modul och pris på anpassning önskvärt)

A.3.2 Produktchef

Produktchefen behöver ett systemstöd för att snabbt kunna bilda sig en uppfattning om:

- Vilka versioner av VACS finns?
- Vilka kunder har en specifik version av systemet?
- Vilka moduler finns till VACS?
- Vilka beroenden finns mellan olika moduler?
- Vilka moduler är kompatibla med respektive VACS-version?

A.4 Övergripande tekniska krav

A.4.1 Skallkrav

- Systemet ska inte kräva en programinstallation på PC eller mobil enhet utan konsumeras förslagsvis som en webapplikation.
- Systemet ska endast kunna användas av en användare som finns upplagd i applikationen.
- Data ska lagras i en relationsdatabas.

- Kommunikation mellan grafiskt gränssnitt och databas ska ske genom ett affärslogiskt lager med tjänster.
- Affärslogiktjänster ska exponeras på ett vis som möjliggör att en mobil app kan konsumera och tillföra data i databasen i ett senare skede (utanför projektets scope).

A.4.2 Börkrav

- Systemet bör designas med hjälp av moderna molntjänster för att maximera utvecklingsinsatsen.
- Grafiskt gränssnitt bör stödja responsivitet och kunna köras på ett användarvänligt sätt även från en mobil enhet (tabletstorlek)
- Systemet bör designas med hjälp av Microsoftteknologi i så många led det går då det stöder EVRY Forest & Logistics utvalda strategiska plattform

A.5 Övriga funktionella krav

En högst önskvärd funktion i det tänkta systemet är att med automatik gå igenom källkoden för VACS och identifiera dimensionerna *kund* och *anpassning*. Dessa dimensioner bör då kunna laddas i Radix Databas utan att en användare manuellt lägger in dessa.

Ett grafiskt gränssnitt där en användare (autentiserad) kan underhålla data ska möjliggöra följande funktioner:

- Skapa/ändra och ta bort kund
- Skapa/ändra och ta bort VACS-version
- Skapa/ändra och ta bort modul
- Skapa/ändra och ta bort anpassning
- Skapa/ändra och ta bort relation mellan moduler/anpassningar/VACS

A.6 Projektstyrning

EVRY Forest & Logistic Solutions AB tillhandahåller en projektplats i Visual Studio Online och ett repository för att lagra källkod. Vi förutsätter att studenterna har tillgång till egna PC:s för utveckling, och med installationer av t ex Visual Studio.

Projektet drivs enligt en övergripande projektplan som studenterna tar fram i samråd med EVRY:s produktägare, och delas in i ett antal sprintar. Innehåll i varje sprint bestäms mellan team (studenter) och EVRY:s produktägare. Features bryts ner, läggs ut i Scrum Board och följs upp på regelbunden basis. Efter varje sprint genomförs en demo av implementerade funktioner.

A.7 Testning

EVRY:s produktägare bistår med funktionell testning av systemet. I övrigt ser vi mycket positivt på användning av enhetstester och vi föreslår att all affärslogik testas med hjälp av

unit-testramverk så som MSTest. Studenterna väljer själva om de vill genomföra kodningen testdrivet.