Examensarbete, Högskolan på Åland, Utbildningsprogrammet för informationsteknik

Utveckling av webbapplikation samt databas med Entity Framework

- För enhetshantering hos Alandia försäkring

Dennis Hellen

2023:34

Datum för godkännande: 16.11.2023

Handledare: Björn-Erik Zetterman

EXAMENSARBETE

Högskolan på Åland

Utbildningsprogram:	Informationsteknik
Författare:	Dennis Hellen
Titel:	Utveckling av webbapplikation samt databas med Entity Framework - För enhetshantering hos Alandia försäkring
Handledare:	Björn-Erik Zetterman
Uppdragsgivare:	Alandia Försäkring - Infrastruktur

Abstrakt

I detta examensarbete beskriver jag mitt programmeringsprojekt med C# ramverket Entity Framework. Syftet med arbetet var att få fördjupade kunskaper i C# och lära mig ett nytt ramverk samt utveckla en applikation åt min uppdragsgivare. Enhetshanteringsapplikationen skall kunna skapa kopplingar mellan en enhet och en anställd och editera dem samt spara ner dem i en SQL-databas, man skall även kunna göra sökningar mot databasen. Arbetet använder sig av C#, och mer specifikt ramverken Entity Framework och ASP.NET samt HTML, CSS och JavaScript. Mitt resultat är att applikationen uppfyllde alla krav som uppdragsgivaren ställde. Jag har även fått fördjupade kunskaper i databashantering, C# och utveckling av C# projekt och fått en bra förståelse för hur utveckling av applikationer i C# går till.

Nyckelord (sökord)

Utveckling av webbapplikation, Entity Framework, C#, Enhetshantering

Högskolans serienummer:	ISSN:	Språk:	Sidantal:
2023:34	1458-1531	Svenska	36 sidor

Inlämningsdatum:	Presentationsdatum:	Datum för godkännande:
15.9.2023	29.9.2023	16.11.2023

DEGREE THESIS

Åland University of Applied Sciences

Degree Programme:	Bachelor of Information Technology
Author:	Dennis Hellen
Title:	Development of web application and database with Entity Framework- For device management at Alandia insurance
Academic Supervisor:	Björn-Erik Zetterman
Commissioned by:	Alandia Insurance - Infrastructure

Abstract

In this degree thesis, I describe my programming project using the C# framework, Entity Framework. The purpose of this project was to gain in-depth knowledge of C# and learn a new framework as well as develop an application for my client. The device management application must be able to create connections between a device and an employee and edit them as well as save them in an SQL database, you must also be able to perform searches against the database. The project uses C# and more specifically its framework, Entity Framework, as well as HTML, CSS, and JavaScript. My result is that the application met the requirements set by the client. I have also gained in-depth knowledge in Database management, C#, and the development of C# projects and gained a good understanding of how the development of applications in C# is done.

Keywords

Development of a web application, Entity Framework, C#, Device management

Serial number:	ISSN:	Language:	Number of pages:
2023:34	1458-1531	Swedish	36 pages

Handed in:	Date of presentation:	Approved:
15.09.2023	29.9.2023	16.11.2023

INNEHÅLLSFÖRTECKNING/TABLE OF CONTENTS

1 INTRODUKTION

I detta kapitel introduceras läsaren till uppsatsen, och syfte, metod och avgränsningar klargörs.

1.1 Inledning

Den här uppsatsen handlar om utvecklingen av en webbapplikation samt konstruktion av tillhörande databas för det åländska bolaget Alandia försäkring Abp . Applikationen är uppbyggd med hjälp av C#, HTML, CSS, JavaScript. Uppsatsen kommer att gå igenom utvecklingsprocessen av applikationen, databasen och teorin bakom den funktionalitet som används. Applikationen som byggts är en webbapplikation där man kan knyta en enhet såsom en mobiltelefon eller laptop till en anställd för att hålla koll på vem som innehar vilken/vilka enheter. Detta sparas sedan ner i en databas som har en lista med alla enheter som existerar och en lista med alla anställda. Sedan finns också en adminsida som bara företagets infrastrukturavdelning har tillgång till där de kan lägga till och uppdatera värden i de olika listorna, såsom enhetstyp, plats och avdelning, i databasen utan att behöva fysiskt ha tillgång till databasen.

1.2 Syfte

Syftet med arbetet är att beskriva utvecklingsprocessen av webbapplikationen som uppdragsgivaren Alandia Försäkrings infrastrukturavdelning tilldelat och förklara teorin samt teknikerna som används för att utföra utvecklingen av den här applikationen.

1.3 Metod

Metoden för detta projekt var en kravställningsdialog som gjordes tillsammans med infrastrukturavdelningen, då jag hade fått det som uppdrag under praktiken var jag inte under någon större tidspress och jag kunde därför bestämma planering över arbetstid och sluttid själv. Jag delade upp det i webbapplikation och databas och sedan delade jag upp applikationen i “add” där man knyter en enhet till en anställd, “search” där man söker upp enheter och visar deras information,”update” från search så kan man länka tillbaka till add där man kan göra uppdateringen av en enhet, “admin” där man kan lägga till och uppdatera de olika listorna i databasen såsom enhetstyp, plats och avdelning utan att behöva fysiskt ha tillgång till databasen.

1.4 Beställning

Beställaren av mjukvaran ville ha en sida där de kunde registrera information för en enhet och dess anställda. På den sidan ville de kunna lägga till information och markera den för radering, vilket betyder att den ska döljas i gränssnittet. Sedan ville beställaren ha en sida där de kunde söka och filtrera alla poster från databasen. De bestämde även att färgschemat skulle vara i Alandias färger, men utöver det var det ganska fritt att designa applikationen. Mot slutet av projektet kom ett till krav om en administratörssida där beställaren skulle kunna uppdatera och lägga till värden i dropdownlistorna, så att de inte skulle behöva gå in i databasen och lägga till värdena manuellt.

1.5 Avgränsningar

Applikationen kommer inte att vara anpassad till mobila enheter, även om det skulle gå att göra, eftersom det skulle komplicera uppbyggnaden av layouten. Applikationen använder sig av CSS-ramverket Bootstrap och det skulle gå att anpassa för mobila enheter men det skulle ta allt för mycket tid att få layouten korrekt för alla olika enheter.

2 TEORI

I detta kapitel ges läsaren en överblick av de tekniker och ramverk som används.

2.1 HTML

Hypertext Markup Language eller HTML är ett så kallat markup-språk som tillåter en användare att visa och strukturera text, bilder och så vidare för uppbyggnad av webbsidor (Winter, 2023). HTML som ett språk är skapat för att vara läsligt för människor, alltså inte bara en vägg av kod som är oläslig för någon som inte är insatt i projektet (Whitehead & Russell, 2005). HTML är också relativt lätt att lära sig eftersom det är väl dokumenterat och innehåller inte så mycket svårförståelig funktionalitet (Whitehead & Russell, 2005). Tillsammans med CSS visar man det grafiska av en webbsida.

2.2 CSS

Cascading style sheets eller CSS är style sheet-språk som beskriver hur ett dokument som har skrivits med ett markup-språk såsom HTML presenteras visuellt (Whitehead & Russell, 2005). CSS introducerades först 1996 och fann begränsat stöd i Internet Explorer 3 men det var inte förrän år 2000 som CSS fick fullt stöd i browsern (Kte'pi, 2023). CSS tillsammans med HTML och Javascript tillåter programmerare att med ett fåtal rader i CSS-koden ändra på utseendet på en webbsida. Om man skulle göra samma ingrepp med hjälp av HTML så skulle det ta mycket mera kod och arbete (Whitehead & Russell, 2005). Nuförtiden består de flesta webbsidor av en del CSS, CSS är även det vanligaste style sheet-språket att använda för webbapplikationer (Kte'pi, 2023). CSS har också en viss versatilitet gällande olika layouts, det vill säga man kan skapa olika style sheets för samma webbsida. Den vanligaste anledningen till detta är att skapa layouter för olika enheter, till exempel mobil och desktop.

2.3 Javascript, Jquery och Ajax

Javascript är ett interpreterande programmeringsspråk, vilket innebär att koden exekveras rad för rad, jämfört med kompileringsspråk såsom C# som översätts till maskinkod som sedan

kompileras. Vad Javascript gör lättare är att skapa interaktiva webbsidor som är dynamiska och responsiva (Haverbeke, 2014). De flesta browsers i dagens läge har stöd för Javascript vilket gör det till det mest behändiga och logiska språket att använda vid skapandet av en webbaserad applikation (Chmiel, 2022). Javascript tillåter programmeraren att använda sig av funktionalitet såsom animationer och interaktiva element (knappar, textrutor osv.). Javascript är ett mångsidigt språk som kan användas på både klient- och server-sidan av webbapplikationen. Allt detta har gjort att Javascript är ett av de mest använda programmeringsspråken i världen (Chmiel, 2022).

Jquery är ett Javascript-bibliotek som förenkla processen för att skapa interaktiva och dynamiska webbsidor. Jquery används för att underlätta användningen av Javascript vid till exempel manipuleringen av HTML-dokument, skapandet av animeringar och eventhantering (York, 2015). Jquery simplifierar koden och minskar den mängd Javascript som behövs men behåller samma funktionalitet, vilket leder till effektivare kod som är lättare att underhålla (Hong, 2018). Med Jquery så kan man utföra Javascript-funktionalitet med endast några få rader. Detta leder till att Jquery är ett mycket vanligt verktyg att använda vid Javascript-programmering (York, 2015).

Ajax (Asynchronous JavaScript and XML) är ett programmeringsverktyg som används vid webbutveckling för att skapa dynamiska och interaktiva webbapplikationer (Clark, 2006). Verktyget möjliggör asynkrona datautbyten mellan klient- och serversidan, detta tillåter webbsidan att göra uppdateringar och hämta innehåll utan att behöva utföra en omladdning av sidan. Ajax utnyttjar en kombination av teknologier, som inkluderar Javascript, XML, HTML och CSS. Genom att göra asynkrona requests till en server tillåter man utvecklaren att hämta, uppdatera och skicka dataspecifika delar av webbsidan i realtid (West, 2006). Ajax är viktigt i utvecklingen av moderna webbapplikationer med funktionalitet som realtidsnotifikationer och livesökning och så vidare. Med Ajax så kan utvecklare leverera en smidigare och behagligare användarerfarenhet.

2.4 Bootstrap

Bootstrap är ett populärt frontend rameverk som används för att bygga responsiva och visuellt tilltalande webbgränssnitt (Hong, 2018). Ramverket tillhandahåller en samling av fördesignade HTML, CSS och JavaScript-komponenter, som tillåter utvecklare att lättare skapa webbsidor med mindre mängd kod för mer komplex funktionalitet (Thornton et al., n.d.). Bootstrap erbjuder ett responsivt rutsystem som automatiskt justerar layouten och dimensioneringen av element baserat på enhetens skärmstorlek, vilket leder till att utvecklare inte behöver skapa flera olika vyer för olika skärmstorlekar. Ramverket inkluderar också UI-komponenter såsom knappar, formulär och navigationsfält. Bootstrap är väl dokumenterat och är ett bra sätt att effektivisera webbutveckling (Hong, 2018).

2.5 WebGrease

WebGrease är ett ramverk för CSS-optimering som vanligtvis används vid webbutveckling. Ramverket erbjuder verktyg och hjälpmedel för att optimera och förbättra prestanda för CSS-filer. WebGrease erbjuder funktionalitet såsom minimering, paketering och komprimering av CSS-filer för att reducera deras storlek och förbättra laddningstider (US Government, 2023). Ramverket hjälper till att effektivisera CSS-filer genom att ta bort onödig kod, kommentarer och *whitespace*. Det finns också stöd för sammanslagning av CSS-filer för att minska antalet HTTP-förfrågningar som behövs för att ladda stylesheets. WebGrease möjliggör användandet av CSS sprites som är en teknik där man kombinerar flera bilder till en enda bild, vilket minskar antalet bildförfrågningar och förbättrar sidoladdningshastigheten. WebGrease integrerar också mycket bra med Asp.NET och kan lätt integreras i byggnadsprocessen med dess optimeringsförmåga. WebGrease bidrar till förbättrad webbsideprestation, minskad bredbandsanvändning och förbättrar användarupplevelsen (US Government, 2023).

2.6 Programmeringsspråk C#

C# är ett programmeringsspråk som har sitt ursprung i C. C# har därför mångsidigheten av C och och den tillagda funktionaliteten objektorientering. Objektorientering är en typ av programmering där man definierar datatyper och deras beteende för att göra datastrukturen till ett objekt (Microsoft C#, 2023). På grund av detta har C# versatilitet, kraft och simplicitet, i språket finns ett brett utbud av funktionalitet som kan utföra en mängd olika uppgifter. C#-program är lätta att förstå sig på eftersom språket är designat för att vara det (Turtschi et al, 2002). C# är ett av de mest använda språken i världen, till exempel Microsofts använder sig delvis av C#.

2.7 ASP.NET

ASP.NET är ett kraftfullt webbutvecklingsramverk utvecklat av Microsoft (Microsoft ASP.NET, n.d.). Ramverket förser utvecklare med en mängd verktyg och bibliotek för att bygga skalbara webbapplikationer. ASP.NET följer MVC arkitekturmönsterstrukturen för att separera applikationslogiken, presentationen och datahanteringen från varandra. Ramverket stödjer flera programmeringsspråk, av vilka C# är ett av de vanligaste. ASP.NET erbjuder en rad av inbyggd funktionalitet så som controller, autentiseringsmekanismer och alternativ för dataåtkomst för att utveckla säkra och data drivna applikationer (Gaylord et al., 2013). Ramverket har också stöd för HTML, CSS och JavaScript som tillåter utvecklare att skapa moderna och interaktiva gränssnitt (Microsoft ASP.NET, n.d.). ASP.NET har funktionalitet så som caching, sessionshantering och felhantering, vilket leder till förbättrad prestanda. Ramverket stöder även cross-plattform utveckling genom .NET core, vilket tillåter utvecklare att bygga applikationer att köra på Windows, Linux och macOS(Gaylord et al., 2013). ASP.NET är därför ett populärt val för att bygga webbapplikationer .

2.8 Entity Framework

Entity framework är ett verktyg i C# eller .NET programmering som förenklar databasåtkomst och -hantering genom object-relational mapping (ORM) möjligheter (Kanjilal, 2015). .NET är ett cross-platform mjukvaruramverk som är utvecklat av Microsoft som kan användas med ett flertal kodspråk (Microsoft .NET., u.å.). Vad som gör att Entity framework är ett så användbart verktyg är att det först och främst abstraherar databaser, vilket betyder att entitetsklasserna utgör en objektorienterad motsvarighet till datamodellen i relationsdatabasen. Sedan tar Entity framework hand om att översätta objektorienterade strukturer till databastabeller och frågor (Kanjilal, 2015). Detta betyder att utvecklaren inte behöver skriva så mycket kod för databasoperationer, vilket gör utvecklingen snabbare och mer effektiv. Entity framework har stöd för många olika databasleverantörer såsom Microsoft SQL, MySQL med mera. Verktyget har även tillgång till annan funktionalitet såsom lazy loading och databasmigration, vilket leder till en mer en effektiverad utvecklingsprocess (Peres, 2016).

2.9 MVC

Model view controller (MVC) är ett mjukvaruarkitekturmönster som är mycket vanligt vid programmering. Målet med detta arkitekturmönster är att dela upp programmet i datamanipulation, användargränssnitt och applikationslogik (Microsoft MVC, 2022). Model hanterar data- och businesslogiken, view hanterar presentationen och rendering av användargränssnitt och controllern agerar som en brygga mellan view och model, hanterar användarinput och uppdatering av view och model när det behövs. Separationen av ansvar främjar modularitet, återanvändbarhet och underhållbarhet av kodbasen. MVC tillåter utvecklare att arbeta på olika aspekter av applikationen självständigt från varandra, vilket gör det lättare att hantera komplexitet och göra ändringar utan att det påverkar andra delar av systemet (Microsoft MVC, 2022). MVC är ett mycket användbart verktyg på grund av ovan nämnd funktionalitet samt det gör koden mer städad och lättare att förstå.

2.10 Microsoft Visual Studio och SQL Server Management Studio

Microsoft Visual Studio 2022 är en av de kodmiljöer som Microsoft erbjuder. Visual studios funktionalitet är riktad främst mot applikationer byggda med .NET eller versioner av C-språk, och erbjuder inbyggd versionshantering samt Azure deployment, vilket innebär att oberoende om en användare distribuerar sitt program på ett internt eller publikt nätverk kan detta göras via Visual Studio. Visual Studio har också verktyg som underlättar koppling till SQL-databaser. Sedan finns det även en inbyggd funktionalitet som tillåter användare att starta upp sina webbapplikationer i valfri webbläsare för att testa dem och visuellt se slutresultatet (Microsoft Visual Studio, 2023).

Microsofts SQL Server Management Studio (SSMS) är ett verktyg för att skapa SQL relationsdatabaser. SSMS kan utföra ett brett sortiment av tjänster såsom konfigurering och säkrandet av en SQL server-instans. SMSS kan även skapa och köra frågor samt förser användaren med funktionalitet som kan utföra backup och återställning, man kan också lägga till regler för databasen för att till exempel bestämma när backuper sker och vem (i till exempel ett nätverk) som får göra vad i databasen (Microsoft SSMS, 2023).

3 IMPLEMENTATION

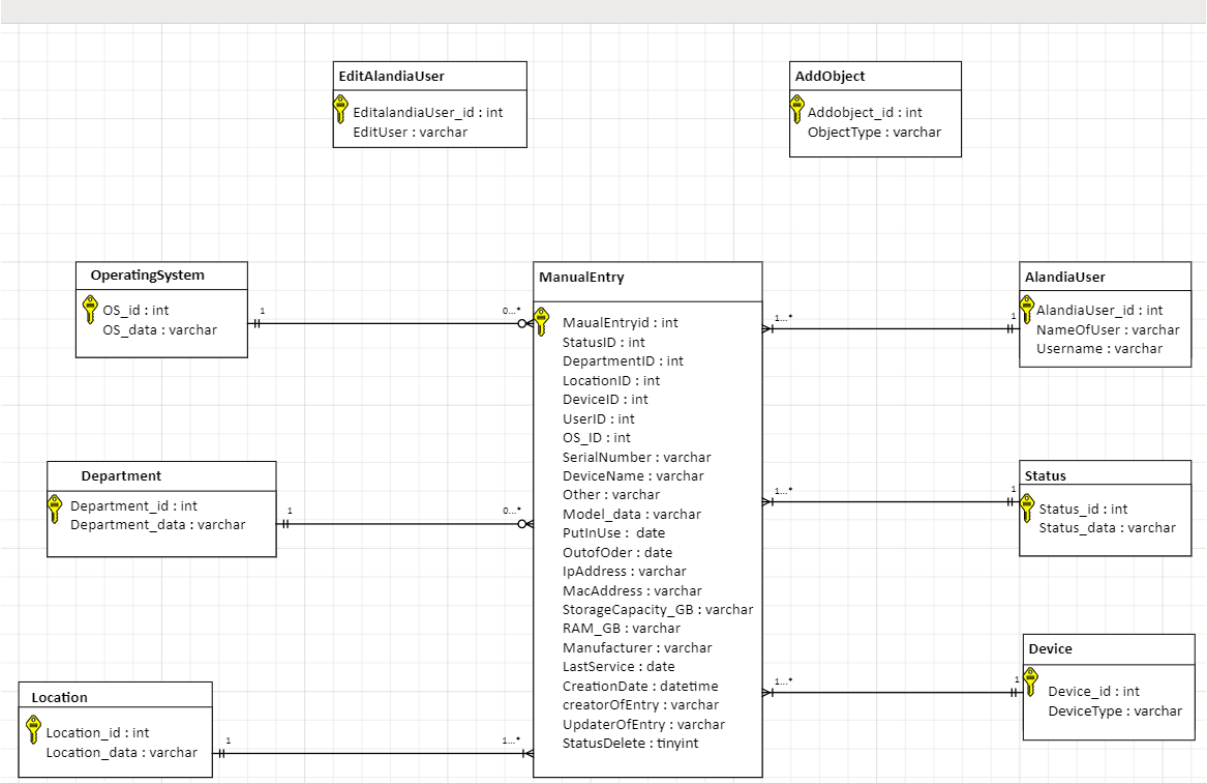
I detta kapitel beskrivs implementeringen av mjukvaran, vilka steg och komponenter som används samt hur det hela binds ihop till en helhet.

3.1 Förberedelser

Jag fick först en produktbeställning av Infrastrukturavdelningen på Alandia försäkringar, där de gav mig krav på vad som behövde finnas i applikationen samt en standard för hur den skulle se ut grafiskt. Sedan ritade jag upp en skiss på hur användargränssnittet skulle se ut och började göra research på ramverket "Entity framework" som jag hade fått i uppgift att använda av min uppdragsgivare. Efter att jag gjort detta startade jag upp ett projekt i Visual Studio och började arbeta.

3.2 Databasstruktur

En av de första saker som jag gjorde var att skapa databasstrukturen i "Microsoft SQL Server Management". Först skapade jag strukturen för huvudtabellen, där all data som handlar om enheter (device) ska samlas. Sedan skapade jag upp hjälptabeller för listor som kommer att användas i applikationer så att jag inte behövde hårdkoda in värden för de listor som jag behöver, till exempel land, avdelning, typ (devicetyp) och så vidare. Senare under projektet skapade jag även en hjälptabell som innehåller användares namn och användarnamn. Först skapades tabellerna i en testserver för att sedan bli överförda till en produktionsserver när applikationen tas i produktion.



Figur 1. Bild på ett ER-diagram över databasstrukturen

3.3 Mappning av databasvärden och Anslutning till databas

Efter skapandet av databasstrukturer läste jag på om hur man skapar en databasanslutning i Entity Framework. Jag kom fram till att man först och främst behöver skapa en entitetsklass såsom “ManualEntry” av de värden som finns i de olika tabellerna, med korrekta datatyper, korrekta egenskaper för de olika värdena såsom primärnycklar och främmande nycklar och eventuella kopplingar mellan tabellerna. Ett exempel på en modell kan ses i Figur 2, där det finns **[Key]** som är en annotationr för en primärnyckel i den aktuella modellen och **[ForeignKey]** som är kopplingar till en annan tabell.

```
public class ManualEntry
{
    [Key]
    public int ManualEntryId { get; set; }
    public int DeviceID { get; set; }
    [ForeignKey(nameof(DeviceID))]
    public Device Device { get; set; }
    public string DeviceName { get; set; }
    public string Model_data { get; set; }
```

Figur 2. Del av en modell

Sedan skapade jag en DbContext class som används för att koppla modellen till kontrollern och vyn, exempel på en DbContext kan ses i figur 3. I Context-klassen gör man en DbSet för att via de olika entitetsklasserna utföra skapa, läsa, uppdatera och ta bort operationer. Med hjälp av en *connectionstring* (se figur 4) som man får från web.config-filen kan man ansluta till databasen. Efter detta så kan man i andra klasser initialisera DbContext-klassen och sedan använda namnet som man gav DbSet variabeln för de olika satta modellerna för att komma åt data från SQL-servern.

```
public class DatabaseContext : DbContext
{
    public DatabaseContext() : base("DatabaseContext")
    {
    }
    public DbSet<Device> Devices { get; set; }
    public DbSet<Location> Locations { get; set; }
    public DbSet<Department> Departments { get; set; }
    public DbSet<Status> Statuses { get; set; }
    public DbSet<OperatingSystem> OSs { get; set; }
    public DbSet<AlandiaUser> Users { get; set; }
    public DbSet<AddObject> Objects { get; set; }
    public DbSet<EditAlandiaUser> EditUsers { get; set; }
    public DbSet<Models.ManualEntry> manualEntries { get; set; }
```

Figur 3. Bild på DbContext klassen

```
<connectionStrings>
  <add name="DatabaseContext" connectionString="Data Source=tcp:mytestapp;Initial Catalog=myapps;
      User ID=hablahabla==; Password=habblahaba=;Persist Security Info=True;Application Name=myappsdev;"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

Figur 4. Exempel på en connectionstring

3.4 Säkerhet

Jag har implementerat säkerhetsåtgärder för att göra applikationen mindre sårbar. Det som jag har implementerat är funktionalitet som gör att man först måste vara inloggad som en Alandia Active Directory inloggning, men det är bara en specifik grupp vars inloggningar har tillgång till applikationen, till exempel “Sven “ på ekonomi har inte tillgång till den, men hela IT-avdelningen har det. Hur detta går till kan ses i figur 5, jag kan dock inte gå in djupare på hur funktionaliteten fungerar eftersom det är Alandias egna kod.

```
private bool HasAccess(ApplicationUser user)
{
    Boolean hasAccess = false;
    var appName = ConfigurationManager.AppSettings["ApplicationName"];
    if (appName == null)
    {
        return false;
    }
    var systemAccesses = SystemAccessCollection.GetListStatic(new SystemAccess() { Value = appName, Type = "Application", IsActive = true });
    foreach (var systemAccess in systemAccesses)
    {
        if (systemAccess.GroupType.Equals("Group", StringComparison.InvariantCultureIgnoreCase))
        {
            if (user.IsMemberOf(systemAccess.GroupNeeded))
            {
                hasAccess = true;
                break;
            }
        }
        else if (systemAccess.GroupType.Equals("User", StringComparison.InvariantCultureIgnoreCase))
        {
            if (systemAccess.GroupNeeded.Equals(user.LoginName))
            {
                hasAccess = true;
                break;
            }
        }
    }
    return hasAccess;

}
```

Figur 5. Bild som visar hur applikationen har koll på vem som har åtkomst till den

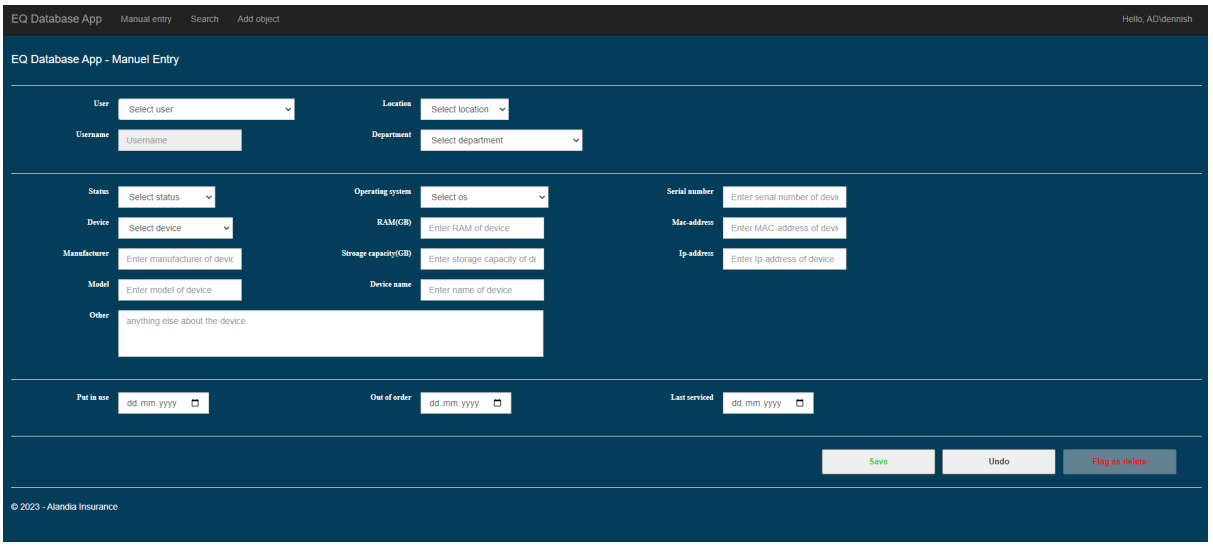
Sedan ser jag till att lösenord för databaskopplingen är krypterade i koden och att ingen känslig data står i projektet som klar text, så att ingen utomstående kan ta sig in i systemet och få åtkomst till hela Alandias databassystem via applikationen. Krypteringen går till genom att jag först körde applikationen med Alandia egna krypteringsfunktionalitet och sedan varje gång som man behöver ha anslutningssträngen så avkrypterar jag den med en hjälpklass som utför alandias egna avkrypteringsfunktionalitet på strängen.

3.5 Model, View och Controller

Projektet är uppdelat i model, view och controller vilket betyder att jag delat upp projektet i det grafiska, backend och styr över bägge med hjälp av controllern. Jag har sen olika vyer för de olika sidorna plus att jag även har en partiell vy för min visning av sökresultaten på sökvyn. En partiell vy är en vy som renderar en del av vad som visas på en sida, detta gör att den kan användas i flera vyer eller använda en annan modell än parent vyn. Under model så har jag alla modeller för mappning för databastabellerna och backend för att genomföra min applikations funktionalitet vilket också är uppdelad i skilda klasser för de olika vyerna. Sedan sköter controllern om vilken sida som skall aktiveras och när funktionaliteten skall utföras. En bild på ett Klassdiagram för att ge översikt kan ses i bilaga 1.

3.6 Startsidan

På den första sidan fyller man i värden för att sedan spara ner poster till databasen, sidan har 7 olika funktioner. Den färdiga produkten ses i figur 6.



Figur 7. Bild som visar hur homepage sidan ser ut

Den första funktione **prepareList()** som kan ses i figur 8, initialiserar listorna med värdena från databasens tabeller så att de motsvarar de listor som skapas i vyn.

```
public void PrepareLists()
{
    EQDatabaseApp.DAL.DatabaseContext databaseContext = new DAL.DatabaseContext();
    Departments = databaseContext.Departments.OrderBy(m => m.Department_data).ToList();
    Departments.Insert(0, new Department() { Department_data = "Select department", Department_id = -1 });
    Locations = databaseContext.Locations.OrderBy(m => m.Location_data).ToList();
    Locations.Insert(0, new Location() { Location_data = "Select location", Location_id = -1 });
    Devices = databaseContext.Devices.OrderBy(m => m.DeviceType).ToList();
    Devices.Insert(0, new Device() { DeviceType = "Select device", Device_id = -1 });
    Statuses = databaseContext.Statuses.OrderBy(m => m.Status_data).ToList();
    Statuses.Insert(0, new Status() { Status_data = "Select status", Status_id = -1 });
    OS = databaseContext.OSs.OrderBy(m => m.OS_data).ToList();
    OS.Insert(0, new OperatingSystem() { OS_data = "Select os", Os_id = -1 });
    Users = databaseContext.Users.OrderBy(m => m.NameOfUser).ToList();
    Users.Insert(0, new AlandiaUser() { NameOfUser = "Select user", Username = "Select user", ID = -1, });
}
```

Figur 8. Bild som visar prepareLists funktionen

Den andra funktionen **saveChanges()** går igenom alla värden för en specifik post och sätter alla variabler som finns i modellen till de värden som definierats från dropdownlistorna och textboxarna i vyn och sedan sparar dem till databasen, detta ses i figur 9 och figur 10.

```
public void saveChanges()
{
    EQDatabaseApp.DAL.DatabaseContext databaseContext = new DAL.DatabaseContext();
    ManualEntry entry = new ManualEntry();
    if (isActionValid())
    {
        if (SelectedUser.OperatingSystem.Os_id == -1) {
            entry.OS_ID = null;
        }
        else
        {
            entry.OS_ID = SelectedUser.OperatingSystem.Os_id;
        }
        if(SelectedUser.Department.Department_id == -1)
        {
            entry.DepartmentID = null;
        }
        else
        {
            entry.DepartmentID = SelectedUser.Department.Department_id;
        }
        if(SelectedUser.ManualEntryId == 0)
        {
            entry.CreatorOfEntry = HttpContext.Current.User.Identity.Name;
            entry.CreationDate = DateTime.Now;
        }
```

Figur 9. Bild som visar saveChanges-funktionen del 1

```
entry.StatusID = SelectedUser.Status.Status_id;
entry.OutOfOrder = SelectedUser.OutOfOrder;
entry.LastService = SelectedUser.LastService;
entry.RAM_GB = SelectedUser.RAM_GB;
entry.StorageCapacity_GB = SelectedUser.StorageCapacity_GB;
entry.Manufacturer = SelectedUser.Manufacturer;
entry.Other = SelectedUser.Other;

if(entry.ManualEntryId == 0)
{
    databaseContext.manualEntries.Add(entry);
}
databaseContext.SaveChanges();
```

Figur 10. Bild som visar saveChanges-funktionen del 2

Den tredje funktionen **flagAsDelete()** sparar en status (1 eller 0) till databasen. Status 1 betyder att den blivit flaggad för att tas bort och kommer att döljas i gränssnittet, vilket är för att beställlaren inte vill ta bort data permanent, men kunna dölja det i visning. För att möjliggöra detta, implementerades denna funktion, som kan ses i figur 11.

```
public void flagAsDelete()
{
    EQDatabaseApp.DAL.DatabaseContext databaseContext = new DAL.DatabaseContext();
    ManualEntry entry = new ManualEntry();
    entry = databaseContext.manualEntries.Where(obj => obj.ManualEntryId == SelectedUser.ManualEntryId).ToList().FirstOrDefault();
    entry.StatusDelete = 1;
    databaseContext.SaveChanges();
}
```

Figur 11. Bild som visar flagAsDelete-funktionen

Den fjärde funktionen **editEntry()** som kan ses i figur 12, används för att i kombination med söksidan välja en “entry” som användaren vill editera.Varje “entry” har en länk, vilken skickar användaren tillbaka till startsidan med entry-id för denna “entry”. Sedan sätter jag alla värden i dropdownlistorna och alla textboxarna till värden för det specifika id.

```
public void EditEntry(int id)
{
    EQDatabaseApp.DAL.DatabaseContext databaseContext = new DAL.DatabaseContext();
    ManualEntry entry = new ManualEntry();

    SelectedUser = databaseContext.manualEntries.Where(m => m.ManualEntryId == id).FirstOrDefault();
    SelectedUser.Department = databaseContext.Departments.Where(m => m.Department_id == SelectedUser.DepartmentID).FirstOrDefault();
    SelectedUser.Device = databaseContext.Devices.Where(m => m.Device_id == SelectedUser.DeviceID).FirstOrDefault();
    SelectedUser.Location = databaseContext.Locations.Where(m => m.Location_id == SelectedUser.LocationID).FirstOrDefault();
    SelectedUser.Status = databaseContext.Statuses.Where(m => m.Status_id == SelectedUser.StatusID).FirstOrDefault();
    SelectedUser.OperatingSystem = databaseContext.OSs.Where(m => m.Os_id == SelectedUser.OS_ID).FirstOrDefault();
    SelectedUser.AlandiaUser = databaseContext.Users.Where(m => m.ID == SelectedUser.UserID).FirstOrDefault();

    if(SelectedUser.OperatingSystem == null)
    {
        SelectedUser.OperatingSystem = new OperatingSystem() { Os_id = -1 };
    }
}
```

Figur 12. Bild som visar editEntry-funktionen

Den femte funktionen **defaultSelectedUser()** skapar en default-användare genom att definiera värden. Funktionen anropas varje gång som användaren lämnar startsidan så att alla värden blir återställda. Om man inte gjorde det så uppstod en bugg när man försökte editera ett “entry”. Min controller har en if-sats som frågar om ID har ett värde och om den inte har värde så anropar jag **defaultSelectedUser()**. Bild på funktionen kan man se i figur 13.

```
public void DefaultSelectedUser()
{
    SelectedUser.ManualEntryId = 0;
    SelectedUser.AlandiaUser = new AlandiaUser() { ID = -1 };
    SelectedUser.AlandiaUser.Username = "";
    SelectedUser.Device = new Device() { Device_id = -1 };
    SelectedUser.DeviceName = "";
    SelectedUser.Model_data = "";
    SelectedUser.SerialNumber = "";
    SelectedUser.MacAddress = "";
    SelectedUser.IpAddress = "";
    SelectedUser.PutInUse = null;
    SelectedUser.Location = new Location() { Location_id = -1 };
    SelectedUser.Department = new Department() { Department_id = -1 };
    SelectedUser.Status = new Status() { Status_id = -1 };
    SelectedUser.OutOfOrder = null;
    SelectedUser.LastService = null;
    SelectedUser.OperatingSystem = new OperatingSystem() { Os_id = -1};
    SelectedUser.RAM_GB = "";
    SelectedUser.StorageCapacity_GB = "";
    SelectedUser.Manufacturer = "";
    SelectedUser.Other = "";
    SelectedUser.StatusDelete = 0;
    SelectedUser.CreatorOfEntry = "";
    SelectedUser.UpdaterOfEntry = "";
    SelectedUser.CreationDate = DateTime.Now;
}
```

Figur 13. Bild som visar defaultSelectedUser-funktionen

Den sjätte funktionen **isActionVaild()** kontrollerar om vissa värden är ifyllda, vilket dessa värden måste vara för att få spara en post till databasen. Bild på denna funktion kan ses i figur 14.

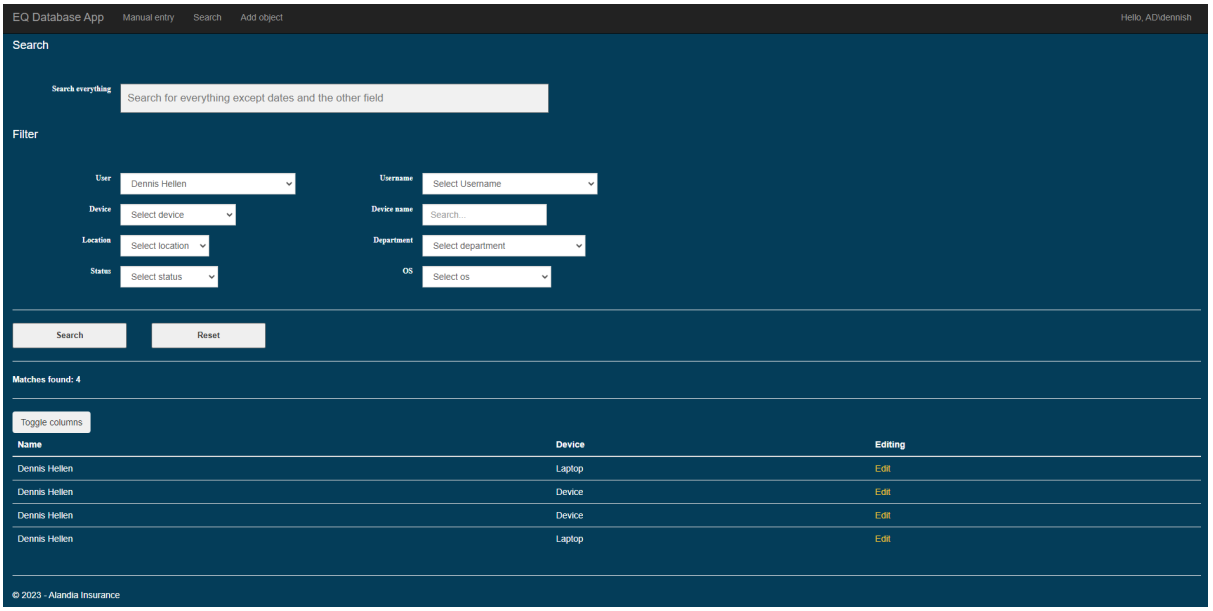
```
public bool isActionValid()
{
    if (SelectedUser.Device.Device_id == -1)
    {
        lblInfoMessage += "Please Select a device. ";
    }
    if (SelectedUser.Location.Location_id == -1)
    {
        lblInfoMessage += "Please Select a location. ";
    }
    if (SelectedUser.Status.Status_id == -1)
    {
        lblInfoMessage += "Please Select a status. ";
    }
    if(SelectedUser.AlandiaUser.ID == -1)
    {
        lblInfoMessage += "Please Select a User. ";
    }

    return true;
}
```

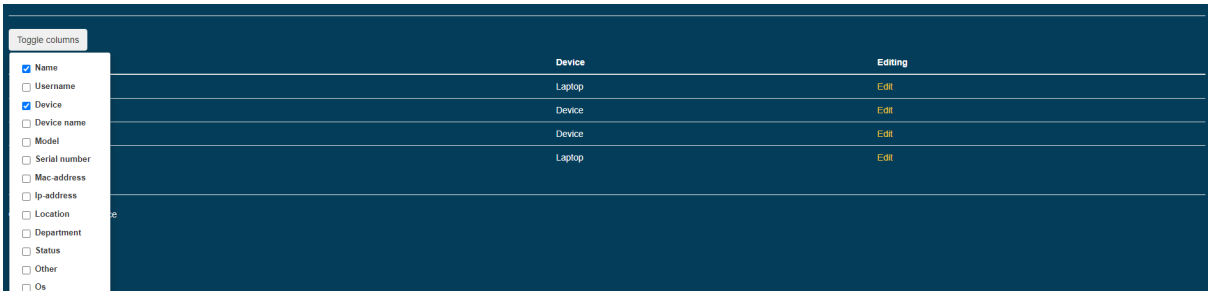
Figur 14. Bild som visar isActionValid-funktionen

3.7 Söksidan

Andra sidan som skapades används för att söka i databasen och visa datan man sökt efter, det finns även ett filtreringssystem och man kan välja vilka kolumner man vill se i resultatet. I resultatet finns även en kolumn med länken för att återgå till första sidan för editering av just det “entry” man valde. Den här sidan har bara 3 funktioner och bilden på den färdiga produkten kan ses i figur 15 och 16.



Figur 15. Bild som visar hur search sidan ser ut



Figur 16. Bild som visar olika kolumner som man kan visa för en sökning

Som på startsidan så har söksidan en **prepareLists()** där man initialiserar listorna med värden från databastabellerna. Funktionen **getFiltermodel()** hämtar de värden från vyn som inte är null från filteringsdelen av vyn.

Den sista funktionen **searchFilter()** som utför sökningen. Den är uppbyggd i två steg, först går man igenom allt från den databastabellen som har all data för entries, och sedan går man igenom filtreringsalternativen en efter en tills man har fått fram den sökning man önskar. Sedan sätter man de värden som stämmer överens med sökningen till rätt html-element. Bild på funktionen visas i figur 17,18 och 19.

```
public void searchFilter(FilterModel filterModel)
{
    EQDatabaseApp.DAL.DatabaseContext databaseContext = new DAL.DatabaseContext();
    bool doSearch = false;
    IQueryable<ManualEntry> query = databaseContext.manualEntries;

    if (filterModel.SearchEverything != null)
    {
        doSearch = true;

        query = query.Where(obj => obj.AlandiaUser.NameOfUser.Contains(filterModel.SearchEverythin
        || obj.DeviceName.Contains(filterModel.SearchEverything) || obj.Model_data.Contains(filter
        || obj.Department.Department_data.Contains(filterModel.SearchEverything) || obj.Location.L
        || obj.StorageCapacity_GB.Contains(filterModel.SearchEverything) || obj.OperatingSystem.OS
        || obj.UpdaterOfEntry.Contains(filterModel.SearchEverything));
    }
```

Figur 17. Bild som visar searchFilter-funktionen del 1

```
if (filterModel.SelectedDeviceID != -1)
{
    doSearch = true;
    query = query.Where(obj => obj.Device.Device_id == filterModel.SelectedDeviceID);
}
if (filterModel.SearchDevicename != null)
{
    doSearch = true;
    query = query.Where(obj => obj.DeviceName.Contains(filterModel.SearchDevicename));
}
if (filterModel.SelectedDepartmentID != -1)
{
    doSearch = true;
    query = query.Where(obj => obj.Department.Department_id == filterModel.SelectedDepartmentID);
}
if (filterModel.SelectedLocationID != -1)
{
    doSearch = true;
    query = query.Where(obj => obj.Location.Location_id == filterModel.SelectedLocationID);
}
```

Figur 18. Bild som visar searchFilter-funktionen del 2

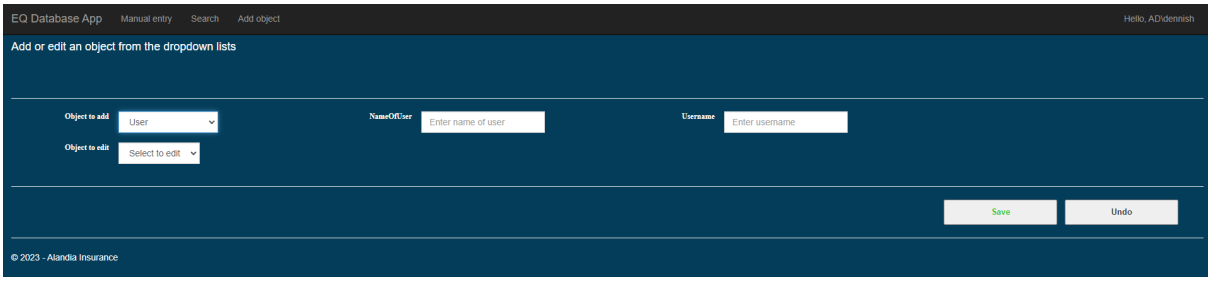
```
    }
    if (doSearch)
    {
        query = query.Where(obj => obj.StatusDelete != 1);
        manualEntriesResult = query.ToList();
        matchesCount = manualEntriesResult.ToList().Count();

    }
```

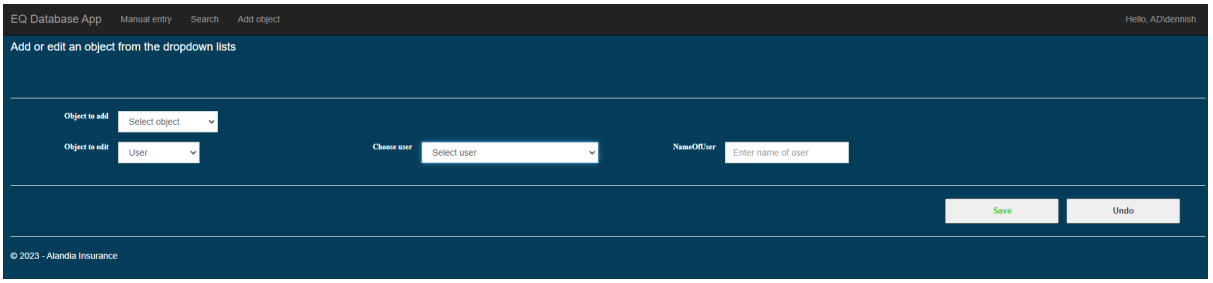
Figur 19. Bild som visar searchFilter-funktionen del 3

3.8 Administratörssidan

Tredje sidan som skapades är till för administration av vissa saker i applikationen. Bild på den färdiga produkten visas i figur 20 och 21. Detta innefattar uppdatering av dropdownlistors värden, till exempel lägga till en ny avdelning eller plats och lägga till och uppdatera användares användarnamn och namn. Det finns 4 funktioner för den här sidan.



Figur 20. Bild som visar add user-delen av adminsidan



Figur 21. Bild som visar update user-delen av adminsidan

Den första är igen **prepareList()** som gör samma sak som på de andra sidorna. Följande funktion är **saveChanges()** som fungerar på samma sätt som de startsidans funktion men eftersom att man måste kolla om det exakta värdet redan existerar i databastabellerna så är den lite annorlunda. Bild på funktionen ses i figur 22 och 23.

```
if (isActionValid())
{
    if(tablevalidation() != null)
    {
        string result = tablevalidation();
        return result;
    }
    if(SelectedObject.ID != -1)
    {
        if (SelectedObject.ID == 2)
        {
            devEntry.DeviceType = SelectedDevice.DeviceType;
        }
        if (SelectedObject.ID == 3)
        {
            locEntry.Location_data = SelectedLocation.Location_data;
        }
```

Figur 22. Bild som visar saveChanges för adminvyn del 1

```
if (userEntry.NameOfUser != null && SelectedObject.ID != -1)
{
    databaseContext.Users.Add(userEntry);
}
if(userEntry.Username != null && SelectedObject.ID != -1)
{
    databaseContext.Users.Add(userEntry);

}
try
{
    databaseContext.SaveChanges();
    return "Object has been saved";
}
catch(DbUpdateException ex)
{

    string location = "Location already exists.";
    string department = "Department already exists.";
    string device = "Device already exists.";
    string operatingSystem = "OS already exists.";
    string user = "User and/or username already exists.";
    string genericError = "Error! something went wrong.";


    if (ex.InnerException.InnerException.Message.Contains("AK_DeviceType"))
    {
        return device;
    }
    else if (ex.InnerException.InnerException.Message.Contains("AK_Department_data"))
    {
        return department;
    }
```

Figur 23. Bild som visar saveChanges för adminvyn del 2

Den tredje funktionen **tablevalidation()**, som kan ses i figur 24, är en funktion som tittar efter om värdet som försöker sparas till databasen redan existerar i tabellerna. Om de gör det skickar funktionen en sträng med ett felmeddelande som säger vad som har hänt.

```
public string tablevalidation()
{
    string location = "Location already exists.";
    string department = "Department already exists.";
    string device = "Device already exists.";
    string operatingSystem = "OS already exists.";
    string user = "User and username already exists.";
    string username = "Username already exists.";
    string editUsername = "Username is the same.";

    if(SelectedObject.ObjectType == "Location")
    {
        for (int i = 0; i < Locations.Count; i++)
        {
            if (Locations[i].Location_data.ToLower() == SelectedLocation.Location_data.ToLower())
            {
                return location;
            }
        }
    }
    if(SelectedObject.ObjectType == "Department")
    {
        for (int i = 0; i < Departments.Count; i++)
        {
            if (Departments[i].Department_data.ToLower() == SelectedDepartment.Department_data.ToLower())
            {
                return department;
            }
        }
    }
```

Figur 24. Bild som visar tableValidation-funktionen

Sist har vi igen **isActionValid()** som de andra sidorna validerar det som man försöker spara ner till databasen.

3.9 De olika vyerna

Vyerna är alla uppbyggda med HTML som kodspråk, alla använder sig dock av lite asp.net kod för att få värden från modellerna att visas i vyerna. Detta gör man genom att lägga till en kodrad i toppen av vyn som bestämmer vilken modell som används, till exempel **@model projektnamn.mappnamn.modellnamn**. Exempel på sådana funktioner är **Html.Label()/Html.LabelFor()**, **Html.TextBox()/Html.TextBoxFor()** och **Html.DropDownListFor()**, som man kan se exempel på i figur 25 och 26. Alla av funktionerna som har For i sitt namn fungerar som en where-sats där man med hjälp av ID kan få värdens namn för att visa i vyn.

```
<div class="col-lg-3">
    <div class="form-group">
        @Html.LabelFor(m => m.SelectedUser.AlandiaUser.Username, "Username", new { @class = "infoLabel" })
        <div class="input-group">
            @Html.TextBoxFor(m => m.SelectedUser.AlandiaUser.Username,
            new { @class = "form-control", @placeholder = "Username", @type = "text", @disabled="disabled" })
        </div>
    </div>
</div>
```

Figur 25. Bild som visar labelFor och textBoxFor-funktionaliteten

```
<div class="col-lg-3">
    <div class="form-group">
        @Html.LabelFor(m => m.SelectedUser.Location.Location_id, "Location", new { @class = "infoLabel" })
        <div class="input-group">
            @Html.DropDownListFor(m => m.SelectedUser.Location.Location_id, new SelectList(Model.Locations,
            "Location_id", "Location_data", Model.SelectedUser.Location.Location_id), new { @class = "form-control" })
        </div>
    </div>
</div>
```

Figur 26. Bild som visar dropdownlistFor-funktionaliteten

I den startvyn så skapar eller uppdaterar man en enhetskoppling. Man skapar kopplingen genom att fylla i värden för enhetskopplingen och sedan sparar ner den till databasen. Man uppdaterar en enhetskoppling genom att från länken som man får i sökresultatet i sökvyn bli skickad tillbaka från sökvyn till startvyn, där man med hjälp av id för kopplingen får alla värden för en enhetskoppling i rätt HTML-element och man kan sedan uppdatera dem och spara ner till databasen. Bild på resulterande vy ses i figur 7.

I sökvyn kan man söka på allt eller med filtrering och så visas det i en tabell som har en dropdown med checkboxar som döljer eller visar kolumner som man vill se eller inte se. Sen

har jag en partiell vy där sökresultaten visas som anropas i sökvyn. Bild på hur man navigerar till den partiella vyn syns i figur 27. Figur 28 visar hur den partiella vyn ser ut. Bilden på resultatet ses i figurerna 15 och 16.

```
<tbody id="searchDataBody">
    @Html.Partial("_SearchPartial")
</tbody>
```

Figur 27. Bild som visar hur man navigerar till en partiell vy

```
@model EQDatabaseApp.Models.SearchModel
@foreach (var item in Model.manualEntriesResult)
{
    <tr>
        <td class="ResultName" style="display:none">@item.AlandiaUser.NameOfUser</td>
        <td class="ResultUsername" style="display:none">@item.AlandiaUser.Username</td>
        <td class="ResultDevice" style="display:none">@item.Device.DeviceType</td>
        <td class="ResulDeviceName" style="display:none">@item.DeviceName</td>
        <td class="ResultModel" style="display:none">@item.Model_data</td>
        <td class="ResultSerial" style="display:none">@item.SerialNumber</td>
        <td class="ResultMac" style="display:none">@item.MacAddress</td>
        <td class="ResultIp" style="display:none">@item.IpAddress</td>
        <td class="ResultLocation" style="display:none">@item.Location.Location_data</td>
        <td class="ResultDepartment" style="display:none">@item.Department.Department_data</td>
        <td class="ResultStatus" style="display:none">@item.Status.Status_data</td>
        <td class="ResultOther" style="display:none">@item.Other</td>
        <td class="ResultOs" style="display:none">@item.OperatingSystem.OS_data</td>
        <td class="ResultRAM" style="display:none">@item.RAM_GB</td>
        <td class="ResultStorage" style="display:none">@item.StorageCapacity_GB</td>
        <td class="ResultManufacturer" style="display:none">@item.Manufacturer</td>
        <td class="ResultPutInUse" style="display:none">
            @if (item.PutInUse.HasValue)
            {
                @item.PutInUse.Value.ToShortDateString();
            }
            else
            {
                @item.PutInUse;
            }
        </td>
```

Figur 28. Bild som visar del av den partiella vyn

Den sista vyn är för adminsidan, där har jag dropdownlistor där man väljer om man ska uppdatera eller skapa ny användare, plats, avdelning, operativsystem eller enhetstyp. Beroende på vad man väljer i dem så visas tidigare dolda dropdownlistor för att välja specifikt vilket värde man vill uppdatera eller om man vill skapa ett nytt värde i listorna. Bild på resultatet ses i figurerna 20 och 21.

4 RESULTAT

4.1 Applikationen i sin helhet

I sin helhet fungerar applikationen ypperligt, inga långa laddningstider, all funktionalitet som uppdragsgivaren ville ha finns, inklusive att adminsidan som var ett väldigt sent påfund i utvecklingen som inte ett direkt krav, utan mera som en “nice to have”-funktionalitet som jag lade till mot slutet av projektet. Applikationen har därmed tre sidor som har funktionalitet för att lägga till och uppdatera poster, söka i databasen efter poster, och administrera genom att kunna lägga till användare och dylikt i dropdownlistorna. Applikationen följer Alandias säkerhetsåtgärder , dessutom ligger den under Alandias interna system så att ingen utomstående kan komma åt den. Databasen fungerar bra och all data kommer att sparas, även om något post skulle bli flaggat för borttagning så döljs de i gränssnittet men på uppdragsgivarens begäran kommer de att finnas kvar i databasen. Under det halvår som appen har legat i produktion uppstod inga buggar utöver att applikationen hade pekat mot testdatabasen istället för produktionsdatabasen, vilket blev korrigerat när det upptäcktes innan projektet var avslutat.

4.2 Levererad nytta till användare

Om man lyssnar på användarna är det helt klart att applikationen har förbättrat deras arbete med den här datan. Enligt användare A har datakvaliteten blivit bättre (användare A):

> Vi har minimerat fel från Excel-filen med appen eftersom vi har ett väldigt stramt ramverk nu då i den här nya appen gentemot Excel där det kan, ja… är det den senaste versionen? Eller är det någon som är in och ändrar någonting? Och så vet vi alla att Excel kan bugga lite ibland och sen är det just sökfunktionen där som är mycket starkare i appen än i ett Excelark och du kan lite lättare söka där.

användare A anser även att användarvänligheten har blivit mycket bättre(användare A):

> Det har blivit mycket bättre, i Excel är det ju kolumner som är för små eller så är det filter som sitter kvar, men här utgår det ju alltid från samma vy i appen. För det första är den lite mörkare, det gillar vi också, rent ögonmässigt och sen då att man "keep it simple" egentligen, du har inte alltför mycket fält att fylla i, vi är ändå begränsade och vi har tagit fram det viktigaste. Det är positivt, absolut

Som slutkommentar berättar användare A att det är mycket lättare att söka fram den data man är ute efter: "man kan dra ut vem som har en iPhone 8 nu då, lite bättre, att inte alla har stavat olika eller alla har gjort ett dubbelt mellanslag istället för en Excel då.".

Enligt användare B har det nog blivit bättre men det fanns kanske vissa fördelar med Excel (användare B):

> Ja, jag tycker nog att det är bättre så här, det är det absolut. Det som kanske var en fördel med Excel ändå, om man tänker sig att datat ska ha varit perfekt där och det ska ha varit perfekt, alltså man ska verkligen kunna litat på datat, så kan man lätt filtrera och sortera på kolumner på vad som helst,i en Excel-kalkyl, har man ju den fördelen så länge den är liten, en stor datamängd på det sättet kan man inte hantera i alla fall, men i det här fallet, just med den här, så kan man det, och då kanske man kunde lätt låta sig ta ut bara ett papper på… Ja, det här är det här segmentet, IP-adresser, och så tog man det ner i källaren när man satte sig ner och började konfigurera en ny hårdvara, en host i VMware, exempelvis, men i appen eller applikationen som du har byggt så får man ju... Alltså det är ju bara också att söka på någonting, fritextsöka till exempel på 10.18. och så antar jag att den hittar allting med de IP-adressegmenten och alla adresserna som kommer upp. Då får jag listan där och så kan jag göra en väldigt enkelt notering på en gul lapp, eller ta ett foto till mobilen och så tar jag det med mig i källaren om vi tar samma exempel eller samma behov så funkar det ju alldeles utmärkt också.

Som slutsats baserat på användarintervjuerna har deras jobb underlättats och risken för felaktig data har minimerats, vilket betyder att applikationen är en välkommen uppgradering från det gamla systemet.

4.3 Levererad nytta till organisationen

Jag frågade CISO (chief information security officer) angående säkerheten och hur applikationen har förbättrat säkerheten för datan som det handlar om. Enligt CISO har säkerheten förbättrats i och med dessa förändringar för hur man hanterar datan från applikationen (CISO).

> Då tänker jag väl så här då att per automatik så har vi ju… alltså själva datat i sig ligger ju på en SQL-databas då och den har vi mer processmässigt bättre koll på att den backupas än en Excel-fil som ligger på en shared folder. Här är en Excel-fil som kanske ligger på en shared folder, den backupas också, men det följs inte upp exakt på lika strikt sätt utifrån våra rutiner på infrastruktursidan här och dagliga checklistor, så det är ju bättre uppföljning med en databas. Sen det andra så är ju att en Excel som någon gör en ändring i... Det var ju det som hände att det blev helt ohållbart att veta vad som har ändrats, att riktigheten i datat... Det blev inte bra, alltså, för att det är lätt att ändra någonting där, i misstag, eller ta bort en rad i misstag, eller liknande, i en Excel-kalkyl jämfört med en i en riktig applikation som skriver och läser från en databas. Vad mer att tänka då om säkerheter? Ja, så blir ju accesserna till det blir ju bättre, loggningen blir bättre, ja det är generellt mycket bättre.

Som en slutkommentar säger han: “Men det är helt klart, det är så här det behöver vara, det här med att seriöst försöka hålla reda på någonting i Excel så tror jag inte på i längden för då ska man definitivt inte vara mer än en person som har skrivrättigheter till filen, sen kan det gå snett ganska snabbt med datakvaliteten där.” Som slutsats har säkerheten ökat och riktigheten av datan har blivit mycket bättre, vilket har lett till förbättringar i förvaringen och användningen av datan om man jämför applikationen med Excel-dokumentet.

5 SLUTSATSER

Min slutsats från detta projektarbete är att Entity Framework är ett mycket användbart ramverk för att skapa MVC webbapplikationer. Jag började på det här projektet under min första praktik och då var jag relativt oerfaren inom C# och helt ny till Entity Framework och ASP.NET, men C# är ett mycket behändigt språk som inte tar länge alls att förstå om man har grunder i andra programmeringsspråk. Eftersom C# är en Microsoft-produkt är det mycket smidigt att använda sig av deras SQL-databasprogram och kopplingen mellan databas och backend blir därför väldigt enkel.

Enligt mig och min uppdragsgivare finns det inget behov av mer funktionalitet just nu, jag implementerade alla krav som de ställde. Jag började projektet och hade nästan gjort klart det innan jag bestämde att jag skulle använda det som mitt slutarbete, men jag kom fram till att det skulle vara ett mycket intressant ämne att göra slutarbete på.

Jag kommer garanterat att jobba med C# i framtiden eftersom det har blivit mitt mest använda språk samt det språk jag är mest kunnig inom. Jag skulle rekommendera att de som har tidigare kunskap i programmering eller är nybörjare inom programmering att lära sig C#. Det är ett av de största språken i världen och det får kontinuerliga uppdateringar. Det finns många olika möjligheter med C# och Entity Framework är bara en liten del av det. Microsoft SQL Server Management är också ett verktyg som jag rekommenderar, det är det bästa databasprogram som jag har använt.

Under detta projekt har jag utvecklats som programmerare och känner mig nu mycket säker och bekväm med att skriva C#-kod. Jag är mycket nöjd med min insats i detta projekt och de erfarenheter som jag fått med mig gällande C# och utveckling av C#-projekt kommer jag att ta med mig under hela min karriär. Jag hoppas att ni som läsare har lärt er något och främst fått ett intresse för att programmera i C# och allt som det erbjuder.

KÄLLFÖRTECKNING

Chmiel, M. (2022). *JavaScript*. Salem Press Encyclopedia.

Clark J.A. (2006). AJAX (asynchronous javascript and XML): this isn't the web I'm used to. *Online*, *30*(6), 31–34.

Gaylord, J. N., Hanselman, S., Miranda, T., Rastogi, P., & Wenz, C. (2013). *Professional ASP.NET 4.5 in C♯ and VB*. John Wiley & Sons, Incorporated.

Haverbeke, M. (2014). *Eloquent JavaScript : A Modern Introduction to Programming*. No Starch Press, Incorporated.

Hong, P. (2018). *Practical Web Design : Learn the Fundamentals of Web Design with HTML5, CSS3, Bootstrap, JQuery, and Vue. js*. Packt Publishing, Limited.

Kanjilal, J. (2015). *Entity framework tutorial : A comprehensive guide to the entity framework with insight into its latest features and optimizations for responsive data access in your projects*. Packt Publishing, Limited.

Kte'pi, B. M. (2023). *Cascading Style Sheets (CSS)*. Salem Press Encyclopedia.

Microsoft ASP.NET. (u.å.). *What is ASP.NET?*. https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet.

Microsoft C#. (4 maj 2023). *A tour of C# - overview - C#.* A tour of C# - Overview - C# | Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.

Microsoft MVC. (27 juni 2022). *Overview of ASP.NET core MVC*. Microsoft Learn. https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-7.0&WT.mc_id=dotnet-35129-website.

Microsoft Visual Studio. (u.å.) *Visual studio 2022: Download for free*, *Visual Studio*. https://visualstudio.microsoft.com/vs/.

Microsoft SSMS. (10 augusti 2023). *Download SQL server management studio (SSMS) - SQL server management studio (SSMS). SQL Server Management Studio (SSMS) | Microsoft Learn.* https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16.

Microsoft .NET. (u.å.). *What is .net? an open-source developer platform.* Microsoft. https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet

Peres, R. (2016). *Entity Framework core cookbook - second edition : Leverage the full potential of Entity Framework with this collection of powerful and easy-to-follow recipes*. Packt Publishing, Limited.

Thornton, J., & Otto, M. (u.å.). *Build fast, responsive sites with Bootstrap*. https://getbootstrap.com/

Turtschi, A., Werry, J., Hack, G., & Albahari, J. (2002). *C#.Net Developer's Guide*. Elsevier Science & Technology Books.

US Government. (2023). *VA technical reference model V 23.8. WebGrease.* https://www.oit.va.gov/Services/TRM/ToolPage.aspx?tid=10820#:~:text=General%20Information&text=WebGrease%20is%20a%20set%20of,CSS)%20in%20a%20web%20application.

West, J. (2006). Ajax: Not Just Another Acronym or Is It? *Searcher. 14*(1), 13–15.

Whitehead, P., & Russell, J. H. (2005). *HTML : Your Visual Blueprint for Designing Web Pages with HTML, CSS, and XHTML*. John Wiley & Sons, Incorporated.

Winter, M. (2023). *HTML*. Salem Press Encyclopedia of Science.

York, R. (2015). *Web development with jQuery*. John Wiley & Sons, Incorporated.

BILAGOR

Bilaga 1