

CHALMERS



GÖTEBORGS UNIVERSITET

Digitalisering av fastighets- och kundregister

Migrering och hantering av data från Excel till MySQL-databas

Examensarbete inom Data- och Informationsteknik

Andin Blomgren

Institutionen för Data- och Informationsteknik

CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2026
www.chalmers.se

EXAMENSARBETE 2026

Digitalisering av fastighets- och kundregister

Migrering och hantering av data från Excel till MySQL-databas

Andin Blomgren



GÖTEBORGS
UNIVERSITET



CHALMERS

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2026

Digitalisering av fastighets- och kundregister
Migrering och hantering av data från Excel till MySQL-databas
ANDIN BLOMGREN

© ANDIN BLOMGREN, 2026.

Uppdragsgivare: X2 Wireless
Handledare på Chalmers Tekniska Högskola: Ulf Norell
Teknisk handledare på X2 Wireless: Gustaf Järgren
Examinator: Jonas Duregård, CS, CSE Examensarbete 2024

Institutionen för Data- och Informationsteknik
Chalmers tekniska högskola
Göteborgs universitet
SE-412 96 Göteborg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Göteborg, Sverige 2026

Digitalisation of property and customer register
Institution of Computer and Information technology
Chalmers University of Technology
Gothenburg University

Abstract

This bachelor's thesis focuses on migrating property and customer information from Excel files to an existing MySQL database at X2 Wireless. The company uses Excel files to store information about properties and customers, but the files often have different structures. This makes it difficult to collect and manage the information in a consistent way. The purpose of the project was to develop a tool that could transfer the information into a common database structure without losing relevant data.
The tool was developed as part of the company's existing administrative Windows application. Before importing the data, the Excel files were converted to CSV format. The user could then select which columns should be connected to the predefined fields in the database and preview the information before saving it. Any remaining information was stored as JSON metadata. The existing application was also extended to allow users to view the imported objects, change sensor statuses and add comments.
The migration tool was tested using thirteen anonymized Excel files containing data from the company's operations. The files had different structures, but all of them could be imported and processed without making changes to the tool for each file. Manual checks of the original information, the preview and the values stored in the database did not reveal any differences in the tested cases.
The results show that the main functional goals of the project were achieved. However, the testing was limited to the available files, and no measurements were made to determine how much time or administrative work the solution could save.

Digitalisering av fastighets- och kundregister
Institutionen för Data- och Informationsteknik
Chalmers tekniska högskola
Göteborgs universitet

Sammanfattning

Detta examensarbete handlar om att migrera fastighets- och kundinformation från Excel-filer till en befintlig MySQL-databas hos X2 Wireless. Företaget använder Excel-filer för att lagra information om fastigheter och kunder, men filerna har ofta olika struktur. Detta gör det svårt att samla och hantera informationen på ett enhetligt sätt. Syftet med projektet var därför att utveckla ett verktyg som kunde föra över informationen till en gemensam databasstruktur utan att relevant information gick förlorad.
Verktyget utvecklades som en del av företagets befintliga administrativa Windows-applikation. Innan importen konverterades Excel-filerna till CSV-format. Användaren kunde därefter välja vilka kolumner som skulle kopplas till databasens fördefinierade fält och förhandsgranska informationen innan den sparades. Övrig information lagrades som metadata i JSON-format. Den befintliga applikationen utökades även med funktioner för att visa de importerade objekten, ändra sensorstatus och lägga till kommentarer.
Migreringsverktyget testades med tretton anonymiserade Excel-filer som innehöll uppgifter från företagets verksamhet. Filerna hade olika struktur, men samtliga kunde importeras och behandlas utan att verktyget behövde anpassas för varje enskild fil. Vid manuella kontroller av originalinformationen, förhandsgranskningen och de lagrade värdena i databasen upptäcktes inga skillnader i de genomförda testerna. Resultatet visar att projektets huvudsakliga funktionella mål uppnåddes. Testerna omfattade däremot endast de filer som fanns tillgängliga, och någon mätning av hur mycket tid eller administrativt arbete lösningen skulle kunna spara genomfördes inte.

Förord

Jag skulle vilja tacka Andreas Jonsson på X2Wireless för att ha gett mig möjligheten att få jobba på detta projekt, samt min handledare Gustaf Järgren vid X2Wireless för att ha stöttat mig under projektet.
Jag skulle även vilja utbringa ett tack till min handledare på Chalmers, Ulf Norell för hans förslagsrika och konstruktiva möten under projektets gång.

Göteborg, September 2026

Förkortningslista

API	Application Programming Interface
CSV	Comma-separated Values
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
IDE	Integrated Development Environment
IoT	Internet of Things
JSON	JavaScript Object Notation
MVP	Minimum viable product
PDF	Portable Document Format
PHP	PHP: Hypertext Preprocessor
RAD	Rapid Application Development
SQL	Structured Query Language
VCL	Visual Component Library
XLSX	Excel Workbook-format

Innehåll

Förkortningslista **xi**

1 Inledning **1**
1.1 Bakgrund . . . 1
1.2 Problemformulering . . . 1
1.3 Syfte . . . 2
1.4 Avgränsningar . . . 2
1.5 Mål . . . 2

2 Teknisk bakgrund **4**
2.1 Excel och tabelldata . . . 4
2.2 Datastandardisering . . . 4
2.3 JSON och metadata . . . 5
2.4 MySQL . . . 5
2.5 PHP Backend och Doctrine Migrations . . . 6
2.6 C++ och Delphi/VCL . . . 6

3 Metod **7**
3.1 Planeringsfas . . . 7
3.2 Arbetssätt och arbetsfördelning . . . 7
3.3 Kravanalys . . . 8
3.4 Val av tekniska lösningar . . . 8
3.5 Valideringsmetod . . . 8

4 Genomförande **9**
4.1 Planering, målsättning och MVP . . . 9
4.2 Migrering från Excel till MySQL . . . 9
4.2.1 Ny tabell i databasen . . . 9
4.2.2 Importering av Excel-filer . . . 10
4.2.3 Hantering av datan . . . 11
4.2.4 Konvertering och tolkning av data . . . 11
4.2.5 Användargränssnitt . . . 12
4.2.6 Exportering till Databas . . . 12
4.3 Validering . . . 14

5 Resultat **15**
5.1 Verktyg för migrering av data . . . 15

5.2 Verktyg för konfiguration av databastabellen 15

6 Etik och hållbarhet **18**
6.1 Etik 18
6.2 Hållbarhet 18

7 Diskussion **19**
7.1 Planering och Tidsaspekt 19
7.2 Frontend-orienterad programmering 19
7.3 Representation av Excel-arket kodmässigt 20
7.4 Resultat och måluppfyllelse 20
7.5 Begränsningar 21

8 Slutsats **22**

Litteraturförteckning **23**

A Appendix **I**

1

Inledning

Här belyses bakgrunden, problemformulering och syfte. Kapitlet beskriver även de avgränsningar som gjorts för att fokusera på projektet på de delar som är relevanta för dess syfte.

1.1 Bakgrund

X2 Wireless är ett företag som arbetar med IoT-lösningar (Internet of Things) för att underlätta säkerheten för företag, bostadsrättsföreningar, kommunala och privata fastighetsägare.
Detta inkluderar både försäljning, installation och underhåll av de sensorer som existerar.
På senare tid så har företaget vuxit, vilket har medfört ökad arbetsbelastning för kundsupporten och installatörerna att kommunicera med varandra och kunderna. Kommunikationen mellan installatörerna sker via Microsoft Teams, medan kommunikationen med kunderna sker via Excel-ark som tillhandahålls av företaget. Informationen i dessa filer bearbetas manuellt av personalen. Då föddes idén om att digitalisera och effektivisera systemet som används för just detta.

1.2 Problemformulering

X2 Wireless hanterar idag fastighets- och kundregister genom flera Excel-ark som lagras på en gemensam filserver. Informationen bearbetas manuellt och dokumentens struktur varierar beroende på leverantör, vilket försvårar en gemensam hantering av datan. För att centralisera informationen behöver den befintliga filbaserade hanteringen ersättas med en standardiserad databasstruktur som samtidigt kan hantera variationerna i källdatan. Informationen behöver därför kunna migreras från Excel-filer till företagets befintliga MySQL-baserade system utan att relevant information går förlorad. Efter migreringen ska informationen även kunna visas och hanteras genom företagets administrativa gränssnitt.
Mot denna bakgrund behandlar projektet följande huvudfråga:
Hur kan fastighets- och kunddata med varierande struktur migreras från Excel-filer till företagets befintliga MySQL-baserade system med bibehållen information och möjlighet till efterföljande hantering.

1.3 Syfte

Syftet med examensarbetet är att utveckla och validera en lösning för migrering av fastighets- och kundregister från Excel-filer till företagets befintliga MySQL-baserade system. Lösningen ska kunna hantera variationer i källdatans struktur genom att informationen konverteras, standardiseras och lagras i en gemensam databastabell utan att relevant information går förlorad.
Arbetet omfattar även vidareutveckling av företagets befintliga administrativa applikation och tillhörande API för att möjliggöra import, förhandsgranskning och efterföljande hantering av de migrerade objekten. Lösningen ska valideras genom tester med filer av varierande struktur samt kontroll av att den importerade informationen lagras korrekt.
Genom att samla informationen i en gemensam databasstruktur förväntas lösningen kunna förenkla den administrativa hanteringen och minska behovet av manuellt arbete. Dessa potentiella effektiviseringsvinster utgör dock inte en del av projektets validering.

1.4 Avgränsningar

Det finns redan en existerande databas, Administrativt GUI och App. Så ingenting kommer att utvecklas från grunden, utan endast vidareutveckla nya funktioner och utöka användningen av de existerande resurserna. Då företagets digitala ekosystem använder sig utav MySQL, PHP, C++ med Delphi/VCL och C# med Xamarin, så kommer projektet behöva använda sig och begränsas av dessa språk.

1.5 Mål

De övergripande målen med projektet var att utveckla ett tillägg till en administrativ applikation med syfte att konvertera, importera och standardisera Excel-data, sedan exportera den till en intern databas. Kriterierna för målet utgick ifrån funktionalitet, enkelhet och möjlighet att vidareutveckla funktionerna. Utifrån detta definierades följande MVP[1]:

- **Import och standardisering:** Användaren ska kunna importera Excel-data som konverteras till CSV-format och konfigurera hur källfilens kolumner ska kopplas till databasens fördefinierade fält. Övrig relevant information ska kunna bevaras som metadata i databastabellen.

- **Förhandsgranskning och migrering:** Användaren ska kunna förhandsgranska och kontrollera den behandlade informationen innan den exporteras till MySQL-databasen.

- **Visning av databasobjekt:** Migrerade objekt ska kunna hämtas från databasen och visas genom företagets befintliga administrativa gränssnitt.

- **Konfiguration av objekt:** Användaren ska kunna granska information om ett valt objekt samt ändra status för objektets sensorer.

- **Validering:** Migreringsverktyget ska testas med filer av varierande struktur för att kontrollera att informationen kan importeras, behandlas och lagras korrekt.

2
Teknisk bakgrund

För utformning av en applikation och migrering från Excel-data till MySQL nyttjades flera ramverk program och gränssnitt som presenteras nedan.

2.1 Excel och tabelldata

Microsoft Excel är en applikation för kalkylblad och är anpassad utifrån en mångfald av användare. Excel inkluderar funktionalitet för lagring och organisering av data i tabellformat, organiserat som rader, kolumner och celler. Rader och kolumner kan användas för att representera enskilda register eller olika egenskaper. En sån här struktur gör så att Excel-filer kan användas för att samla, organisera och utbyta strukturerad data.[2]
Ett kännetecken hos databaserade kalkylblad är att strukturen kan variera i varje fil, till skillnad från databastabeller där strukturen följer ett fördefinierat schema. Ett Excel-kalkylblad följer eller genomdriver inte nödvändigtvis en fast uppsättning av attribut eller specifikt organiserad data. Skillnader sker oftast bland namn, ordning, mängd och information hos kolumner och rader. Olika kalkylblad som innehåller en liknande uppsättning av information behöver inte nödvändigtvis ha samma struktur trots att de har näst intill identisk data.[2]
Variationen i data blir en utmaning när man försöker att migrera eller sammanställa data mellan kalkylblad och ett databassystem. Då som tidigare nämnt en databastabell behöver följa ett fördefinierat schema där varje kolumn representerar ett specifikt attribut och varje register förväntas följa den fördefinierade strukturen. Datan från kalkylbladen behöver därmed transformeras innan det kan anpassas till ett databas schema.
I denna rapport kommer in-datan bestå av Excel-kalkylblad med information i form av objekt. Kalkylbladet följer en väldigt smal standardisering och kan därmed innehålla datamängder som varierar sinsemellan. Detta resulterar i att kalkylarken innehåller data som ej är representerad i samma omfattning som databasens kolumner. För att kunna säkerställa rätt hantering av datan så har Excel en funktion för att exportera som .csv (Comma-separated values), vilket möjliggör identifiering av varje existerande kolumn i kalkylarken och användning av dessa i efterhand.

2.2 Datastandardisering

Datastandardisering är processen av att transformera data från olika strukturer eller format till en gemensam och konsekvent representativ struktur. Syftet med

standardisering är att kunna ta data från olika källor med samma användningssyfte till ett gemensamt användarsystem. Inom dataintegration, standardisering innebär transformering av värden, struktur, namngivning och formatering så att de anpassar sig efter kraven utifrån användning.[3]
När data transformeras mellan system uppstår skillnader i strukturen av datan som kan skapa integrationsproblem. Främst uppstår detta på grund av skillnad i attributers namn, datarepresentation eller olika nivåer av detalj jämfört med användarsystemet. En standardiseringsprocess kan därför användas för att grunda en definierad målstruktur för hur ursprungsdatan konfigureras.[3]
I en databaskontext så kan målstrukturen representeras med ett fördefinierat databas schema. Varje standardiserat register innehåller en definierad mängd av attribut, vilket möjliggör register från in-data som har varierande struktur. En fast målstruktur kommer inte kunna representera varje attribut från källdatan. Information som ej korresponderar till ett fördefinierat attribut kommer därmed bli avfärdad eller vara representerad av ett alternativt attribut.

2.3 JSON och metadata

JSON (JavaScript Object Notation) är ett text-baserat dataformat som används för att representera strukturerad data som använder sig av "key-value" par och arrayer. Ett JSON objekt består av en samling egenskaper ("properties"), där varje egenskap har ett namn och ett associerat värde. Värden kan bestå av strängar, nummer, booleska värden, arrayer, objekt eller null. Den hierarkiska strukturen av JSON tillåter kopplad information att grupperas inom samma objekt.[4]
Metadata refererar till den informationen som beskriver övrig kontext om ett data-objekt. I ett strukturerat datasystem, kan metadata innehålla information som är relevant för en annan identitet, men nödvändigtvis inte form en del av entitetens standardiserade attribut. JSON kan användas för att representera sådan metadata då dess struktur tillåter flertalet egenskaper att lagras inom ett enskilt värde och samtidigt behålla namnen och värdena på de individuella egenskaperna. Detta skapar ett flexibelt sätt att representera information där de tillgängliga egenskaperna varierar mellan registren.
Medan JSON lagras som en del av ett databasregister så kan objektet användas vid sidan av övriga databasfält. Statiska och vanliga attribut representeras av databas kolumner, medan övriga varierande attribut lagras i ett JSON fält. Detta kombinerar en fördefinierad databas struktur med flexibel representation av information.

2.4 MySQL

MySQL är ett relationsdatabashanteringssystem, det används till att lagra och hantera strukturerad data. Datan organiseras till tabeller som består av rader och kolumner. Kolumnerna representerar attributer med individuella register. Ett databas schema definierar strukturen av tabellerna, deras kolumner, datatyper och begränsningar.[5]
MySQL stödjer även flertalet datatyper för lagring av olika typer av information, bland annat numerisk, text och datum. Det finns även stöd för JSON datatypen och

lagring av strukturerade JSON-filer.[5]
MySQL databaser styrs av Structured Query Language (SQL), som tillhandahåller funktioner att skapa och modifiera databasstrukturer samt att infoga, hämta, uppdatera och ta bort data. SQL har även funktioner för migrering, vilket tillåter möjligheten att strukturerat ändra på databasers schema eller ändra kolumner och rader. [5]

2.5 PHP Backend och Doctrine Migrations

PHP är ett server-side programmeringsspråk som vanligtvis används för att utveckla webapplikationer och backend system. En backend är ansvarig för bearbetning och hantering av applikationens logik och kommunikation, samt samspelet med datalagring såsom databaser. PHP kod körs på servern, vilket tillåter backenden att hantera "requests" och utföra operationer som ej hanteras direkt av frontend applikationen.[6]
Doctrine är en samling av PHP bibliotek som innehåller verktyg för att jobba med databaser. Genom att definiera databas-relaterad funktionalitet i backenden, kan applikationen hantera data och databas operationer genom en strukturerad mjukvaruhinna.[7]
Doctrine Migrations är en komponent inom Doctrine för att hantera ändringar till ett databas schema. Databasändringar såsom skapa eller modifiera kolumner, definieras som migrationsfiler som tillämpas i en kontrollerad sekvens. Detta medför att databas schema ändringar kan återproduceras samt möjliggör att uppdatera en databas konsekvent över många miljöer.[7]

2.6 C++ och Delphi/VCL

C++ är ett generellt programmeringsspråk som stödjer bland annat procedurell objektorienterad och generisk programmering. C++ tillhandahåller funktionalitet för utveckling av applikationer genom bland annat klasser, funktioner, datastrukturer och databibliotek. Det kan användas för att implementera applikationslogik samt för att bearbeta och manipulera data.[8]
Delphi är ett programmeringsspråk och en utvecklingsmiljö baserad på Object Pascal. Det tillhandahåller verktyg för utveckling av skrivbordsapplikationer och stödjer bland annat utveckling av grafiska användargränssnitt, applikationslogik samt kommunikation med externa resurser såsom filer och databaser.[9]
Visual Component Library (VCL) är ett kraftfullt ramverk för att bygga Windows-applikationer i C++. VCL har utvecklats av Borland för "integrated development environment" (IDE), C++ Builder och erbjuder en rik uppsättning visuella och icke-visuella komponenter som avsevärt förenklar processen med att utveckla "graphical user interfaces" (GUI) och hantera olika applikationsfunktioner. Med tiden har VCL utvecklats och blivit en integrerad del av modern C++-applikationsutveckling, särskilt inom ramen för "Rapid Application Development" (RAD).[10]

3
Metod

Projektets planerade tillvägagångssätt beskrivs i detta kapitel.

3.1 Planeringsfas

Som en del av planeringsfasen analyserades och planerades den struktur som skulle användas av den nya databastabellen. Fokus låg på att identifiera vilka fält som behövde finnas i tabellen samt hur informationen från de inkommande Excel-filerna skulle representeras i databasen.
Parallellt planerades hur funktionaliteten skulle integreras i den befintliga applikationen. Detta innefattade att skissa på det grafiska användargränssnittet och planera hur den nya funktionaliteten skulle presenteras för användaren. Hänsyn togs till den befintliga applikationens struktur och utformning för att den nya delen skulle kunna integreras på ett enhetligt sätt. En ytterligare del av planeringsfasen var att undersöka olika sätt att läsa in Excel-datan in i C++ och sedan på något sätt lägga upp datan internt inom C++ för att kunna iterera igenom på så lätt sätt som möjligt.
Planeringsfasen resulterade därmed i en övergripande plan för databasstrukturen, det grafiska användargränssnittet samt hantering och omvandling av Excel-datan. Detta låg sedan som grund till den fortsatta implementationen och funktionaliteten.

3.2 Arbetssätt och arbetsfördelning

Utvecklingen av applikationen bedrevs i RAD Studio, som använder sig av VCL och Delphi för att skriva, köra och testa koden. Då all tidigare utveckling och arbete inom företaget skett i denna miljö ansågs det vara den optimala lösningen.
Själva migreringen och dess arbete skedde i PHPStorm, en PHP IDE, som användes till att skriva, köra och testa Doctrine migrations. Även detta på grund av att tidigare utveckling på företaget använder sig av den.
Projektet bedrevs parallellt med att arbeta halvtid på företaget i fråga. Vilket ledde till konstant arbete i miljöerna i fråga, samt konstant möjlighet för påverkan och feedback på arbetet av handledaren. Arbetet jobbade med versionshantering i git för PHP samt TortoiseSVN för C++ koden.

3.3 Kravanalys

De funktioner och egenskaper som den nya funktionaliteten behövde uppfylla identifierades genom en kravanalys. Kraven baserades på de övergripande målen, den befintliga applikationens struktur samt ett urval av de Excel-filer som fanns tillgängliga. Fokus låg på att skapa en lösning som kunde användas för varierande Excel-filer med ett lägre tak för att kunna skapa en standardiserad struktur och lagra resultatet i den nya databastabellen. Det centrala kravet var att applikationen skulle kunna läsa in en Excel-fil som var konverterad till CSV-format, sedan kunna identifiera de kolumner och dess information som skulle exporteras vidare till databasen. Informationen skulle under denna hanteringen kunna mappas till de fördefinierade fälten `object_id`, `type`, `status` och `sensor_amount`. Övrig information som ej motsvarar de fördefinierade fälten skulle bevaras som metadata i `object_json`.

3.4 Val av tekniska lösningar

För hanteringen av Excel-filer valdes CSV-format eftersom det möjliggör en enkel och direkt inläsning av filinnehållet som text i C++. Informationen kan därefter bearbetas radvis och delas upp i kolumner med hjälp av avgränsare, utan behov av externa bibliotek för att tolka Excel-formatets interna struktur. Därför konverterades Excel-filerna från .xlsx till .csv innan de importerades i applikationen. Trots exportering genom Excel för att gå från .xlsx till .csv så formaterades filerna till att separeras med semi-kolon ; istället för ett komma ,, detta med grund i att standard lokala inställningar på windows datorer i bland annat sverige fungerar så. Detta förenklade hanteringen av tabellinformationen och möjliggjorde bearbetning av innehållet utan att behöva implementera någon separat hantering av Excel-filens interna struktur. Utöver detta valdes det även att inte räkna med eller hantera rader där 90% av raden saknade information, detta för att det i vissa fall uppstod i filerna som tillhandatogs, men ej var behov att importera. För övriga tekniska delar utgick implementationen från den befintliga strukturen i den redan existerande applikationen. Den grafiska funktionaliteten utvecklades inom den befintliga Delphi/VCL-strukturen, medan de databasrelaterade delarna anpassades till den etablerade arkitekturen med PHP, Doctrine och MySQL. Detta för att underlätta integreringen med det befintliga systemet utan att introducera ny teknisk struktur.

3.5 Valideringsmetod

För att validera den utvecklade funktionaliteten planerades tester med Excel-filer av varierande storlek och struktur. Testerna skulle omfatta hela migreringsflödet, från import av CSV-filer till lagring i MySQL-databasen, där den importerade informationen skulle jämföras med källdatan, förhandsgranskningen och de lagrade värdena. Även hanteringen av ofullständig data skulle undersökas. Den efterföljande administrativa hanteringen skulle valideras genom att kontrollera att migrerade objekt kunde hämtas, visas, redigeras och sparas tillbaka till databasen. Utöver detta planerades ett användartest där en person utanför utvecklingsarbetet skulle få prova verktyget och lämna återkoppling.

4

Genomförande

I det här kapitlet beskrivs genomförandet av projektet.

4.1 Planering, målsättning och MVP

Projektet inleddes med planeringsmöten gällande hur tabellstrukturen skulle se ut, samt vilka olika potentiella möjligheter som fanns för att hantera Excel-filer och möjliggöra hantering av respektive data som objekt i en existerande C++ applikation. En brainstorm session startades för att bestämma vilken information från Excel-filerna som var minimi kravet på önskvärd data konsekvent nog att använda som nyckelvärden i en databastabell. Faktorer som räknades in här var vilken information som skulle komma till att brukas mest under framtida arbeten i företagets verksamhet, samt vilken information som är mest konsekvent i de Excel-filerna som förväntas hanteras i framtiden.
Följande detta sattes målen genom att definiera upp modeller för funktionerna utifrån de interna applikationerna samt framtida önskade funktioner som behövs för installatörer och kundsupport. I slutet av planeringen lyckades det plockas fram avgränsningar och krav för att sätta en MVP.

4.2 Migrering från Excel till MySQL

Migreringen skedde i fyra steg där första var att ta fram en design för hur den nya databastabellen skulle se ut. Därefter behövdes Excel-filen konverteras till CSV format och läsas in i det administrativa GUI:t för hantering. Följande till detta behövde respektive kolumner i Excel-filen hanteras så att kolumnerna på första raden som agerar nyckelvärden, matchar de som sattes för den nya databastabellen, samt de övriga mappas till metadata kolumnen. Sist men inte minst behövdes den nystrukturerade datan extraheras in till den nya databastabellen.

4.2.1 Ny tabell i databasen

I detta steget så skapades en ny tabell i den redan existerande MySQL databasen med syftet att sedan användas och refereras till i de övriga tjänsterna inom företaget. Kolumnerna bestämdes utifrån datan som bedömdes vara mest önskvärd och konsekvent (se Tabell 4.2.1), där `id` blev tabellens `PRIMARY KEY`. Med de bestämda kolumnerna skapades därefter en migration i PHP (se Listing 4.1) med hjälp av

biblioteket Doctrine Migrations, vilket används inom företaget för databas justeringar och migreringar.

id	object_id	type	status	sensor_amount	object_json

Tabell 4.2.1 Nyckelvärden för databastabellen.

Listing 4.1: Migration adding the `object` table

```
public function up(Schema $schema): void
{
    $this->addSql(
        'CREATE TABLE object (
            id INT AUTO_INCREMENT NOT NULL,
            object_id VARCHAR(64) NOT NULL,
            type VARCHAR(64) NOT NULL,
            status VARCHAR(64) NOT NULL,
            sensor_amount INT NOT NULL,
            object_json TEXT,
            PRIMARY KEY (id)
        )'
    );
}
```

4.2.2 Importering av Excel-filer

För importering av data och information från en Excel-fil till den administrativa GUI applikationen i C++, användes en av Excels funktioner att konvertera filen från .xlsx till .csv. På så sätt blir varje rad representerad av en enda lång sträng text där varje kolumn separeras med ett semi-kolon (;). Vi har antagit att den första raden i varje Excel-fil som importeras inkluderar kolumntitlar, det vill säga en kort beskrivning på 2-3 ord om vad som befinner sig i respektive kolumn. Detta lämnade oss med en grund till vad vi hade att arbeta med och hur datan kom in i programmet innan hanteringen av den.
Med den förbehandlade datan på plats kunde implementationen påbörjas genom att producera en dialogruta i den existerande applikationen, det gjordes genom att skapa en `TOpenDialog` som öppnar en filväljare med fördefinierat filter på `.csv` filer men möjlighet för övriga filtyper också. När en `.csv` fil har valts så hämtas filen och itererar igenom varje rad, separerar med hjälp av `Split()` varje värde på (;), därefter med hjälp av `std::vector::push_back` lägger en `std::vector<std::string>` vilket representerar raderna, i en `std::vector` som då blir en vektor där alla rader i filen är representerade (se Listing 4.2).

Listing 4.2: CSV Import

```
std::filesystem::path file = SysToStl(dialog->FileName);
std::ifstream fileStream(file);
std::string row;

std::vector<std::vector<std::string>> values;

while(std::getline(fileStream, row)){
    auto parts = x2::utils::Split(row, ';');
    values.push_back(parts);
}

UpdateKeyList(values);
```

4.2.3 Hantering av datan

För att hantera datan skapades en `TForm` i det administrativa verktyget, i denna skapades då flertalet `TLabel` och `TComboBox` där syftet av Labels var att representera namnen på kolumnerna i databastabellen, och syftet av ComboBoxes var att innehålla respektive information för varje databaskolumn beroende på informationen som importerats. På följande sätt hanterades respektive `TComboBox`:

- `object_id` och `sensor_amount` Innehöll varje kolumntitel från den importerade filen.
- `status` Innehöll de fördefinierade värdena (`OK, Fault, Not OK`)
- `type` Hade de fördefinierade värdena (`Lägenhet, Kontor`), men även möjligheten att mata in fritext.

Den sista kolumnen i databastabellen `object_json` var också den kolumnen med flest och mest specifika värden. Till en början fördefinierades några nycklar som var användbara och som borde vara lika mellan alla objekten, det var `personal_name`, `home_address`, `phone_nr` och `email`. Till dessa så parades grafiskt `TComboBox` som innehöll kolumnerna från Excel-filen med de fördefinierade nycklarna som `TLabel`. På liknande sätt gjordes en liknande struktur fast istället för `TLabel` så skapades `TComboBox` med möjlighet för fritext på nyckelvärden.
I de fallen där alla kolumner ej blivit hanterade i tidigare steg, används en fail-safe som först kör `std::transform` och `std::replace` på alla kolumner för att få små bokstäver och ersätta alla mellanslag med (_). Sedan mappa dessa som nyckelvärden i `json_object`.

4.2.4 Konvertering och tolkning av data

En central del av datatransformationen berörde beräkningen av `sensor_amount`. Eftersom källdatan i denna kolumnen ej innehöll antalet sensorer som ett direkt

numeriskt värde behövde ett befintligt attribut i Excel-filen användas som underlag för klassificeringen. Då de förväntade värdena i kolumnen var formaterade antingen som `X rum (och kök)` eller `X kvm`, konfigurerades identifieringen av de unika värdena i den valda kolumnen genom att använda *std::stoi* för att extrahera de numeriska värdena från strängen. Följande detta jämfördes de numeriska värdena med varandra för att sortera in respektive sträng till respektive `TComboBox` tillhörande 1, 2 och 3 sensorer. På sådant sätt kunde olika representationer av samma källinformation omvandlas till ett standardiserat värde som kan användas av den nya databasmodellen.

4.2.5 Användargränssnitt

Gränssnittet för migreringsverktyget utvecklades med RAD Studio Form Designer och VCL-komponenter. Gränssnittet fungerar som en mellanhand mellan importen av CSV-filen och den efterföljande bearbetningen och exporteringen av datan. Syftet är att användaren ska kunna konfigurera hur informationen i den importerade filen ska behandlas innan den anpassas till databasens struktur.
Gränssnittet är främst uppbyggt med `TForm`, `TComboBox` och `TLabel`. Efter att en fil har importerats används kolumnrubriker och värden från den importerade datan för att fylla relevanta `TComboBox`. Användaren kan därmed göra val utifrån informationen i den ursprungliga filen utan att behöva ändra källdatan manuellt. De val som görs i gränssnittet används därefter av programmet som konfiguration för den fortsatta databehandlingen. Gränssnittet fungerar därför inte enbart som en visuell representation av datan, utan även som en koppling mellan den importerade tabellstrukturen och programmets konverteringslogik.

4.2.6 Exportering till Databas

Efter att den importerade datan har konfigurerats och konverterats delas informationen upp i de värden som ska lagras i tabellens fasta kolumner och den metadata som ska lagras i `object_json`. Uppdelningen görs innan informationen visas för användaren och därefter exporteras till databasen.
För varje importerad rad skapas ett separat JSON-objekt med hjälp av biblioteket `nlohmann::json`. Funktionen `JsonToInsert()` (se Listing 4.3) hämtar först den JSON-data som användaren har konfigurerat och kompletterar därefter objektet med de kolumner som ska bevaras som metadata, kolumnnamnen standardiseras som tidigare nämnt i 4.2.3.
De värdena som ska representeras av databasens övriga kolumner behandlas separat och samlas radvis i en `std::vector`. Resultatet av bearbetningen består därmed av två datastrukturer, en tvådimensionell `std::vector` innehållande de fasta värdena och en `std::vector<nlohmann::json>` innehållande JSON-objekten. Dessa skickas till en `TListView` förhandsvisning, där användaren kan kontrollera den färdigbehandlade informationen innan exporten genomförs.
När användaren godkänner förhandsvisningen kan informationen exporteras till MySQL-databasen. Varje objekt exporteras genom en individuell `SQL INSERT`-operation där de behandlade värdena kopplas till motsvarande kolumner i den nya databastabellen. Individuella `INSERT`-operationer valdes för att förenkla felsökning

och hantering av eventuella problem med enskilda objekt under migreringen.

Listing 4.3: Function JsonToInsert

```
nlohmann::json TDialogImportObject::JsonToInsert(int row){
    auto configuredJson = JsonValues(row);
    UpdateUnusedColumns();
    CloudMain->debugFmt("Passed box handling,
                         start json handling.");
    try{
        int i = 0;
        for(auto allCol: Columns){
            bool exists = false;
            for(auto col: UnselectedColumns){
                if(allCol == col){
                    exists = true;
                }
            }
            if(!exists){
                std::string value = KeyValues[row][i];
                auto formattedKey = allCol;
                std::transform(formattedKey.begin(),
                    formattedKey.end(),
                    formattedKey.begin(),
                    [](unsigned char c)
                {
                    return std::tolower(c);
                });
                std::replace(
                    formattedKey.begin(),
                    formattedKey.end(), ' ', '_');
                configuredJson[formattedKey] =
                    StlToSys(value);
            }
            i++;
        }
    }catch (std::exception const &e) {
        CloudMain->debugFmt("JsonToInsert error: {}", e.what());
    }
    return configuredJson;
}
```

4.3 Validering

Testmaterialet bestod av anonymiserade Excel-filer med verkliga fastighets- och kunduppgifter från företagets verksamhet. Filerna hade olika struktur och innehåll och användes för att undersöka om migreringsverktyget kunde hantera variationer i den data som förekommer i praktiken. Tre filer med 6, 14 respektive 23 kolumner användes som huvudsakliga testfall, varefter ytterligare tio filer användes för att kontrollera funktionaliteten med fler filstrukturer. Originalfilerna redovisas inte i rapporten eftersom de innehåller verksamhetsrelaterad information. Valideringen genomfördes genom att kontrollera kolumnmappningen, förhandsgranskningen och den importerade informationen i databasen.
Valideringen omfattade hela migreringsflödet från importerad CSV-data till lagrade objekt i MySQL-databasen. Importeringen kontrollerades genom att jämföra den inlästa informationen med källdatan. Därefter testades mappning och konvertering av de importerade kolumnerna samt genereringen av metadata till `object_json`. Den färdigbehandlade informationen kontrollerades även i programmets förhandsvisning innan exporten genomfördes.
Testdata innehållande ett stort antal tomma värden användes för att kontrollera hantering av ofullständig data. Vid bearbetningen räknar programmet antalet tomma värden på varje rad. Om mer än 90 procent av värdena är tomma avbryts den fortsatta behandlingen av efterföljande rader. Detta användes för att undvika att tomma delar av de importerade filerna behandlades som objekt.
Efter migreringen kontrollerades att de skapade objekten kunde hämtas från databasen och visas i det administrativa verktyget. Även konfigurationsverktyget testades genom att läsa in migrerade objekt, modifiera deras information och spara ändringarna tillbaka till databasen. På detta sätt validerades både migreringen till den nya databasstrukturen och den efterföljande hanteringen av den lagrade informationen. För att säkerställa funktionaliteten av verktyget kördes även en testrunda med en utomstående arbetare som gav återkoppling på vad som kunde ändras och förbättras.

Övrig testning som utfördes för att validera funktionerna var följande:

- Onödigt många tomma rader i Excel-filen för att kolla hur verktyget agerar.
- Inläsning och redigering av databasobjekten sker som de ska i konfigurations verktyget, samt sparas i samma format som det lästes in.
- Ingen information läcks eller tappas bort under migrering eller konfigurering av objekten.

5
Resultat

I detta projekt har två verktyg utvecklats. Det första för migrering av Excel-filer till en MySQL-databastabell, skapats och implementerats som verktyg till en redan existerande intern Windows applikation. Det andra för att vidareutveckla existerande funktioner att konfigurera, ta bort och lägga till nya objekt i denna tabellen, samt tillägg av kommentarer till migrerade objekt, i samma Windows applikation.

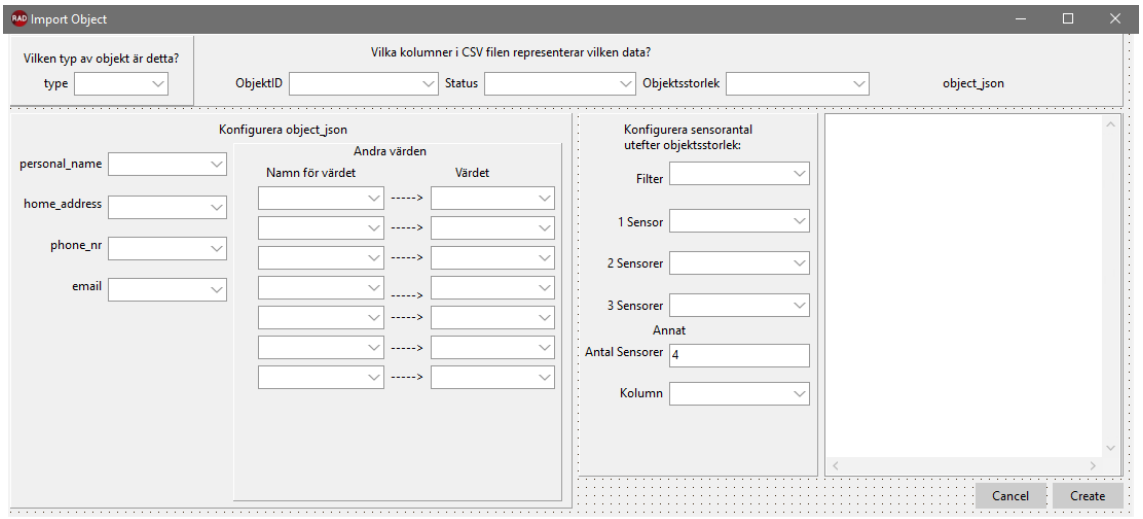
5.1 Verktyg för migrering av data

Det utvecklade migreringsverktyget gör det möjligt att läsa in en CSV-version av ett Excel-ark och konfigurera hur informationen ska överföras till den nya databastabellen. Efter import identifieras kolumnrubrikerna i filen och presenteras i gränssnittet, där användaren kan välja vilka källkolumner som ska motsvara databasens fasta fält. Övrig information kan konfigureras för lagring i `object_json`. Under konfigureringen visas även en dynamisk förhandsgranskning av JSON-informationen, vilket gör det möjligt att kontrollera resultatet innan migreringen fortsätter (se Figur 5.1). När konfigureringen godkänns visas därefter en förhandsgranskning av den färdigbehandlade datan i tabellform (se Figur 5.2). Användaren kan där kontrollera de värden som kommer att lagras innan exporten till databasen genomförs.
Vid testningen användes först tre testfiler med 6, 14 respektive 23 kolumner. Samtliga filer kunde importeras, konfigureras och exporteras till databasen utan att migreringsverktyget behövde anpassas efter respektive filstruktur. Utöver dessa genomfördes även tester med tio ytterligare filer av varierande storlek och struktur, vilka kunde behandlas genom samma migreringsflöde. Efter importen kontrollerades de migrerade filerna manuellt genom att jämföra information i originalfilerna med de värden som lagrats i databasen. Vid de genomförda testerna observerades inga avvikelser mellan originalfilerna, förhandsgranskningen och den lagrade informationen.

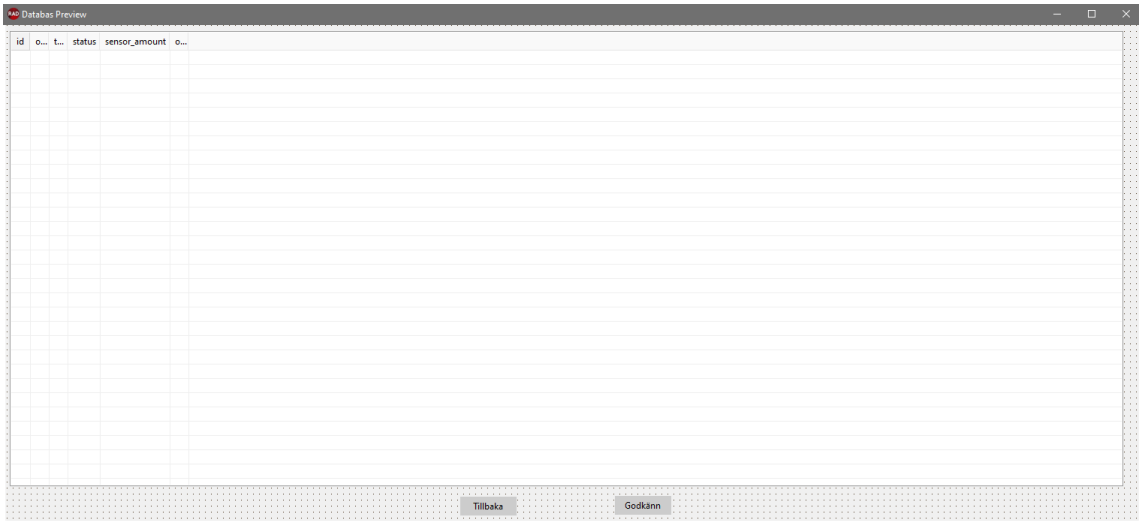
5.2 Verktyg för konfiguration av databastabellen

För att kunna hantera informationen efter migreringen kan objekten i den nya databastabellen visas i det befintliga administrativa gränssnittet. För att hantera de migrerade objekten utvecklades även en grafisk vy i det befintliga administrativa gränssnittet. När ett objekt öppnas visas dess `object_id` samt vilken typ av objekt det är. Gränssnittet innehåller även en lista över de sensorer som tillhör objektet och

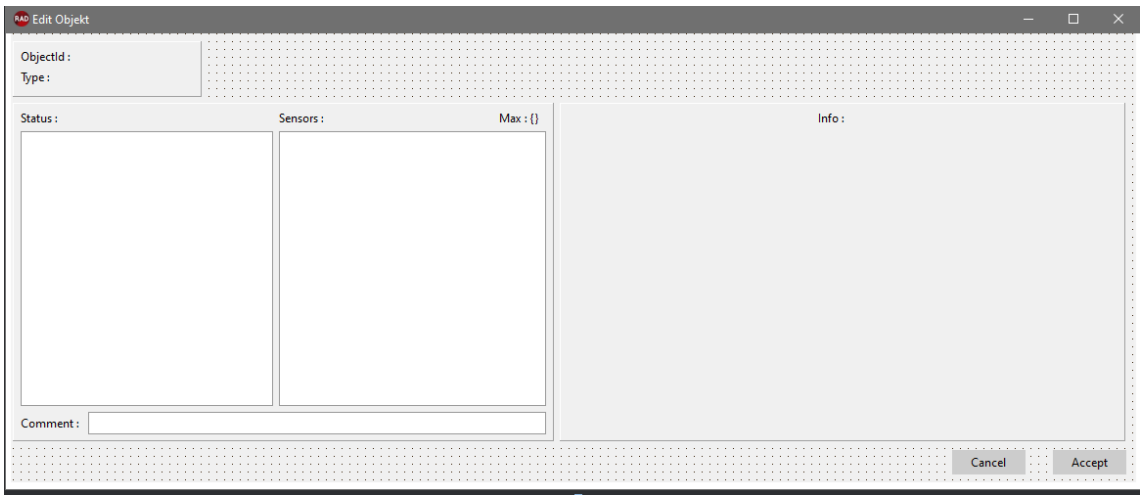
en lista över de statusvärden som kan tilldelas den valda sensorn. När användaren väljer en annan sensor uppdateras statuslistan med den valda sensorns nuvarande status markerad bland de tillgängliga statusalternativen. Användaren kan därefter ändra sensorns status. I gränssnittet visas även informationen från `object_json`. Denna information kan granskas men redigeras inte direkt genom vyn. Det färdiga konfigurationsverktyget visas i Figur 5.3.
Utöver detta finns ett kommentarsfält där användaren kan lägga till en kommentar till objektet. Kommentaren lagras tillsammans med objektets övriga metadata i `object_json`. När ändringarna godkänns sparas den uppdaterade informationen tillbaka till databasen. Verktyget implementerar inte funktionalitet för att skapa eller ta bort objekt eller sensorer. Funktionalitet för hantering av databasobjekt fanns sedan tidigare i det administrativa gränssnittet och behövde därför inte utvecklas inom projektet.



Figur 5.1: Konfigurationsfönster för Excel-ark till databastabell



Figur 5.2: Förhandsgranskning av konfigurerade värden



Figur 5.3: Konfigurationsfönster för redigering av ett objekt

6
Etik och hållbarhet

6.1 Etik

Migreringen innebär att information som tidigare lagrats i Excel-filer flyttas till en gemensam databas. Eftersom informationen kan innehålla uppgifter kopplade till kunder och fastigheter är dataskydd en viktig etisk aspekt. Informationen bör endast vara tillgänglig för behöriga användare och hanteras i enlighet med företagets befintliga rutiner för dataskydd.
Det utvecklade migreringsverktyget innehåller en förhandsgranskning innan informationen skrivs till databasen. Användaren ges därmed möjlighet att kontrollera hur informationen har tolkats och strukturerats innan migreringen genomförs. Detta minskar risken för att felaktigt konfigurerad information lagras utan att användaren först har möjlighet att upptäcka felet.
Projektet omfattade inte utveckling av nya funktioner för behörighetskontroll eller en utvärdering av systemets efterlevnad av GDPR. Dessa delar hanteras därför av företagets administrativa system och rutiner.

6.2 Hållbarhet

Papperslös verksamhet: Digitalisera register för att minska pappersavfallet. Att digitalisera hyresgästdata är ett viktigt steg mot en papperslös verksamhet, vilket kan bidra till att minska miljöpåverkan genom minskat pappersavfall och effektivare resursanvändning.

Arbete på distans: Uppmuntra distansarbete för att minska koldioxidavtrycket. Att flytta hyresgästdata till en digital databas möjliggör och uppmuntrar distansarbete, vilket kan ha betydande positiva effekter på företagets koldioxidavtryck.

7

Diskussion

7.1 Planering och Tidsaspekt

Planeringen av projektet genomfördes enligt företagets arbetssätt för nya projekt, där användningsområde, funktionskrav, avgränsningar och planerade aktiviteter placerades på en tidslinje. Under utvecklingen visade det sig dock att vissa delar av arbetet hade underskattats tidsmässigt. Detta gällde framför allt utveckling av användargränssnittet och hanteringen av de olika sätt som den importerade informationen kunde vara strukturerad på.
Projektets omfattning förändrades även under arbetets gång när ytterligare funktionalitet identifierades som nödvändig för att skapa ett användbart migreringsflöde. Detta medförde att mer utvecklingstid behövde läggas på delar som ursprungligen hade uppskattats som mindre omfattande. En tydligare prioritering mellan funktioner som krävdes för projektets MVP och funktioner som kunde lämnas till framtida vidareutveckling hade därför kunnat ge en mer träffsäker tidsplanering.

7.2 Frontend-orienterad programmering

Vid projektets början uppskattades utvecklingen av användargränssnittet som en relativt liten del av arbetet. Denna uppskattning visade sig vara för låg. Gränssnittet behövde inte enbart presentera information, utan även ge användaren möjlighet att konfigurera hur importerade kolumner skulle kopplas till databasens struktur, kontrollera resultatet genom förhandsgranskning samt senare hantera de migrerade objekten.
En bidragande faktor var att tidigare erfarenhet främst låg utanför frontend-orienterad utveckling och design av grafiska gränssnitt. Arbetet med VCL-komponenterna och gränssnittets beteende tog därför längre tid än vad som hade uppskattats i planeringsfasen.
En lärdom från projektet är därför att utveckling av ett administrativt användargränssnitt bör ges större utrymme i tidsuppskattningen, särskilt när gränssnittet innehåller dynamiska komponenter och användarstyrd konfigurering. Ytterligare önskemål som uppkommer under utvecklingen bör samtidigt bedömas mot projektets avgränsningar och tidsplan innan de inkluderas.

7.3 Representation av Excel-arket kodmässigt

För att behandla den importerade tabelldatan behövdes en intern representation av filens rader och kolumner. Ett alternativ som undersöktes var Microsoft Automation, vilket hade gjort det möjligt att arbeta mer direkt mot Excel. Detta valdes bort då lösningen skulle kräva ytterligare integration med det befintliga systemet för en funktion som endast utgjorde en del av migreringsverktyget.
Ett annat alternativ som diskuterades var att skapa en separat `struct` för att representera en rad eller ett tabellobjekt. Efter diskussion med handledaren på företaget valdes istället en tvådimensionell struktur baserad på `std::vector`. Eftersom den importerade informationen huvudsakligen behövde behandlas som rader och kolumner inom migreringsfunktionen bedömdes detta vara tillräckligt för projektets behov. Valet innebär samtidigt att representationen är relativt generell och saknar den tydligare semantik som en egen datastruktur hade kunnat ge. Om samma funktionalitet i framtiden skulle användas på fler platser i systemet eller utvecklas till en mer generell lösning för hantering av tabelldata skulle en separat datastruktur kunna vara mer lämplig. För den avgränsade användningen i detta projekt innebar vektorrepresentationen däremot att ytterligare abstraktion kunde undvikas.

7.4 Resultat och måluppfyllelse

Utifrån de genomförda testerna bedöms projektets huvudsakliga mål ha uppnåtts. Migreringsverktyget kunde hantera samtliga tretton testfiler trots variationer i deras struktur, utan att implementationen behövde anpassas för respektive fil. Detta visar att den valda lösningen fungerade för det testmaterial som fanns tillgängligt och att användarstyrd kolumnmappning var ett fungerande sätt att hantera skillnader mellan källfilerna.
Vid den manuella jämförelsen mellan förhandsgranskningen och informationen i databasen observerades inga avvikelser. Resultatet ger stöd för att migreringsflödet fungerade som avsett i de genomförda testerna. Valideringen var dock begränsad till ett urval av anonymiserade filer från verklig verksamhetsdata och kan därför inte säkerställa att alla tänkbara filstrukturer kan hanteras utan problem.
Målet om efterföljande hantering av migrerade objekt uppfylldes genom att det befintliga administrativa gränssnittet utökades med funktionalitet för granskning, ändring av sensorstatus och tillägg av kommentarer. Funktionalitet för att skapa och ta bort databasobjekt fanns redan i systemet och utgjorde därför inte en del av projektets utvecklingsresultat.
Sammantaget visar resultaten att projektets funktionella mål uppnåddes inom de avgränsningar som fastställdes. Däremot genomfördes ingen mätning av tidsbesparing eller minskad administrativ arbetsbelastning. Det går därför inte att utifrån resultaten fastställa hur stora eventuella effektiviseringsvinster lösningen medför i praktisk användning.

7.5 Begränsningar

Valideringen genomfördes med tretton anonymiserade filer från verklig verksamhetsdata. Trots variationer i filernas struktur kan testmaterialet inte anses representera alla möjliga filformat och datavariationer. Resultaten är därför begränsade till de undersökta fallen.
En teknisk begränsning är att migreringsverktyget förutsätter CSV-filer med en förväntad kolumnavgränsare. Dessutom kan regeln att avbryta bearbetningen vid rader med mer än 90% tomma värden innebära att giltiga efterföljande rader inte behandlas.
Valideringen byggde huvudsakligen på funktionella tester och manuella kontroller. Återkopplingen från en enskild användare ger även ett begränsat underlag för att bedöma användbarheten. Mer omfattande tester skulle därför behövas för att bedöma lösningens tillförlitlighet och användbarhet i fler situationer.

8

Slutsats

Projektet utgick från behovet av att ersätta företagets filbaserade hantering av fastighets- och kundregister med en mer enhetlig databasstruktur. Eftersom informationen lagrades i Excel-filer med varierande struktur behövdes en lösning som kunde hantera skillnader mellan källfilerna och samtidigt bevara relevant information vid migrering.

För att möta detta behov utvecklades ett migreringsverktyg som integrerades i företagets befintliga administrativa Windows-applikation. Verktyget möjliggör import av Excel-data som konverterats till CSV-format, användarstyrd kolumnmappning och förhandsgranskning innan informationen lagras i en ny MySQL-tabell. Information som inte motsvarar databasens fasta fält kan bevaras som metadata i `object_json`. Det administrativa gränssnittet vidareutvecklades även för att möjliggöra granskning av migrerade objekt, ändring av sensorstatus och tillägg av kommentarer.

Vid valideringen användes tretton anonymiserade filer från verklig verksamhetsdata med varierande struktur. Samtliga filer kunde behandlas genom samma migreringsflöde utan att verktyget behövde anpassas för varje enskild fil. Vid manuella jämförelser mellan förhandsgranskningen och den lagrade informationen i databasen observerades inga avvikelser.

Utifrån dessa resultat bedöms projektets huvudsakliga funktionella mål ha uppnåtts. Lösningen visar att fastighets- och kundinformation från filer med olika struktur kan överföras till en gemensam databasstruktur och därefter hanteras genom företagets befintliga administrativa system. Resultaten är samtidigt begränsade till de filstrukturer som ingick i valideringen. Verktyget är även beroende av att källfilerna konverteras till det förväntade CSV-formatet. Framtida arbete skulle därför kunna omfatta stöd för fler filformat, förbättrad hantering av avvikande data samt mer omfattande tester av lösningens tillförlitlighet och användbarhet.

Litteraturförteckning

[1] ProductPlan, “Minimum Viable Product (MVP).” [Online]. Tillgänglig: https://www.productplan.com/glossary/minimum-viable-product/. Hämtad: May 29, 2024.

[2] “Excel User Guide 2025–2026.” Scribd. [Online]. Tillgänglig: https://www.scribd.com/document/834091975/Smart-Excel-Complete-PDF-Book. Hämtad: Sep. 8, 2026.

[3] IBM, “Making Data Consistent Through Standardization.” [Online]. Tillgänglig: https://www.ibm.com/docs/en/iis/11.3.0?topic=jobs-standardizing-data. Hämtad: Oct. 13, 2026.

[4] JSON.org, “Introducing JSON.” [Online]. Tillgänglig: https://www.json.org/json-en.html. Hämtad: Sep. 8, 2026.

[5] Oracle Corporation, “MySQL 8.4 Reference Manual.” [Online]. Tillgänglig: https://dev.mysql.com/doc/refman/8.4/en/. Hämtad: Sep. 14, 2026.

[6] The PHP Group, “What is PHP?.” [Online]. Tillgänglig: https://www.php.net/manual/en/introduction.php. Hämtad: Oct. 13, 2026.

[7] Doctrine Project, “Doctrine Migrations Documentation.” [Online]. Tillgänglig: https://www.doctrine-project.org/projects/migrations.html. Hämtad: Sep. 14, 2026.

[8] “C++ Language.” [Online]. Tillgänglig: https://en.cppreference.com/cpp/language/basic_concepts. Hämtad: Oct. 12, 2026.

[9] Embarcadero Technologies, “Delphi Language Reference.” [Online]. Tillgänglig: https://docwiki.embarcadero.com/RADStudio/Sydney/en/Language_Overview. Hämtad: Oct. 12, 2026.

[10] Embarcadero Technologies, “VCL Overview.” [Online]. Tillgänglig: https://docwiki.embarcadero.com/RADStudio/Sydney/en/VCL_Overview. Hämtad: Oct. 12, 2026.

A
Appendix

A.1 Exempel på testdata

Figur A.1 visar ett anonymiserat utdrag från en av de Excel-filer som användes vid valideringen.

Distrikt	Distrikt namn	Område	Områdesnamn	Fastighetsbeteckning	Telefon arbete	Våning	Alias	Lägenhetstyp	Yta
901	Centrum	002	Högsbo	Kungsladugård 93:2		1	1102	2 rum och kök	53
901	Centrum	002	Högsbo	Kungsladugård 94:1		3	1302	2 rum och kök	53
901	Centrum	002	Högsbo	Kungsladugård 94:1	07X-XXX XX XX	3	1301	2 rum och kök	53
901	Centrum	002	Högsbo	Kungsladugård 94:1		2	1201	2 rum och kök	53
901	Centrum	002	Högsbo	Kungsladugård 93:2	07X-XXX XX XX	2	1301	2 rum och kök	53
901	Centrum	002	Högsbo	Kungsladugård 92:1		2	1201	2 rum och kök	53
901	Centrum	002	Högsbo	Kungsladugård 94:1		1	1101	2 rum och kök	53
901	Centrum	002	Högsbo	Kungsladugård 95:1		3	1301	2 rum och kök	53
901	Centrum	002	Högsbo	Kungsladugård 94:1	07X-XXX XX XX	2	1202	2 rum och kök	53
901	Centrum	002	Högsbo	Kungsladugård 94:1	07X-XXX XX XX	1	1102	2 rum och kök	53

Figur A.1: Anonymiserat utdrag från testdata som användes vid valideringen.

Institutionen för Data- och Informationsteknik
Chalmers tekniska högskola
Göteborgs universitet
Göteborg, Sverige
www.chalmers.se



GÖTEBORGS
UNIVERSITET



CHALMERS