Nico Virtanen

OFFICE 365
LICENCE MANAGEMENT SYSTEM
Implementation and Deployment

Bachelor's thesis
Information Technology / Game Programming

2018

Kaakkois-Suomen
ammattikorkeakoulu

Tekijä	Tutkinto	Aika
Nico Virtanen	Insinööri (AMK)	Huhtikuu 2018
Opinnäytetyön nimi Office 365 lisenssihallintajärjestelmän toteutus ja käyttöönotto		45 sivua
Toimeksiantaja Kymijoen ICT – Kaakkois-Suomen Tieto Oy		
Ohjaajat Niina Mässeli, XAMK; Timo Tuppurainen, Kymijoen ICT		

Tiivistelmä

Opinnäytetyön tavoite oli luoda uusi Office 365 -lisenssihallintajärjestelmä tilaajan vanhan järjestelmän tilalle. Tämä dokumentti kuvaa järjestelmän suunnittelu-, toteutus- ja käyttöönottovaiheita.

Opinnäytetyö toteutettiin toteutuspainotteisella lähestymistavalla hyödyntäen tutkimusvaiheessa kerättyä tietoa. Työn toteutusosa toteutettiin Visual Studio -ohjelmointiympäristössä käyttäen C # ohjelmointikieltä ja .NET-ohjelmistokirjastoa. Projektin alku koostui erilaisten järjestelmien ja toteutusmetodien tutkimisesta, jonka jälkeen itse järjestelmä toteutettiin siihen parhaiten sopivilla työkaluilla. Toteutuksen jälkeen uusi järjestelmä otettiin käyttöön tilaajan tuotantoympäristössä.

Projektin pääpainona oli luoda olemassa olevan järjestelmän tilalle uusi, joka ratkaisisi vanhan järjestelmän ongelmat. Vanha järjestelmä sisälsi tietoturva-aukkoja, jotka poistuivat uuden järjestelmän myötä. Uuden järjestelmän oli pystyttävä kommunikoimaan tilaajan IdM-järjestelmän kanssa, sekä pyörimään kellon ympäri.

Opinnäytetyön lopputuloksena tilaaja sai käyttöönsä uuden lisenssihallintajärjestelmän tuotantoympäristöönsä IdM-järjestelmän käskyläiseksi. Uusi järjestelmä palvelee yli 10 000 käyttäjää kellon ympäri.

Tämän dokumentin lukijalta odotetaan perustason ymmärrystä Windows-järjestelmistä sekä C-pohjaisista ohjelmointikielistä.

Asiasanat

Office 365, C#, Ohjelmistosuunnittelu, Windows-järjestelmät

South-Eastern Finland University of Applied Sciences

Author	Degree	Time
Nico Virtanen	Bachelor of Engineering	April 2018

Thesis Title

Office 365 license management system implementation and deployment

45 pages

Commissioned by

Kymijoen ICT – Kaakkois-Suomen Tieto Oy

Supervisors

Niina Mässeli, XAMK; Timo Tuppurainen, Kymijoen ICT

Abstract

The purpose of this thesis was to produce a new O365-licence management system for the commissioner, which would replace their existing system. This thesis details the process of designing, implementing and deployment of such system.

The thesis was created with implementation-focused approach, making use of the information acquired in the research phase of the thesis. The new system was created using Visual Studio-programming environment, C# programming language, and .Net software framework. Thesis started by examining different systems and implementation methods, and then proceeded to designing and implementing a system with the tools best suited for the job.

The focus was to create a new system which would replace the existing one, solving its problems. These problems included security-issues and oversights, all of which were fixed in the new system. The new system had to be able to communicate with the company's IdM-system and run around the clock.

As a result, a new system was created for the commissioner, which became a pawn of their IdM-system. The new system is serving more than 10 000 users around the clock.

The reader of this document is expected to have some knowledge of windows-based systems as well as basic understanding of C-based programming concepts.

Keywords

Office 365, C#, System Design, Windows-system

TABLE OF CONTENTS

ABBREVIATIONS AND GLOSSARY

API – Application Point Interface, A set way of communication between software and/or hardware components from code.

GUI – Graphical User Interface.

Hack – A system, solution or a method what might get a job done, but in an inefficient, non-optimal or "ugly" way.

High-level programming language: A Programming language, which usually provides the user simple ways to interact with and implement desired functionality.

IDE – An Integrated Development Environment, usually a program what offers basic tools and other features which are needed in software development.

IDM – Identity Management, management of individual identities, their authentication, authorization, roles and privileges within or across system and enterprise boundaries.

IDM System – A system what automatizes IDM Management inside organizations.

MySQL – An Open-source database system.

Network Zone: Isolated part of the network, usually a part of a larger infrastructure.

O365 – Office 365, Microsoft's Online Platform what provides business tools via subscriptions.

PHP – Hypertext Preprocessor, A scripting language used mainly for creating web-applications.

SQL – Structured Query Language.

SVN: Apache Subversion, an open source version control system.

1 INTRODUCTION

The modern world has grown heavily dependent of information technology. Complex systems automate many functions of today's society which were once done with manual labor. New systems are rolled out every day, and this in return has created many new job opportunities for tech-savvy people. After all, like most things, these systems must be maintained to keep them running smoothly.

However, not all systems behave optimally. They might accomplish what they were meant to do, but they might be something called "hacks". A hack is a programming term which means a system, solution or a method that might work, but in an inefficient, non-optimal or "ugly" way. Programmers might submit to "hacks" because they either lack the skill, knowledge or time to implement a proper solution. These hacks can be dangerous as they can compromise the system's security and may cause unforeseen problems.

1.1 Objective of the thesis

This thesis was commissioned to replace the commissioner's Identity Management (IdM) system's existing method of communicating with Microsoft's Cloud-based Office 365-system. The old system had its fair share of problems. It was slow, insecure, and prone to errors. The new system aims to fix these problems by providing a new, modern and secure solution for communicating with O365 platform. This thesis is a documentation of implementing such system and deploying it to an existing production environment.

The thesis is divided into three parts. The first part will introduce the tools and techniques that will be used to create the new system. The second part is solely focused on program design and implementation, while the third part documents the deployment phase of the project.

1.2 Commissioner

This thesis was commissioned by Kaakkois-Suomen Tieto Oy. It is a moderate-sized company located in Kymenlaakso, Finland, and currently has over 60 employees. The company maintains the ICT-infrastructure for the municipalities of Kotka and Kouvola. It was founded in 2010 by the city of Kouvola, sometime after the neighboring municipalities were merged with it.

The commissioner proposed the project during the fall of 2018, after the author of this thesis worked at their company for the duration of the summer. The Specifics of the system were discussed with the commissioner, and an agreement to create a new system was made.

2 REQUIREMENTS OF THE SYSTEM

Before work could begin on the new system, a meeting was held with the commissioner's system designer, Timo Tuppurainen, who would define the requirements of the new system. Defining the requirements of software projects is important because the project is easier to manage and plan when the needed functionalities are documented and defined. The new system would have to have at least the same functionalities as the existing system, fix its problems, and must be able to:

- Be integrated into existing infrastructure
- Grand, remove and check licenses of O365 users
- Be robust, configurable, and secure
- Able to run around the clock without interruptions
- Able to handle a large workload of a 10000+ user environment
- Work independently without any user input

The new system would be integrated into an existing IdM-solution which is running and processing orders around the clock, managing user rights for Microsoft's Office 365 platform. Therefore, should the new system fail to function, it would cease all IdM operations relating to O365.

All maintenance and reconfiguration would have to be done outside of office hours, with minimal downtime. The system should be able to recover from errors by itself. With the requirements set, work began on the new system.

3 PREPARATION AND RESEARCH

If the requirements were to be met, the new system would need a strong foundation of knowledge to build upon. Research started without prior knowledge of the inner workings of O365 licensing scheme. The assumption was that it would have complex API to communicate and exchange information with, so the project had an initial assumption that it would take 3 months of full-time work to complete.

3.1 How the existing system worked

Research started by interviewing the existing system's creator on how the communication was implemented. O365 uses a PowerShell-module called "MSOnline" to send commands and queries to the cloud. The MSOnline module is used to manage Azure Active Directories with PowerShell, and since O365-tenant information is stored inside an Azure AD, they both use the same module for administrative work (Martin M, 2017).

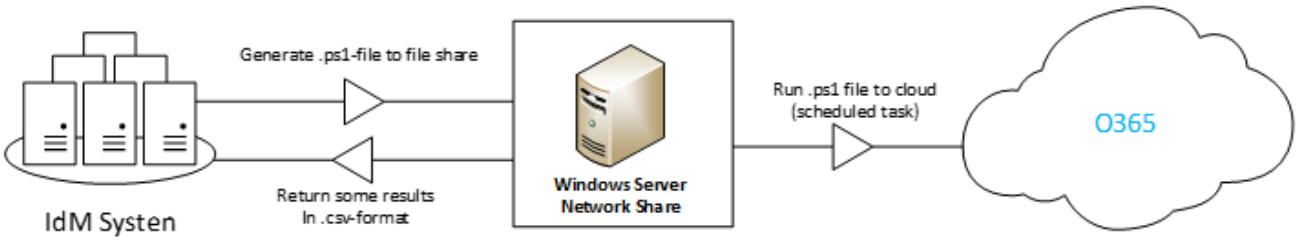

Figure 1 - How the existing system worked

As shown in the image above, commands were generated into a PowerShell-script on a Linux-machine and then saved to a windows-based network share running on another machine. The share host would then periodically execute the contents of the file to perform the needed operations.

This method is extremely unsafe as the system does not check the contents of the generated file before execution. If someone were to be able to write his own code into the file, the system would execute it with highest privileges without any second doubts. This would allow the attacker to basically take over the machine as PowerShell is able to alter almost all components and functions of a Windows-based system (Lee, 2011).

After executing the script, a .csv file would be generated to the network share, which would sometimes contain some results of the script. The Linux machine would then periodically check the contents of this file and perform actions based on the results. There was little to no error checking included in this whole process, and that caused much unnecessary work and investigations for the company.

3.2 How to communicate with O365 platform

The user has two options for communicating and doing administrative work with the O365 platform. The first option is to do everything by hand with the web-based graphical interface. However, this process is cumbersome and unsuitable for environments with thousands of users but is well suited for managing a small number of users in small companies.

The other option is to use the MSOnline-Powershell module which can also be used by hand, but its real power lies in it automation capacities. The user can create his own systems and methods which can use the module’s available commands and queries as they please.

3.3 Interfacing with company IdM

The Company's IdM-system was written with PhP-scripting language and it ran on a Linux-based system. The IdM made extensive use of a MySQL-database engine to store and manage data.

MySQL is a hugely popular and well-supported database engine which has different kinds of plugins available for a large selection of programming languages. As databases are easy to interface with, it was decided that the new O365-system would communicate with the commissioner's IdM with database-connections.

3.4 Databases and security

When choosing the database engine for the new system, several things had to be taken into consideration. The new O365-systen was going to be run on a Windows platform, so a Microsoft SQL Server would have been a natural choice. This would be a secure way of implementing database connections, as the database uses Window's build-in credentials for authenticating connections. However, this turned out to be a bad idea later for various reasons which are detailed in chapter 5. So instead, MySQL was chosen for the job.

Systems using databases must deal with one major security issue for such system: SQL Injection. SQL Injection is a vulnerability that occurs when a user is given the ability to influence SQL queries that an application passes to a back-end database (Clarke-Salt, 2014). For example, if a website has a form which the user can write his name into, a SQL injection might be able to be executed by entering SQL-statements into the form instead of a username. A Poorly written program might not check the input for an injection attempt, and depending on the setup, the attacker might be able to access most of the database freely with the help of the poorly-coded username-form, or even delete the whole database.

SQL Injection can be prevented by multiple ways. One technique is to filter and validate the input given to the system, for example, by checking for black- and whitelisted characters, or checking if the input is in the expected form. Another option is to use parametrized SQL-queries which are pre-compiled SQL-statements where only parameters can be customized (Clarke-Salt, 2014). Database admins should also grand privileges to database credentials sparingly, so in an event of SQL-injection, the attacker is only able to access some of the database and unable to cause major harm.

3.5 24 / 7 Availability

One of the best ways to implement a 24 / 7 running program on a Windows environment is to make it run as a Windows-service. Windows services are long-running programs what operate independently in the background and require no user input (Windows Service Documentation, 2017). They can be configured to start as soon as the Windows-operating system is loaded into the memory and run until the operating system shuts downs. In simple terms, a Windows service is running in the background by itself if the computer is powered on.

In order to ensure maximum uptime of the service, an external monitor can be attached to a service to monitor it. This can be programmed to react to status changes of the service. For example, if the service shuts down for some reason, the monitor can automatically restart it. There are many ways of implementing this, by using manual programming or ready-made software.

After preliminary research, it was decided that the new system would be implemented as a Windows-service application and it would use MySQL database connections to communicate with the commissioners IdM, and use the MSOnline-Powershell module to communicate with the O365-system.

3.6 Tools and Techniques

After research, it was time to choose how the new system would be created. This chapter introduces all tools and techniques used in creating the system.

There exists a wide range of different tools and products what a programmer can use to accomplish his tasks. In the end, the tools chosen were the tools what the author of the thesis was most comfortable with.

3.6.1 Microsoft Visual studio

Visual Studio 2017 Is a Programming IDE released by Microsoft. It focuses on building applications targeting Microsoft platforms but it can also be used to build applications for various other systems using multiple programming languages (Chowdhury, 2017). It offers a large selection of tools and features what are designed to make software development easier and more effective. For example, it uses IntelliSence-technology for writing code, which automatically checks the written code for errors and offers auto-completion for code snippets. Other tools include, but are not limited to, Windows-software templates, build-in debugger, and support for extensions.

This IDE was chosen for the task because it offers a vast selection of helpful tools and has built-in support for many Windows-features what would be needed, for example, creating windows service-applications.

3.6.2 C#

C# is a simple, general-purpose, object-oriented programming language developed by Microsoft in the early 2000s (Jagger, 2007). It is an object-oriented language, which means that it roughly follows the concept of creating multiple “objects” for various tasks which are then programmed with functionality designed to carry out a specific task. For example, one object might be in charge of all networking-related functionality, while another object is specialized in processing text files.

C# was chosen mainly because of .NET-framework. The .NET-framework is a software framework for C# created by Microsoft. It includes a large library of pre-made functions and classes what can be used with Windows-operating systems. As the project would be running on a Windows-machine, C# was the programming language of choice in this thesis study.

3.6.3 SQL Language

SQL, or Structured Query Language, is a language used in controlling various database-systems. While its syntax is mostly the same between different systems, the queries for different database systems usually have their quirks what the user should be aware of.

At the start of the study, it was decided that Microsoft, MSSQL would be used since the database that was to be used was using Microsoft SQL Server, but later the SQL language of choice was changed to MySQL because of problems outlined in chapter 6.

3.6.4 PowerShell

PowerShell is Microsoft's task automation and configuration framework for Windows-systems. It provides tools to manage and automate tasks on Windows systems and applications that run on them (Lee, 2011). In addition to its command line usage, it also allows users to create scripts what can be run on demand.

PowerShell is usually used to automate desired processes and access functionalities what might not be available in graphical form in Windows-platforms. As the O365 API is accessed with PowerShell commands, it was a natural choice.

3.7 Creating the development environment

Before actual work could begin, an environment for testing and producing had to be set up. The commissioner of the thesis had a “demo-environment”, which was used to demonstrate their products to potential customers. The environment had its IdM-system already set up and running, which could be used to test the new system on a small scale.

As the test environment was already up and running, only systems needed in the actual development work needed to be set up. This includes the test server, and the development machine.

The new system was going to be run on Windows-platform. A Windows Server 2016 operating system was installed to a VMware virtualization platform. The server would be used as the test machine for the project. The commissioner provided a Microsoft Surface Pro-hybrid computer what would be used as the main development computer.

Testing had to be done on a separate computer. The company’s network had a strict security policy, so computers in different network zones could only communicate with each other in pre-determined ways. The development machine would not be able to contact the server located in the demonstration environment by database connections, and vice versa. The program would be copied to the server during testing manually.

The chosen IDE, Visual Studio, was acquired for the project via the company’s Visual Studio subscription and installed in the development machine. In addition to the access to the software, the subscription allows its users to download various Microsoft operating systems and products for free to be used in development. The chosen version of the IDE was the newest, 2017. MSSQL server was downloaded from Microsoft website and installed to the test server. For source control, a VisualSVN server was installed to an existing company server and it would serve as a SVN repository in this thesis study.

4 DESIGNING THE PROGRAM

This chapter describes the features and functionality what the program will have. First, the main features and addons are explained, and then the program structure is explained in detail.

Basic C# functionality and implementation is not explained. Instead, this chapter focuses mainly on the functionality and PowerShell-communication of the system.

4.1 Overview

The following figure shows a broad overview of the planned system's components and functionality:

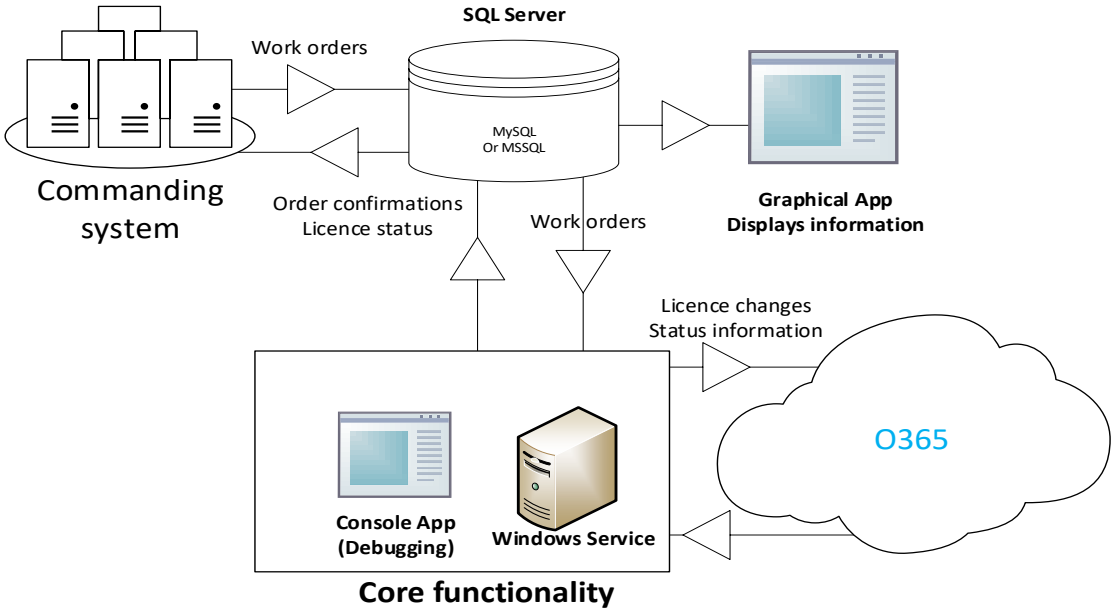

Figure 2 - The implementation plan for the program.

As shown in the above figure, the Program will become an underling of an existing system. It does not care what system will command it, the only requirement of the commanding system is that it can read and write into a database system.

The commanding system inserts orders into a database table for the program to fulfill. The program will periodically check the database for new orders and once it finds work, it will execute them accordingly. It will have support for the following orders:

- Add a license to a user
- Remove a license from a user
- Add a license to a user, and execute pre-defined scripts after licensing

In addition to order-based work, the system will also do some work independently from orders. This includes keeping track of cloud's license-situation and storing it in a SQL-database, and tracking user license usage.

The system's main program will have multiple versions of itself. A Debug version will be created as a Windows console application, which will display verbose information of the program's actions. It will be used for debugging purposes since all errors and actions can easily be displayed on the screen. A Windows Service-version will be implemented and used when the program has been deployed into production environment. It will be running 24 / 7, even if no users are logged on.

Lastly, a GUI-version of the program will be implemented. It will be a graphical tool for browsing the program's database. Administrators can use it to check the license-situation, program workload and user license usage. It is also used to change program settings graphically.

4.2 Addons and libraries

The program will make use of C #'s core functionalities. Operating System functions will be implemented with the .NET-libraries. These include file-management and program control.

Some addons will also be needed for some of the planned functions. These will be detailed in the coming chapters.

4.2.1 MySQL Data

MySQL.Data is a NuGet package. A NuGet package is a downloadable package for Visual Studio, which usually contains precompiled code in the form of .dll-files. In their most basic form, they are addons what add extra functionality to a code project.

MySQL.Data package provides the tools to interact with MySQL databases. As the program will contain support for MySQL-database, this was an essential package for the program. It uses the same syntax for its commands as standard C# SQL commands except that all commands have the My-prefix.

4.2.2 SMA

System.**M**anagement.**A**utomation is also a C# NuGet package. It gives the program an ability to run PowerShell-scripts natively in code. As all communication with O365 is done via PowerShell-scripts, this was also an essential add-on. After adding the package to the project, PowerShell can be invoked as follows:

```
PowerShell ps = PowerShell.Create();
ps.AddScript(PSScript);
ps.Invoke();
```

First, a PowerShell object is created. After creation, scripts can be added to the object as needed. The AddScript-function can be called multiple times with different script-parts. When all wanted scripts been added, the script collection is executed with Invoke-function. If the output of the script needs to be processed, one way of doing so can be seen below:

```
 var results = ps.Invoke();
 foreach (PSObject result in results)
Pname = Convert.ToString(result.Properties["ResultName"].Value);
```

The Invoke-functions returns a collection of PSObjects which can be parsed to different values. The code above loops through the results, looks for a value named “ResultName”, and saves it to a variable.

4.2.3 MSOnline

MSOnline is a PowerShell module, which contains commands for communicating and commanding Microsoft Azure Active directories. Conveniently, O365-tenants use Azure AD:s to store user data, so most of the commands in the module can be applied directly to it (Martin M, 2017).

The module can be used to read and write data into the cloud, which makes it a powerful tool for developers. It was included to allow running of automated PowerShell commands to the cloud. In order to open the connection to the cloud, a PowerShell command is used:

```
Connect-MsolService -Credential $Creds
```

The command needs a valid PSCredential-object to work. A PSCredential object is an object which contains one or more Windows-user credentials. This object must be created with O365 credentials which have admin privileges in the cloud environment. One way of creating a credentials looks like this:

```
$Pass = ConvertTo-SecureString 'Password'" -AsPlainText -Force
$Creds = New-Object System.Management.Automation.PSCredential
('Username', $Pass)
```

As shown above, the password is supplied in cleartext format without encryption. This is a security risk, so the password should be stored somewhere else in the program encrypted, and momentarily decrypted for the commands as needed. Once the connection has been opened, the program has access to a large number of PowerShell-commands what can be used to manipulate the cloud environment.

4.3 Database structure

The program will need several database tables to function. For data processing, it will need 3 tables. One for new work-orders, one for orders in progress, and one for completed orders. The commanding system will insert a work-order into the table for new orders. Once the system picks it up, it will move it into the in progress-table, and once it is complete, it will finally move to the table for completed orders.

For tracking, the program will need 3 tables. The license table will contain all licenses currently assigned to users. The license status table contains an overview of the company's owned licenses. It will contain all names of the owned licenses, and their usage information (Used, Available, Expired). License usage data will be stored in a table, which will contain each user's username and the last time when they have used their license.

Finally, the program needs a table to store its dynamic settings. These settings are values what might need to be changed dynamically when the program is running. A settings table will be created for these settings, and it will contain settings with a name-value pair presentation.

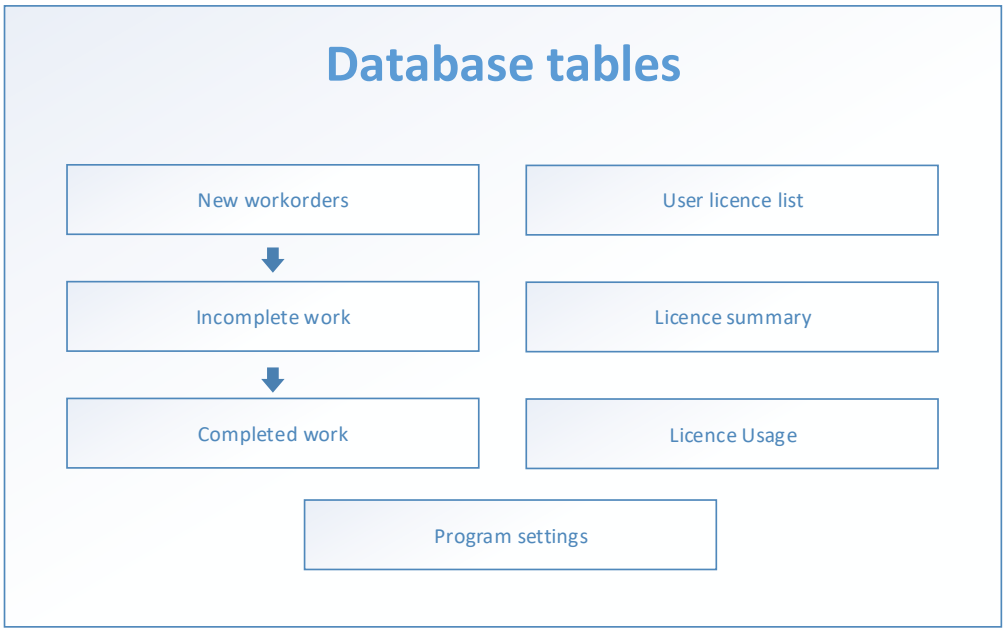

Figure 3 - Planned database tables

4.4 Interfacing with the program

As the program is controlled with database tables, the commanding system will communicate with the program using SQL tables. As mentioned in the previous chapter, new orders are made by inserting information into the worktable. The program will want to know:

- Username of the user which the order concerns
- License which the order concerns
- Order action ID
    - 1: Add license
    - 2: Remove license
    - 3: Add license and execute additional scripts
- External order ID of the work order

The parameters are self-explanatory except for the external order id. It will be used by the commanding system to uniquely identify the order. Completed orders are moved into the completed worktable. The commanding system should periodically check this table for completed orders. Each entry will have some additional meta-information about the order:

- Completion time
- Result
- Acknowledgement

The result of the order will tell the commanding system if the order succeeded. For succeeded orders, it will be a positive number, the same action id the order was made with. However, if something went wrong with the order, it will be a negative value. These are detailed in chapter 4.6.2. The acknowledgement value should be set to 1 by the commanding system to notify the program that the commanding system has received the completion notification.

Acknowledged orders are periodically removed from the table. This way, the database does not grow over time. The commanding system can also use the information located in the license-tables as it wishes, all it needs is to use SQL queries against the table.

4.5 Class design

This chapter will introduce the details about the program's class design. Each class will have its own specialized job in the program, following the object-oriented programming paradigm.

4.5.1 Logging

For logging purposes and debugging, a static **Logger** class will be created. It will support logging into different locations, such as console screen, Windows event log, and a database-table.

The class was made static so it would be easily accessible from different parts of the program. Generally, static classes should be used sparingly since they can be accessed freely from anywhere in the program. This can cause problems in larger projects with multiple programmers, but in this case, the static class does not cause any issues.

4.5.2 Database-classes

Database functionality will be implemented into three different classes, each with its own specialty. All settings and their related functionality will be implemented into a class called **DatabaseSettings**. It will store all variables needed in connecting and querying the used database engine. For example, it will contain the address of the database server, table names, and user credentials. User credentials will be encrypted to increase security.

A second database class will be responsible for handling database connections. This will be named **ConnectionManager**. It will use the settings defined in DatabaseSettings to open, test and close database connections.
The two classes will be nested inside the main database class, simply called **Database**. It will be used as a wrapper to wrap the database-related functionality inside a single class, and the database-functionality will be exposed as needed with functions. The wrapper class will be capable of reading .XML-files, which will be used to store program settings.

4.5.3 Worker-classes

Two classes will be created for handling the ‘core’ functionality of the program. **WorkMaster**-class will act as an overseer and periodically checks if the program has been asked to do something. It will execute work based on requested actions and it will use SQL connections with the help of the main database class to communicate with the database.

A **PSGenerator** class will help the WorkMaster class in executing orders. PSGenerator will be responsible for all work what involves using PowerShell commands, for example, reading information from the O365-platform.
In addition to the main worker classes, a few helper classes will be created to make data processing easier.

4.5.4 Security

A Cryptographic class called **StringCipher** will be implemented to encrypt and decrypt configuration information for the program as needed. As the program will store the usernames and passwords in its configuration file, they will need to be stored in encrypted format to maximize security.

The encrypted values will be loaded into the program’s memory on startup. From there, they can be momentarily decrypted when their information is needed. Otherwise, they will stay encrypted.

4.5.5 Wrapper

All classes described in this chapter will be wrapped inside a single class called **Core**. Basically, this class is the program, since it implements every function needed to fulfill its duties.

This approach makes the program very mobile and gives it an addon-like functionality. The core functionality can be included into any C#-project just by including the wrapper class. This was implemented because the core functionality will be included into several different versions of the program.

4.5.6 Summary

An overview of the program's class structure is detailed in the figure below:

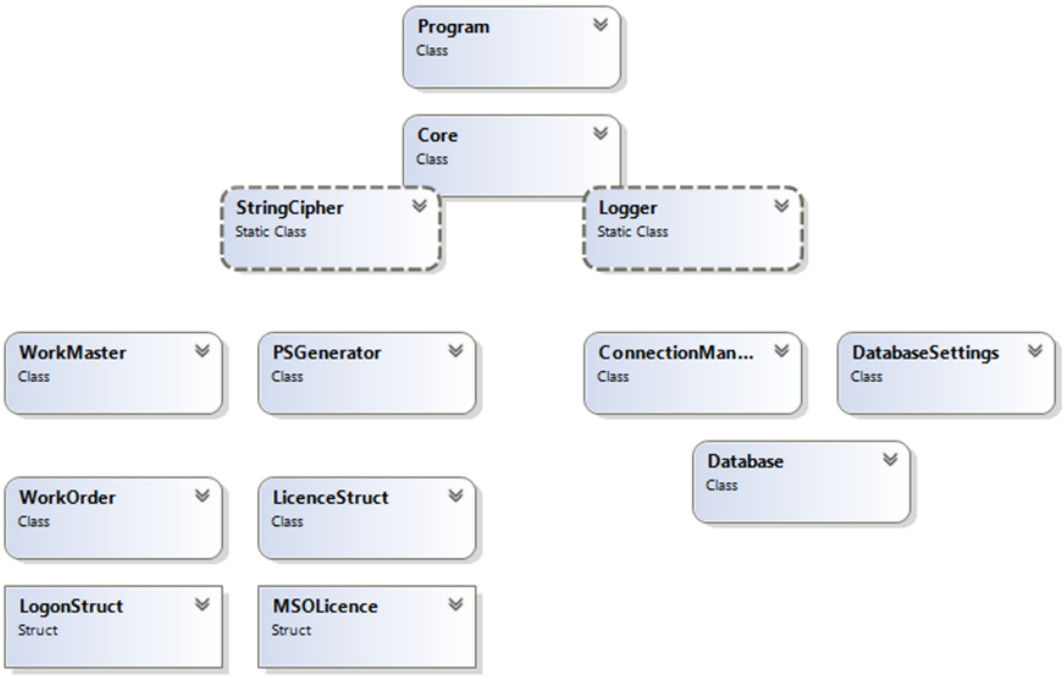

Figure 4 - Program's classes

A program can include the core-functionality into itself by including the Core class. The Core class can be included basically into any C# project, be it a console program, a GUI-Project, or a Windows Service.

The program contains two static classes, StringCipher (encrypting and decrypting), and Logger (for keeping log). The database functionality is implemented into 3 database classes (seen on the right), which will handle the database related tasks. The main functionality is wrapped into two worker classes (seen on the left) what contain the actual functionality of the program. They have access to several helper classes (bottom left), which make data processing easier.

4.6 Program Structure

As the program is going to run around the clock, the whole functionality will be wrapped inside one big loop structure. The loop structure will then be divided into smaller “steps”, which each execute their specific tasks. The program will continue to run this loop unless it encounters an error, or is told to stop.

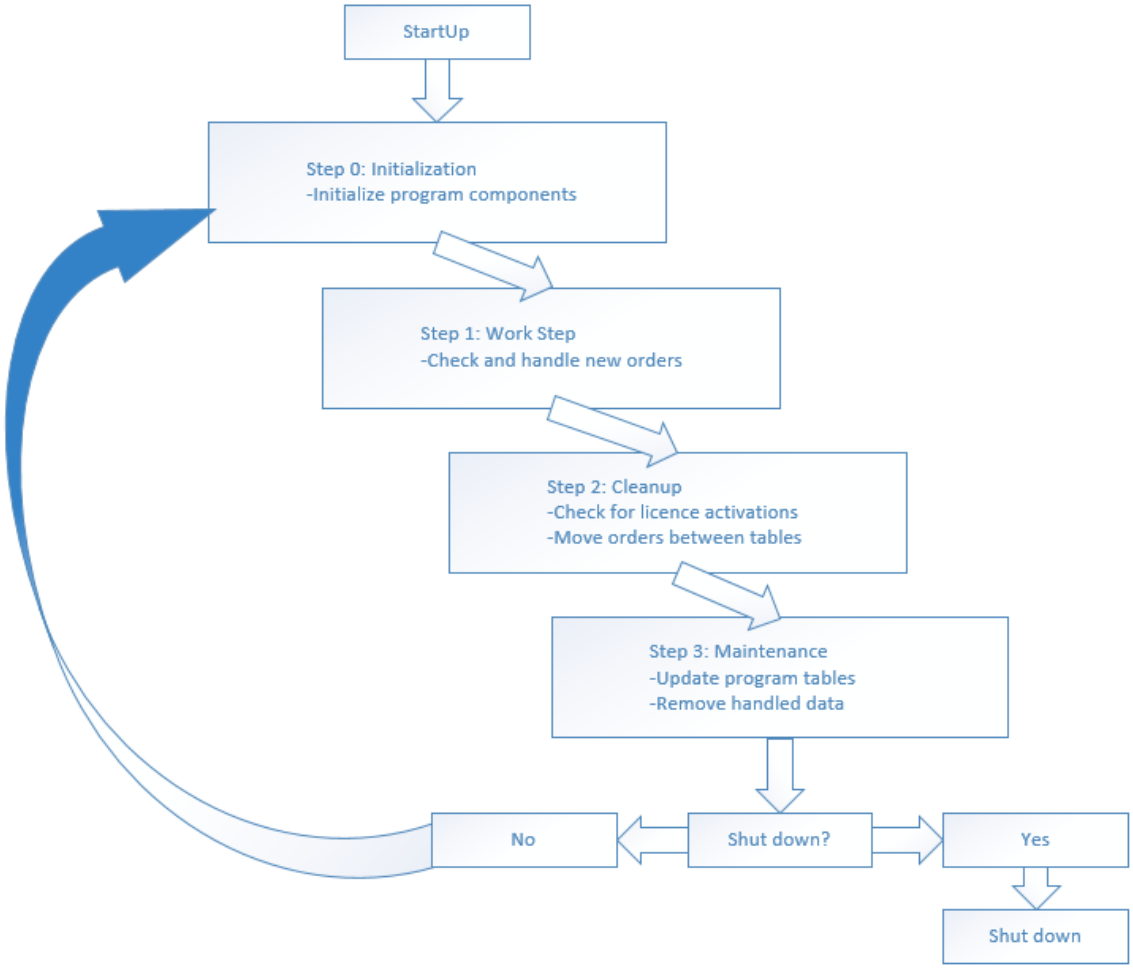

Figure 5 - Program loop structure

The above figure is a quick overview of the program’s work sequence. The following chapters explain each individual step in in detail.

4.6.1 Pre-execution step

When the program first starts, it enters a step called “pre-execution step”. In this step, the program tries to initialize all of its components and stops execution if it encounters and error. During initialization, the program reads the contents of its config file and assigns variables to classes accordingly. If it fails to read a variable into memory, or if the variable is invalid, it stops execution.

If the program is successful in reading the variables into memory, it will begin to initialize its components. First, it will check if it manages to establish database connection with the provided information. If the database connection fails, the program stop execution.

If the database connection test succeeded, the program proceeds to initialize the O365 connection. It tests that if it is able to open a connection to the O365 cloud servers using the MSOnline PowerShell module. The connection is validated by querying the cloud with the provided O365 admin username. It finds the admin username in the cloud; the program deems that the test was successful and proceeds forward.

After testing the database and office 365 connections, the program proceeds to update its license-tables. It will first update the user license table, which contains all assigned licenses with username-license pairs. After that, it will generate a license report and store it in the license-summary table. The table contains all owned licenses of the company and their usage. (Used, available, expired). After that, the program enters the big main loop which it will keep executing until told otherwise.

4.6.2 Work step

The work step is the first step of the big main loop. It mainly consists of handling new orders. First, the program checks the database's work table for new orders. Technically, it executes a "SELECT * FROM TABLE"-SQL statement against the work table. If the query returns any results, it will loop through them and temporarily store them in an array of WorkOrder classes. Before storing them into the array, some checks are performed. First, it will check if the order is valid. Invalid orders are marked as such. The order is valid, if:

- None of its fields are empty
- Order username is sufficiently long
- Order action id is between 1 and 3
- There is no pending order with the same username, license and action
- The ordered license exists in license-summary table.

After performing the first validation, the program checks if the provided usernames exist in the cloud with the help of the PSGenerator class. The Work Order array is passed to a function, which performs a query to the cloud with the provided usernames. The cloud will then return a result for each username that exists there. After that, the program compares the usernames of its order list entries to the cloud's returned values. If the order's username does not exist in the returned list, the order is marked as invalid.

When the cloud-check has finished, the program will remove all invalid orders from its worktable, and move them to the completed work table. They will have a negative action-result so the commanding system will know that something went wrong. The list of possible negative action values includes:

- -1: Duplicate order
- -2: Invalid fields in order
- -3: Ordered license is invalid
- -4: Order username is invalid

Orders what were not removed from the WorkOrder array by invaliding them are processed in the next sub step.

In the next sub step, the program will move forward orders what were left to the WorkOrder array. First, it will check if the order still exists in the work order table, and if it does, it will move it to the incomplete work-table. It will also remove its information from the WorkOrder array.

After the tests, it is time to start working on the orders. The program will retrieve the data of every single order in the incomplete-work table, and store them in an WorkOrder-array. After storing the information, the program generates PowerShell-code based on the order's action id. The action id tells the program what kind of PowerShell code it should generate. The id has 3 possible values, which were explained in in chapter 4.4. PowerShell commands to manipulate licenses are:

Set-MsolUserLicense -UserPrincipalName <UserName> -AddLicenses <LicenceName>

Set-MsolUserLicense -UserPrincipalName <UserName> -RemoveLicenses <LicenceName>

After generating the script for each order, the script is executed to the cloud. The program has now finished working on the new orders as it has moved them into the incomplete work-table and ran the script to the cloud. This is the end of the Work step.

4.6.3 Cleanup step

After executing orders, the program waits 10 seconds before moving into the next step. This pause gives O365 some time to process changes. After the pause, the program fetches all orders from the incomplete work-table and stores them into an array. It then proceeds to generate PowerShell-script based on the orders. It will generate the following script:

Get-MsolUser -UserPrincipalName <UserName> | *select UserPrincipalName -ExpandProperty Licenses* | *Select UserPrincipalName, AccountSkuID*

The script queries the cloud for information about the supplied username. It then filters the returned information and outputs a table which contains the usernames and licenses of licensed users. After retrieving the information, the program will compare its worklist with the retrieved license table. For each license-order, it will check if the returned table contains a valid license for the user. If it does, the order will be marked as complete. For license removals, it does exactly the opposite, so if the user is not found in the license-table, it will mark the removal order as complete. Orders that were not marked as complete will be processed again in the next loop cycle.

After license checking, it is time to clean up the work tables. First, the program will query the incomplete work-table for orders what have been marked as complete. For each completed order, it will move it into the completed work-table, and add a completion timestamp to it. After moving, the program proceeds to clean up the completed-work table. It will look for entries that have been marked as acknowledged by the commanding system and delete them.

4.6.4 Maintenance step

The last step in the cycle is called maintenance step. During this step, the program updates its tables and settings. First, it will update the license overview-table, which shows the number of licences owned, in use and expired. This is accomplished with the following PowerShell command:

```
Get-MsolAccountSku
```

After parsing the information, the license summary-table is emptied and then updated with the new information. Then, it is time to update the large license-table, which contains all licenses owned by single users. This is accomplished with a PowerShell command:

```
Get-MsolUser -All | Select UserPrincipalName -ExpandProperty Licenses
| Select UserPrincipalName, AccountSkuID
```

The command fetches a list all users residing in the cloud and selects all licenses assigned to them. It will return a table with user–license values. The returned information is then parsed and inserted to the program's license-table.

Next, the program updates its dynamic settings from the settings-table. The settings are stored in the table as variable–value pairs. The functionality of retrieved settings must be programmed separately. These settings can be used, for example, to tweak the user creation process. Action ID 3 is reserved for adding licenses to users and running special scripts. A setting might tell the program, for example, to set the users language for Finnish during the license assignment.

The last sub step in the program is simply called maintenance. It will only be executed, by default, once a day at night. This step is reserved for operations which take a very long time to complete, and for this reason, they are executed at night when the workload of the program is at its lowest.

First, the program will update the license usage monitoring data. This is accomplished by using Microsoft Exchange PowerShell commands. These are accessed by creating a new PowerShell session with the O365 admin credentials, and then querying the cloud for the last time the user has used his O365 account. This is an extremely slow process and can take multiple hours in environments that have several thousand users. After updating the usage information, the program can be configured to execute other actions as needed, for example, adjusting O365 user settings in the cloud environment.

4.6.5 Program Structure Summary

The previous chapters detailed the program's workflow in detail. It starts from order processing, and ends in maintenance. Summarized, the program's structure looks like this:

0. Initialization

    0.1. Initialize program components
    0.2. Update license-tables

The main loop begins

1. WorkStep

    1.1. Check for new orders
    1.2. Handle new (possible) orders
    1.3. Start working on new orders

Wait 10 seconds.

2. Cleanup-step

    2.1. Check for license activations
    2.2. Move completed jobs to completed work-table.

3. Maintenance Step

    3.1. Update license summary
    3.2. Update user license-table
    3.3. Update program settings
    3.4. Perform maintenance and long tasks (once a day)

Main loop finishes.

Should the program continue running?

Yes -> wait 10 seconds and do the loop again

No -> Exit the loop and close the program

4.6.6 Error Handling

As a large portion of the program code is capable of throwing exceptions, a proper error handling procedure had to be implemented. An exception is an error state within the program, which usually occurs when a requested operation fails inside the code. In this program, the main exception throwers will be the database classes as SQL-related code can fail for many different reasons and they are constantly in use.

When an exception is thrown, the program will ‘crash’ or ‘hang’ if the exception is not handled. Exceptions can be handled by encapsulating the error-prone code inside a try / catch blocks. When an exception is thrown inside a try / catch block, the program will not crash, and the programmer has a chance to handle the exception. As the program is going to be ran as windows service, it is vital that all exceptions are caught since the program is going to run around the clock.

Exception handling was implemented using the fine-grained approach. This approach is useful if there are multiple different exceptions what need to be handled in a specific way (Hiliyard, 2015). Exception throwing code is surrounded with try / catch blocks, and their exceptions are handled accordingly. In this case, the codebase contains a large amount of database and PowerShell-functionality whose exceptions need to be handled accordingly. In case of program crashes, an external monitoring software will be attached to the program service and it will try to restart the program when it is not running.

4.6.7 Program configuration

The program will be configured with an external GUI tool what will set and encrypt the user credentials used in running the program. Testing tools will be implemented to ensure that the database and cloud connections are working with the provided configuration.

Configuration will be stored inside an .xml-file. It will contain necessary settings, such as database configuration and encrypted user credentials.

5 CREATING THE PROGRAM

This chapter describes the different versions of the program what were created. It also details the problems what were encountered during creation.

5.1 Command line version

A Command line version of the program was created using Visual Studio's standard C# console application template. It was created for debugging purposes since outputting text to the screen in real time is very easy with console applications.

This version will be mainly used to test out the program's configuration and to debug program functionality in case of errors. It functions as explained in the previous chapter.

5.2 Windows service

A Windows service version of the program was created with a predefined Visual Studio template what was customized to work with the main features of the program. This version of the program would be used in production. Since Windows services work a little differently from a 'normal' programs, the functionality was split into 3 different branches. When the service first starts, it will execute the program initialization and tests. If something went wrong, the startup is aborted.

When the services are running, they only execute code on demand. The main functionality of the program is tied to a timer object what executes the main work loop of the program once a minute. The timer checks if the previous loop is running before starting a new one.

Since services need to be shut down 'nicely' by telling them to stop instead of simply shutting them down, a shutdown procedure was implemented. When the service receives a command to stop, it will stop the timer from starting any more work loops. It will then periodically check if a work loop is in progress and shuts down the service if no loop is running.

5.3 GUI Tool

A graphical tool for the program was implemented using Windows Forms framework. Windows forms is a .NET class library what provides a set of classes for writing form-based Windows applications (Templeman, 2002).
The graphical interface enables users to see a visual presentation of the data created by the program. A user can use the graphical interface to:

- Check an overview of the company's owned O365 licenses
- Check licenses owned by individual users
- Check for unused licenses
- Change program settings graphically

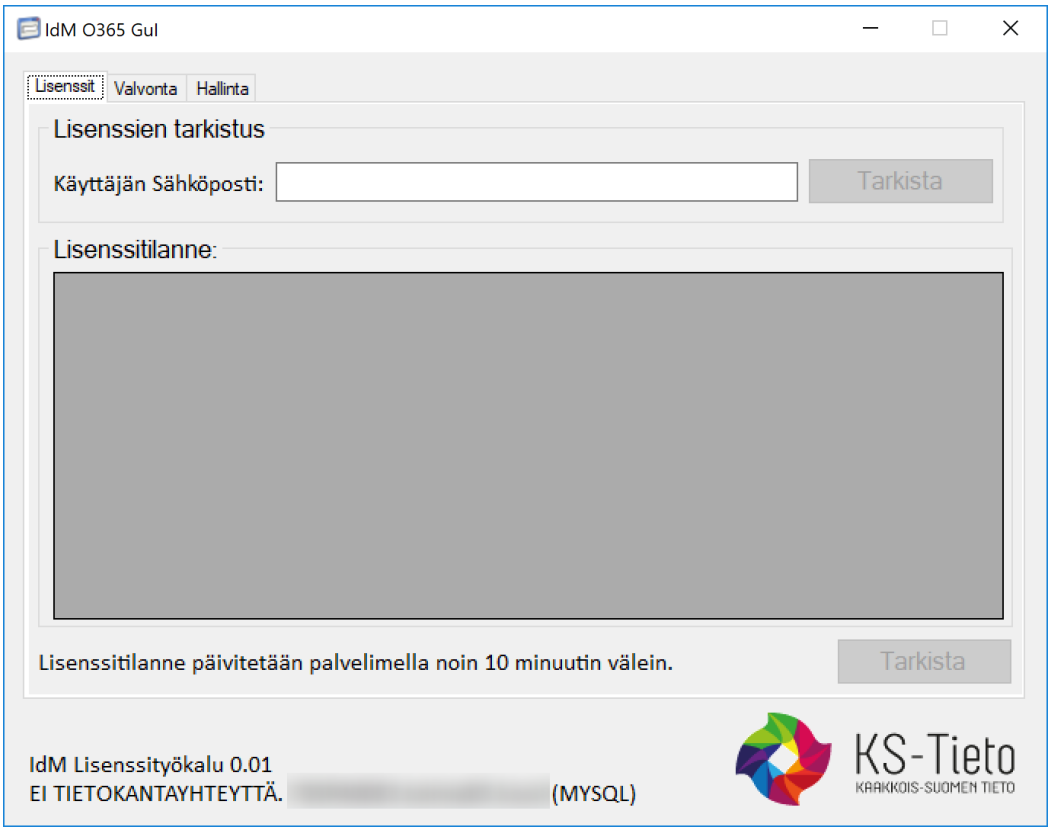

Figure 6 - Image of the graphical tool

5.4 Config Tool

A Configuration tool was created for the program in order to help administrators install and configure the program to work in their environment. The config tool was created with Windows Forms framework.

The program's initial configuration is created by using the tool. The user is required to input all needed credentials for the system, including SQL and O365 credentials. These credentials are then tested to see if a successful connection can be opened by the program to their respective systems. If the connections are successful, the program will permit the user to save the configuration on hard disk. In order to increase security, usernames and passwords are stored in an encrypted format.

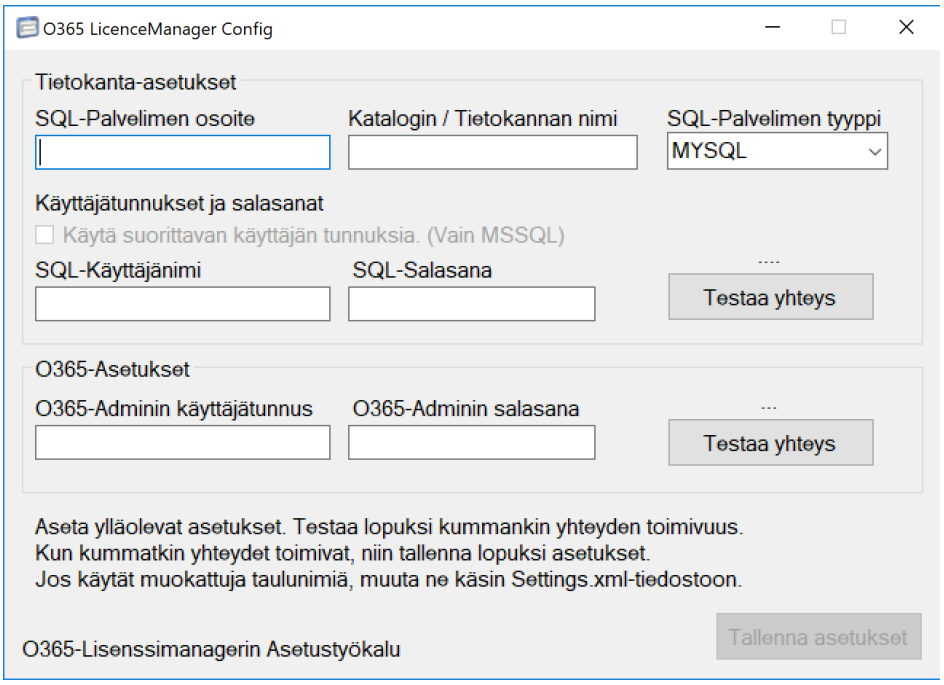

Figure 7 - Config tool

5.5 Installer

In order to make the program installation procedure easier, a simple installer was created. The installer is a simple .bat-file what contains commands to:

- Install program prerequisites from supplied .msi packages
- Launch the configuration utility
- Install the windows service

Windows services are installed in the system using a command line utility called installutil.exe. Each version of the .NET framework has its own version of the executable, and installing services manually can be a cumbersome process. Luckily, the .NET framework offers a C#-class called **ManagedInstaller-Class**, which can be used to install the service from code. One way to install a service from code can be done as shown below:

```
ManagedInstallerClass.InstallHelper(new string[]
{Assembly.GetExecutingAssembly().Location });
```

The installer can be found in the System.Configuration.Install namespace. It expects a path to the installable service executable, which can be retrieved with *GetExecutingAssembly()*-function. The service can be later installed in the same way by adding the prefix “/u” before the assembly path.

5.6 Encountered problems

Overall, the thesis study advanced and finished smoothly. Minor problems were encountered, which were caused by unforeseen circumstances.

5.6.1 MSSQL Combability with commanding system

When the thesis was in its early stages, the commissioner was investigating how they could make their IdM system to communicate with MSSQL databases. They could make it compatible, but it would require installing 3rd party software for their existing systems. The lead programmer of the system decided that they did not want to run such software in production environment, so the engine had to be changed from MSSQL to MySQL.

Luckily, an extension was available for Visual Studio what would enable the project to use almost the same commands as MSSQL to communicate with MySQL databases. As the MSSQL version of the program was almost complete, support for MSSQL databases was left in the program, in case it would ever be needed. MySQL Support was added to the program, and its database would be integrated into the IdM system’s database.

5.6.2 PowerShell memory leak

When the program was almost complete and in its test phases, it encountered a major problem. It would steadily and increasingly consume more memory from the host computer until it would run out and crash.

After it was discovered, the source code was carefully examined for memory leaks. Strangely, nothing was found that would even make it possible for leaks to occur. However, the source of the leak was pinpointed to the PowerShell commands what used to Import PS-Session-command.

After testing the code in a controlled environment, it became clear the Import PS-Session command was indeed leaking large amounts of memory, almost 100mb each time it was used. The memory leak can be seen in the following figure:

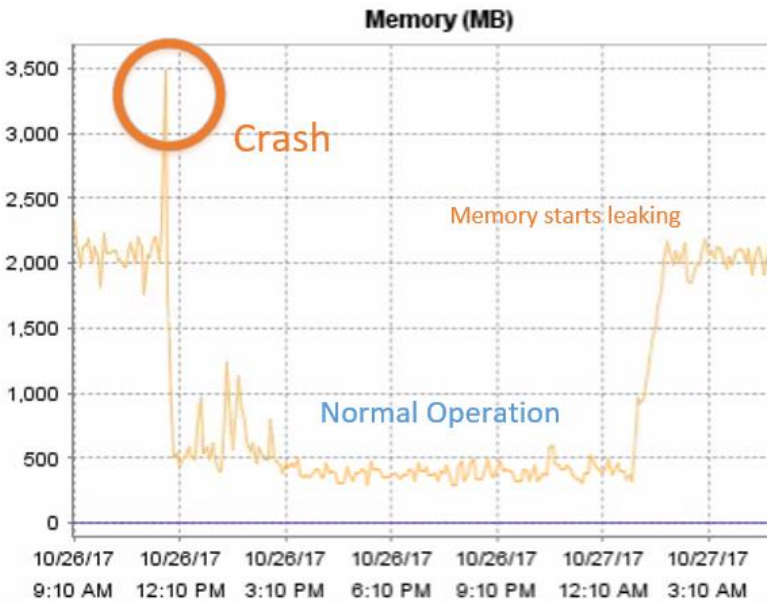

Figure 8 - Graph of the server's memory usage

Luckily, the leak was confirmed by the PowerShell Development team in GitHub (Memory leak in Import-PSSession and Remove-Module). According to their roadmap, a fix for the leak was scheduled for version 6.1 which would be released sometime in the future.

Unfortunately, the thesis schedule could not wait for PowerShell developers to create a fix so a workaround was created. The program would gather all commands what used the mentioned command and execute them once a day at defined time. After executing them, and leaking memory, the program would restart itself to free the leaked memory.

6 TESTING

Before the product could be rolled into development environments, it would have to be tested in a controlled environment. During the thesis, the commissioner did not have a development environment setup, so the program was tested in commissioner's demonstration environment.

6.1 Test environments

For testing, the commissioner provided access to their demonstration environment. This environment was mostly used to demonstrate their products to potential customers and test system changes on small scale. The following figure shows how the test environment was set up:

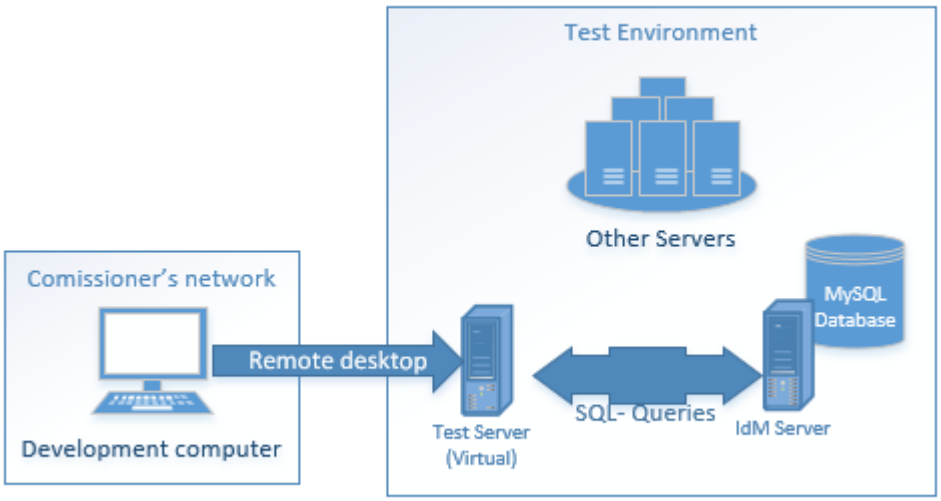

Figure 9 – Test Environment

Essentially, the demonstration environment was a small-scale copy of the commissioner's production environment, offering most of the services present in production. A single Windows server was set up for development and testing. The commissioner provided different kinds of tools for testing the program.

A server resource monitor was attached to the server what would monitor the program's uptime and resource usage. This provided valuable information, especially in discovering the PowerShell memory leak.

An access to the commissioner's firewalls was given to monitor network traffic between the server and other systems. The program's functionality was mostly tested using Visual Studio's build in debugger. The debugger offers verbose error messages and detailed resource usage in addition to the basic debugger functionality.

6.2 Testing 24 / 7 availability

The 24/7 availability of the program was tested with several different methods. First, an external monitor from PRTG network monitor was attached to the Windows service, which would periodically check if the service is available and healthy. It would log the results to its own internal database where they can be viewed as needed. It would also send out email alerts if the service is not running.

Secondly, the test environment would periodically send work orders to the program around the clock to see if it was able to fulfill them. Some queries would have errors in them on purpose to test out the program's error handling procedures.

Finally, the program's service would be left running for long periods of time to see if it would be able to work without errors and human interaction. After the latest version of the program was left running for a month without errors, it was time to test it in a small production environment.

7 DEPLOYMENT INTO PRODUCTION ENVIRONMENT

Before the newly created program would be able to be rolled out into existing environments, it would have to be tested in small-scale environments. Database and system backups were taken before rollout to prevent data loss.

7.1 Testing the deployment internally with the commissioner

First, the program was rolled out internally to the commissioner's IdM system, where it would serve approximately 50 users. The old O365-system was disabled and replaced with the new one.

Luckily, the program functioned and worked as expected. Still, the commissioner wanted to be sure that the program would function well for longer periods of time, so it was left running for several weeks. As no problems surfaced, it was time to test it in a larger environment.

7.2 Rollout into production

The next step would be to rollout the program to a larger environment consisting of approximately 9500 users. This would be its real test. The environment chosen for the initial rollout was the least critical – education network. The rollout was scheduled to take place after the normal working hours, so if problems would arise, they could be fixed before anyone would notice them.

One thing was different when running on a larger environment. The program did not work as speedily as it did in smaller environments. The Startup of the program took multiple minutes as it would update the database with thousands of new entries. The nightly maintenance period would take several hours to complete as the amount of data grew drastically. However, once the program was up and running after the initial startup slowdown, it would work very well. Several oversights were also discovered during the larger rollout. The program did not take the possibility of empty variables into account when processing data. This would cause the program to crash as no error handling was implemented. The problem was fixed as soon is was discovered.

Another oversight was the mismatch of data between the source system and the cloud. This would happen when the user data was manually changed in the cloud and not exported into the source systems. For the program, this manifested in the form of invaliding orders because the supplied username was not found in the cloud. Some research was done and soon it was discovered that the username modifications in the cloud were done in wrongly by user managers. The problematic entries were corrected, and soon the program was running smoothly without problems. After deployment, uptime monitors were attached to the program's Windows services and they would alert the administrators if the services ever stopped working.

8 PRODUCT AND SUMMARY

By the time of reporting the study, the created system has been running for 4 months in 10000+ user production environments. The new system achieved its desired functionality and requirements and has yet to encounter problems. It can execute its work effectively and reliably. It can be integrated into any existing system what can communicate with SQL databases. It offers several SQL tables filled with information, which can be accessed and presented in preferred way using SQL statements. The system saved the commissioner large amounts of working time because they no longer have to handle the problems created by the old system.

The thesis also provided the author and the commissioner with valuable new information and experience, which can be utilized as needed. Further deployment for the system is already planned with new functionalities and properties. It has sparked a side project with the commissioner where the system was being used as a base.

For the author, this thesis was an excellent opportunity to get familiar with Windows-system programming as the project involved working with many different parts of the Windows-system. The gained experience will be invaluable in the future.

REFERENCES

Chowdhury, K. 2017. Mastering Visual Studio 2017. Packt Publishing.

Clarke-Salt J. 2014. SQL Injection Attacks and Defense. Elsevier Science & Technology

Hihayak. Memory leak in Import-PSSession and Remove-Module. WWW-document. Available at: https://github.com/PowerShell/PowerShell/issues/5244 [Accessed 31 March 2018].

Hilyard, J. 2015 C# 6.0 Cookbook: Solutions for C# Developers. O'Reilly Media.

Jagger, J. 2007. Annotated C# Standard. Elsevier Science & Technology.

Lee, T. 2011. Windows Powershell 2.0 Bible. John Wiley & Sons, Incorporated

Martin M. 2017. PowerShell for Office 365: Automate Office 365 administrative tasks. Packt Publishing.

Templeman, J. 2002. Visual Studio .Net Black Book. Paraglyph Press.

Windows Service Documentation 2017. Introduction to Windows Service Applications. WWW document. Available at: https://docs.microsoft.com/en-us/dotnet/framework/windows-services/introduction-to-windows-service-applications [Accessed 31.3.2018]

LIST OF FIGURES