CTEC3451 Development Project

microCMDB

*A minimalist, cross-platform, and customisable Configuration Management Database*

Name: Arawn Davies

Supervisor: Dr Francois Siewe

Programme: Computer Science BSc.

De Montfort University

Contents

[Acknowledgements 3](#_Toc166377172)

[List of Figures 4](#_Toc166377173)

[List of Tables 5](#_Toc166377174)

[Introduction 6](#_Toc166377175)

[microCMDB Objectives & Functional Requirements 7](#_Toc166377176)

[microCMDB Project Overview 7](#_Toc166377177)

[microCMDB.CLI 8](#_Toc166377178)

[microCMDB.OS 8](#_Toc166377179)

[microCMDB.Web 10](#_Toc166377180)

[microCMDB.Test 11](#_Toc166377181)

[Development Lifecycle 13](#_Toc166377182)

[Phase 1: Requirement Analysis 14](#_Toc166377183)

[Phase 2: Design 14](#_Toc166377184)

[Initial entities and their properties: 16](#_Toc166377185)

[Phase 3: Development 16](#_Toc166377186)

[Finalised entities and their properties 17](#_Toc166377187)

[Phase 4: Testing & Continuous Integration 18](#_Toc166377188)

[Phase 5: Review and Feedback 20](#_Toc166377189)

[Phase 6: Deployment and Delivery 21](#_Toc166377190)

[Critical Analysis and Reflection 21](#_Toc166377191)

[Negative elements of the development project 21](#_Toc166377192)

[Positive elements of the development project 22](#_Toc166377193)

[Lessons to be learned for the future 22](#_Toc166377194)

[Conclusion 23](#_Toc166377195)

[Technical conclusion on the microCMDB system 23](#_Toc166377196)

[Non-technical conclusions on the overall project. 23](#_Toc166377197)

[Appendix 24](#_Toc166377198)

[Appendix A – Miscellaneous Info 24](#_Toc166377199)

[Appendix B – Source listing 25](#_Toc166377200)

[Entity 25](#_Toc166377201)

[ConfigItem 26](#_Toc166377202)

[Node 27](#_Toc166377203)

[Software 28](#_Toc166377204)

[Service 29](#_Toc166377205)

[Host 30](#_Toc166377206)

[NetworkUser 31](#_Toc166377207)

[Appendix C – Database seed data 32](#_Toc166377208)

[Appendix D – Test Suite 35](#_Toc166377209)

[Appendix E – Command Line Interface User Documentation 37](#_Toc166377210)

[References 38](#_Toc166377211)

# Acknowledgements

I would like to firstly thank my friends, family, peers and colleagues for their support and assistance over my years at De Montfort University, and for their understanding and patience when providing the much-appreciated support.

I would like to especially give thanks to Clinton Ingrams for his guidance, wisdom, and technical knowledge, and to Lauren Wysocki, for her mentorship, guidance and unwavering support have been a true lifeline over the last 4 years, helping me through the maze of university life.

I would finally like to dedicate this to my grandparents Keki and Thritty Mistry who recently passed away these past few months. Without their years of experience, love, knowledge, and expertise, I would not be where I am today. You are forever in my thoughts.

# List of Figures

[Figure 1: microCMDB CLI main entry screen displaying ConfigItems 8](#_Toc166585449)

[Figure 2: microCMDB Operating System running in VMWare Workstation Player 17 9](#_Toc166585450)

[Figure 3: microCMDB OS initialized as an empty database with zero entries 9](#_Toc166585451)

[Figure 4: microCMDB Web home page (signed out) 10](#_Toc166585452)

[Figure 5: microCMDB Web home page (signed in) 11](#_Toc166585453)

[Figure 6: microCMDB Web sign-in page 11](#_Toc166585454)

[Figure 7: microCMDB.Test results 12](#_Toc166585455)

[Figure 8: ConfigItems Create page 15](#_Toc166585456)

[Figure 9: ConfigItems index page 15](#_Toc166585457)

[Figure 10: Initial schema and entity relationship diagram 17](#_Toc166585458)

[Figure 11: Finalised schema and entity relationship diagram 19](#_Toc166585459)

# List of Tables

[Table 1: microCMDB main user objectives 7](#_Toc166585382)

[Table 2: Software entity 16](#_Toc166585383)

[Table 3: Service entity 16](#_Toc166585384)

[Table 4: Node entity 16](#_Toc166585385)

[Table 5: User entity 16](#_Toc166585386)

[Table 6: Host entity 16](#_Toc166585387)

[Table 7: Finalised ConfigItem entity 17](#_Toc166585388)

[Table 8: Finalised service entity 17](#_Toc166585389)

[Table 9: Finalised node entity 18](#_Toc166585390)

[Table 10: Finalised user entity 18](#_Toc166585391)

[Table 11: Finalised host entity 18](#_Toc166585392)

[Table 12: Finalised Software entity 18](#_Toc166585393)

[Table 13: List of required test cases for entity classes 20](#_Toc166585394)

# Introduction

As the world’s population increases, so has the number of large-scale enterprise organisations that provide services to digitally networked persons. To help deal with the unprecedented rate of expansion that IT is facing, businesses need to be prepared to effectively coordinate the deployment and management of IT infrastructure and assets. The overall practice of managing IT infrastructure, assets and services is known collectively as IT Service Management (ITSM).

Thankfully, the very machines that organisations may struggle to manage also make the archaic task of keeping paper inventory records much more accessible, efficient, and accessible. The databases that are specifically crafted for these purposes are known as Configuration Management Databases. A vast majority of large-scale businesses mainly operating in other sectors than the IT industry will often outsource ITSM needs due to the complex nature and layout of their infrastructure.

In my spare time, a hobby of mine has been restoring and using older computing technologies. Over these many years, I have collected far too many devices to accurately keep track of what software, services, user credentials and other configurations reside on each asset. A CMDB would be highly utilised in my home environment as this way it will allow me to efficiently monitor and manage these devices from one centralised location. For example, not all devices are Linux or Windows based, have modern operating systems or even TCP/IP protocol support, so a processor-agnostic system for preserving such records would be very beneficial and necessary in some cases.

The aim of the microCMDB project is to provide a robust, reliable, and lightweight alternative to SaaS cloud-based solutions. Any organization, whether it be private or public, which has a need for a streamlined, effective and open-source CMDB, but may not have the option of subscribing to or entering contract negotiations with a major CMDB provider (e.g. ServiceNow, IBM Tivoli etc.). microCMDB seeks to be a suitable, efficient, and minimal overhead solution to traditional CMDB needs without unnecessary bloat and overhead.

# microCMDB Objectives & Functional Requirements

The overall objective for the microCMDB project is to provide a robust, reliable, and lightweight configuration management database that is not tied to a specific interaction interface or specific platform architecture. microCMDB meets these objectives by providing three independent yet related interoperable platforms with a common interface and database backend.

Example users of the system will include network technicians diagnosing infrastructure issues, administrators, IT support who need to identify where a specific asset is deployed and to whom, and security/compliance teams who need an up-to-date layout of the infrastructure and networked devices.

The main objectives, functional user and system requirements include the following:

Functional Requirements:

* Easy to use
  + An intuitive and cleanly designed web interface is provided, every method and function of the system is fully documented, and easily accessible ‘help’ functionality is provided.
* Comprehensive
  + With an extensive and extendible database system, many different assets, their properties, and relationships can be established and maintained.
* Secure
  + By providing a ‘Just Enough OS’ operating environment with a cross-platform command line interface, the system can run with a very minimum amount of required memory or system resources.
* Efficient
  + microCMDB is highly responsive due to the low amount of overhead and fast object access.
* Accurate
  + microCMDB validates and corrects potentially invalid data types for various entity properties

|  |  |
| --- | --- |
| Responsiveness | The system needs to respond to all queries in a fast and timely manner |
| Security | Unauthorized access and amendments to the data should not be possible |
| Accuracy | The latest updates to any asset information needs to be made immediately accessible |
| Validation | Any updates made by a user need to be validated for incorrect data type, invalid range, and other erroneous entries. |
| Sanitation | Data entries need to be sanitized to avoid any potential vulnerabilities and attacks (including SQL injections or remote code execution). |

Table 1: microCMDB system data safety requirements

# 

# microCMDB Project Overview

One of the main objectives of this final year project is to provide a cross-platform system that implements a relational database model that represents a set of objects and entities that may exist in a complex network environment. For this, I knew the best development language and frameworks most applicable for this project would be C# and .NET Core, as I had the best knowledge of these through previous research and work carried out on my various software projects held publicly on GitHub and privately for Leicestershire County Council.

.NET Core provides a framework layer called ASP.NET Core, which allows a Web HTTP-based API or Model-View-Controller architecture providing CRUD functionality. Projects created using .NET Core can be ran under both Windows, macOS and Linux, and in containers such as Docker. To provide for even more simplicity, less overhead and lower system requirements, I also utilized the Cosmos Operating System project, which I first used over a decade ago, contributed to for many years and served as a core team member of the project and community. This made it abundantly clear that it would provide an excellent foundation for achieving the objectives of minimalism and cross-platform support.

The aim is to have a common code base that provides functionality for creating, reading, updating, and deleting various entities and the relationships between them. These models, classes and methods would then be referenced and compiled along with the frontend components that provide further interactivity and accessibility for the system.

## microCMDB.CLI

The CLI project includes the command line interface, a common runtime environment for interacting with the various entities and methods included in the system. This interface could either be ran as a remote .NET executable, a containerized daemon running alongside several other container instances, all cross-platform on a range of different IoT devices (Raspberry Pi, Intel NUC, etc.)

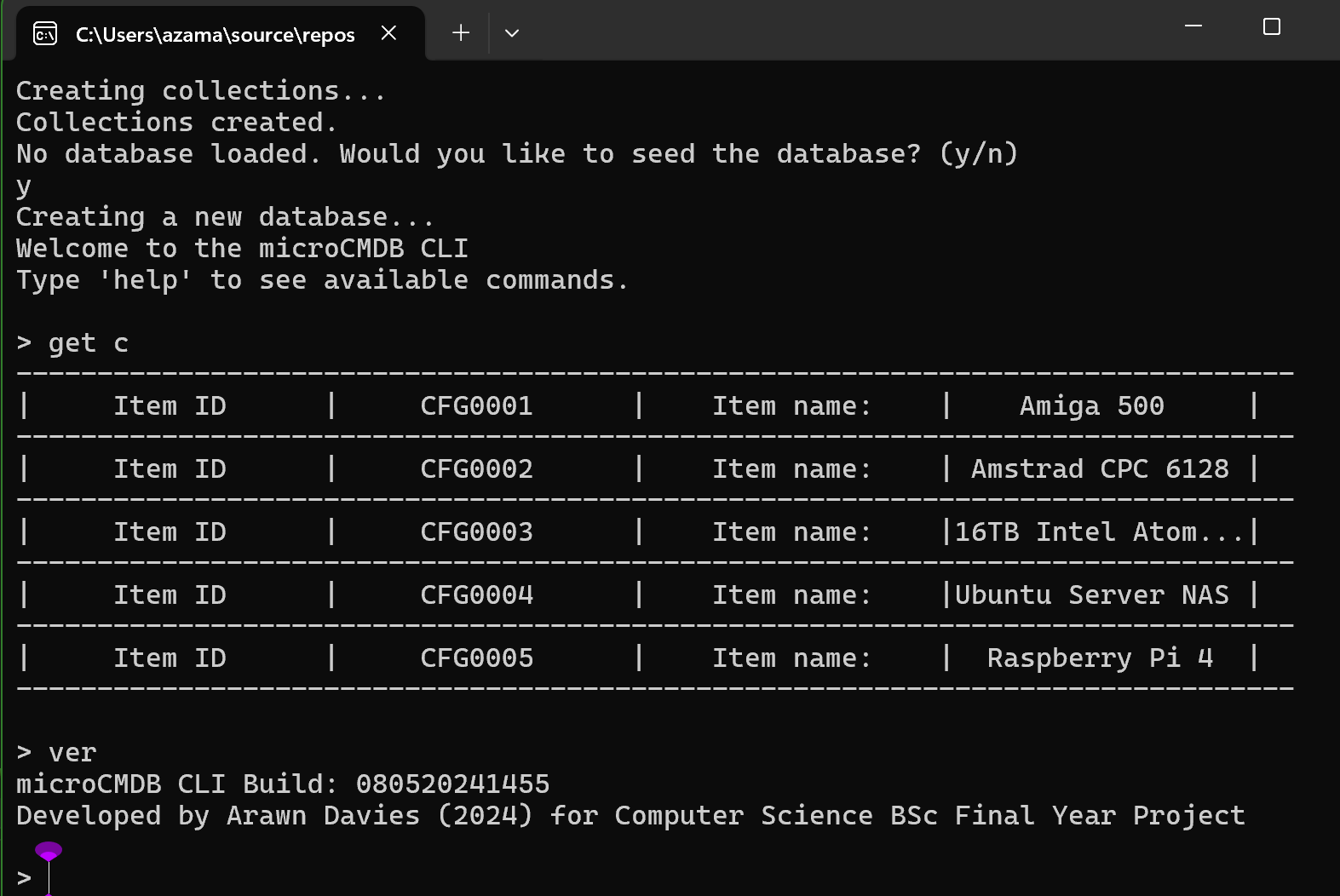


Figure 1: microCMDB CLI main entry screen displaying ConfigItems

To make the CLI source code very portable with a minimal set of runtime libraries, a tabular console formatting functionality was sourced from an answer on StackOverflow which allowed the CLI codebase to be cross-compiled to bare-metal with minimal required effort (McDonald, 2009).

## microCMDB.OS

This project contains the Hardware Abstraction Layer which allows microCMDB to be cross-compiled to a bare-metal x86 Assembly operating system kernel. This would allow microCMDB to be ran on a wide range of different platforms utilising the x86 processor architecture, including virtualization hosts VirtualBox, QEMU and VMware, to physical hardware hosts such as a thin client or other Small Form-Factor devices.

It uses the C# Open-Source Managed Operating System (Cosmos) as a .NET cross-compiler that allows the Windows/Mac/Linux CLI project to share a common codebase with the freestanding OS variant.

A screenshot of a computer program

Description automatically generated

Figure 2: microCMDB Operating System running in VMWare Workstation Player 17

Cosmos allows microCMDB to provide a ‘Just Enough Operating System’ experience, optimized for virtualization and heightened security. Using a custom non-POSIX backend with minimal device drivers allows the system to be less vulnerable against remote or arbitrary code execution.

A screenshot of a computer

Description automatically generated

Figure 3: microCMDB OS initialized as an empty database with zero entries

To introduce a mainline kernel to run underneath the microCMDB core would also introduce several issues such as potential cyberattacks and further maintenance and patches for the unused or unrecognised software components.

An example of where microCMDB CLI can be deployed in a minimalist secure environment would be with other operating systems which have also been developed using the Cosmos toolkit. One operating system which has been developed for the past decade includes the Medli operating system, also written by me. (Davies, A, 2016). While it has previously been known as ‘Chocolate’, the source code continues to be maintained to this day under the name ‘Medli’.

microCMDB can easily be integrated into the Medli OS’s source tree and compiled as part of the same project.

## microCMDB.Web

One of the main components of this project is the Web UI interface, currently built using an Entity Framework and SQLite backend with ASP.NET Core MVC. This allows the configuration management database administration tasks to be performed via a modular web interface. Existing CMDBs are increasingly switching to cloud web-based UIs to carry out most workflows and operations. The modular nature of this project allows for future expansion with community-contributed plugins or further features to communicate via several protocols (SSH terminal access to servers, generation of connection strings for one-click session starts).

The web user interface was a later addition to the development items as more focus was spent on making the database system accessible to other platforms via remote command line access. Introducing a web interface to the configuration management system introduces a significant amount more bloat as it depends on ASP.NET Core as well as the other frameworks and systems (EntityFramework and SQLite in this case).

The focus on the system is to reduce the amount of bloat that traditional CMDBs come pre-packaged with, but because of the unavoidable bloat (through unnecessary compilation of statically linked libraries). Development was

The web system allows a user to login via a username and password to securely access any authorized assets assigned to them.

A screen shot of a computer

Description automatically generated

Figure 4: microCMDB Web home page (signed out)

A screenshot of a computer

Description automatically generated

Figure 5: microCMDB Web home page (signed in)

A screenshot of a computer

Description automatically generated

Figure 6: microCMDB Web sign-in page

## microCMDB.Test

The fourth main component project of the microCMDB system is the test suite, comprised of classes and methods which will extensively test the system-specific methods. It is imperative that all methods in all namespaces which are interacted by the user are tested for validation and sanitation, hence resulting in this project being a key component for development of all three systems. For the project, MSTest was used as it came bundled with Visual Studio 2022 and the .NET SDK, resulting in more integrated testing tools and faster testing.

A screenshot of a computer

Description automatically generated

Figure 7: microCMDB.Test results

Below is a table of each ID tests devised during the initial design and planning stages, and their individual completion statuses:

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Test Type | Value | Result |
| ConfigItemID | Minimum | 0 | Y |
|  | Minimum – 1 | -1 | Y |
|  | Maximum | Int32.Max | Y |
|  | Maximum +1 | Int32.Max +1 | Y |
|  | Invalid | “Example” | Y |
|  |  |  |  |
| Node ID | Minimum | 0 | Y |
|  | Minimum – 1 | -1 | Y |
|  | Maximum | Int32.Max | Y |
|  | Maximum + 1 | Int32.Max +1 | Y |
|  | Invalid | “Example” | Y |
|  |  |  |  |
| User ID | Minimum | 0 | Y |
|  | Minimum – 1 | -1 | Y |
|  | Maximum | Int32.Max | Y |
|  | Maximum + 1 | Int32.Max +1 | Y |
|  | Invalid | “Example” | Y |
|  |  |  |  |
| Host ID | Minimum | 0 | Y |
|  | Minimum – 1 | -1 | Y |
|  | Maximum | Int32.Max | Y |
|  | Maximum + 1 | Int32.Max+1 | Y |
|  | Invalid | “” | Y |
|  |  |  |  |
| Service ID | Minimum |  | Y |
|  | Minimum – 1 |  | Y |
|  | Maximum | Int32.Max | Y |
|  | Maximum + 1 | Int32.Max | Y |
|  | Invalid |  | Y |
|  |  |  |  |
| Software ID | Minimum |  | Y |
|  | Minimum – 1 |  | Y |
|  | Maximum | Int32.Max | Y |
|  | Maximum + 1 | Int32.Max | Y |
|  | Invalid |  | Y |

# Development Lifecycle

Development of the microCMDB project will be done in a series of steps.

Firstly, is the Requirements Analysis stage. This involves the collection, examination, and investigation of a system to be implemented that provides a solution to the problem. Secondly, there will be the initial Design work of the system and its underlying system architecture and frameworks. It is of paramount importance that the design work and accompanying documentation is of good detail before developers can begin work on an initial implementation. During this stage, the initial models are created and scaffolded methods and functions are defined.

Once a sufficiently operational system has been implemented, initial Testing work is carried out. It is during this process where analysis of the system is carried out and the degree to which it meets the requirements, as defined during the second stage, is measured using metrics and testing frameworks.

Before the system is fully operational and live, there is the process of Delivery and Deployment. This involves the installation of the system and setup of permissions in its production environment, i.e. prepare for usage in real-world business operations. Lastly, is the Review and Feedback stage, where user input and customer feedback are collected from the end-user and used to further improve the systems security, functionality, performance, and any other issues which may have arisen during design and development.

These must be completed in the order specified for maximum operational and developmental performance, but it can be expected in this scenario to be revisiting steps while development continues.

At the start of the project, it was apparent that a significant amount of time would be necessary to develop this project, so baring this in mind in addition with many other deadlines and deliverables, it was important that an appropriate software development methodology was needed. During the placement spent with Leicestershire County Council, the Agile methodology was used for most of the work so that was chosen as it was the most familiar methodology and flexible with the ever-changing and loose requirements.

I believe that the Agile development system is the best applicable methodology used in this project, due to the exploratory nature of this project and the evolving requirements. However, because this project has a clear start and end with defined deadlines and clear requirements, it makes the more traditional Waterfall methodology equally as applicable.

If microCMDB to be developed professionally, full-time by a team of dedicated developers and other project members, then issues or errors that may arise over the course of development would likely be addressed sooner, due to more eyes on the work being carried out. However, as microCMDB was developed for a final year project in an academic setting, with one lead developer (myself), it would not be feasible to work on this project full time. Considering the nature of the final year with exams, coursework and other personal commitments which require attention too, it’s not clear cut as to which methodology works best for a specific project. The best applicable development methodology depends on various aspects: the student developing the project; supervisors and academic staff who expect guidelines to be followed; and the nature of the development project itself.

As opposed to regular sprint meetings with several system developers, a group of one-on-one meetings were held with myself and Francois (my project supervisor), which allowed me to regularly ensure that this project met the academic guidelines and objectives, and that any tasks were nearing completion and ready for the approaching deadlines.

## Phase 1: Requirement Analysis

During traditional software development, the analysis of requirements focuses on the needs, expectations, and objectives of the stakeholders, whether that’s the end-users, product owners or developers and other persons involved. For the microCMDB project specifically, the stakeholders would be my supervisor, advisors, and peers who provide peer reviews and feedback, and the academic staff in charge of the final year projects of students and submissions.

Unlike typical software development projects, the objectives, functionality, and requirements for microCMDB are self-identified and documented as opposed to usually either provided solely or contributed towards by the client and/or end-user. As a result, focus can be dedicated to compliance of the academic guidelines instead of stricter and more complex restrictions dictated by a third party.

For the microCMDB Command Line Interface, which makes up the core methods and functionality of the system, there were several requirements which were identified at the start:

## Phase 2: Design

The tabular approach to the application’s database influences a significant amount of the design choices and underlying frameworks and technologies. As the aim for the system is focused on cross-platform portability and minimalism, a simple query-based database with a low amount of required overhead was chosen. As a lot of development experience has been in low-level and cross-platform systems development has been done in the C# and .NET runtime, this was chosen as the primary target.

The programming language and development environment selected was .NET SDK (formerly .NET Core) with ASP.NET Core as a web frontend. EntityFramework Core was also employed as an ORM (object-relation mapper) and SQLite as the RDBMS backend. This choice facilitated targeting a wide range of platforms to target, including Linux, macOS, and Windows on all commonly used CPU architectures, and potentially in the future, the bare metal x86\_32/64 platform through the Cosmos User Kit, which translates .NET Common Intermediate Language into 32-bit x86 assembler instructions or a 64-bit UEFI binary (NativeAOT, 2022).

The Cosmos project was used as an underlying operating system toolkit as it efficiently cross-compiles .NET executables from the Common-Intermediate-Library to native x86 bare-metal assembly (Cosmos, ‘*IL2CPU’, 2024)*

As would be the case for most database systems, it was vital that the fundamental methods were implemented and designed. These would be the Create, Read, Update and Delete methods that allow the user to interact with the main database tables.

A screenshot of a computer

Description automatically generated

Figure 8: ConfigItems Create page

Several draft database designs were created to allow for a skeleton bare-bones system, only providing for a basic object model for each of the main entities. The five main initial entities used for development included those for ConfigItems, Nodes, Hosts, Users, Services and Software.

A screenshot of a computer

Description automatically generated

Figure 9: ConfigItems index page

Visual Studio 2022 provides easy-to-use and powerful scaffolding templates which provided a basis on which to build up the microCMDB web user interface. Once the tables were designed and controllers and views were generated, it was difficult to revert any changes or make any adjustments to the views and properties displayed in the UI. This resulted in a significant number of refactoring and reverting of changes, reinforcing the preference for command line logic development.

### Initial entities and their properties:

|  |  |
| --- | --- |
| Software | |
| Integer (Primary Key) | Sw\_ID |
| VARCHAR | Sw\_Name |
| VARCHAR | Sw\_Pub |
| VARCHAR | Sw\_Licence |
| VARCHAR | Sw\_InstPath |

Table 2: Software entity

|  |  |
| --- | --- |
| Service | |
| Primary key, Integer | Service\_ID |
| VARCHAR | Service\_Protocol |
| VARCHAR | Service\_URL |
| Integer | Service\_Port |

Table 3: Service entity

|  |  |
| --- | --- |
| Node | |
| Primary key, Integer | Node\_ID |
| VARCHAR | Node\_Arch |
| Integer | Node\_RAM |
| Integer | Node\_Storage |
| VARCHAR | Node\_OS |

Table 4: Node entity

|  |  |
| --- | --- |
| User |  |
| Primary key, Integer | User\_ID |
| VARCHAR | User\_name |
| VARCHAR | User\_pass |

Table 5: User entity

|  |  |
| --- | --- |
| Host | |
| Primary key, integer | Host\_ID |
| Foreign key (Service) | Services |
| Foreign key (Software) | Software |
| Foreign key (User) | User\_ID |
| Foreign key (Node) | Node\_ID |

Table 6: Host entity

## Phase 3: Development

Provide a detailed overview of each stage of the development lifecycle, such as requirements analysis, design, implementation, testing, and deployment. Explain how validation and verification techniques were applied at each stage to ensure the quality and correctness of the system.

During the development stage, it became evident that most .NET standard I/O formatting methods and projects are incompatible with the microCMDB.OS abstraction layer, due to extra non-ported namespaces and non-standard code. As a result, further time was taken to implement a tabular console I/O class of methods used for the representation of assets and entities.

While the design decisions made in the last phase were sufficient to allow development to begin, there were numerous issues that arose out of the initial entity models. It was clear that a relationship between each of the entities needed to be clearly defined, however, with the evolving list of objectives and requirements, they soon became ineffective and too time consuming to accurately represent the assets in the database.

A diagram of a computer

Description automatically generated

Figure 10: Initial schema and entity relationship diagram

Redefining the entities and the relationships between them allowed minimizing circular dependencies and redundant data which likely would have resulted in higher memory usage.

### Finalised entities and their properties

|  |  |
| --- | --- |
| ConfigItem | |
| Integer (Primary Key) | CFG\_ID |
| DateTime | PurchaseDate |
| VARCHAR | DeployLocation |
| CINodeMapping | CINodeMapping |

Table 7: Finalised ConfigItem entity

|  |  |
| --- | --- |
| Service | |
| Primary key, Integer | Service\_ID |
| Integer | Protocol |
| Integer | Host\_ID |
| HostServiceMapping | HostServiceMapping |

Table 8: Finalised service entity

|  |  |
| --- | --- |
| Node | |
| Primary key, Integer | Node\_ID |
| VARCHAR | Node\_Arch |
| Integer | Node\_RAM |
| Integer | Node\_Storage |
| VARCHAR | Node\_OS |

Table 9: Finalised node entity

|  |  |
| --- | --- |
| NetworkUser |  |
| Primary key, Integer | User\_ID |
| VARCHAR | User\_name |
| VARCHAR | User\_pass |

Table 10: Finalised user entity

|  |  |
| --- | --- |
| Host | |
| Primary key, integer | Host\_ID |
| Foreign key (Service) | Services |
| Foreign key (Software) | Software |
| Foreign key (User) | User\_ID |
| Foreign key (Node) | Node\_ID |

Table 11: Finalised host entity

|  |  |
| --- | --- |
| Software | |
| Primary Key, integer | Sw\_ID |
| VARCHAR | Sw\_Version |
| VARCHAR | Sw\_Developer |
| VARCHAR | Sw\_LicenceType |
| ICollection<SoftwareInstallation> | Installations |

Table 12: Finalised Software entity

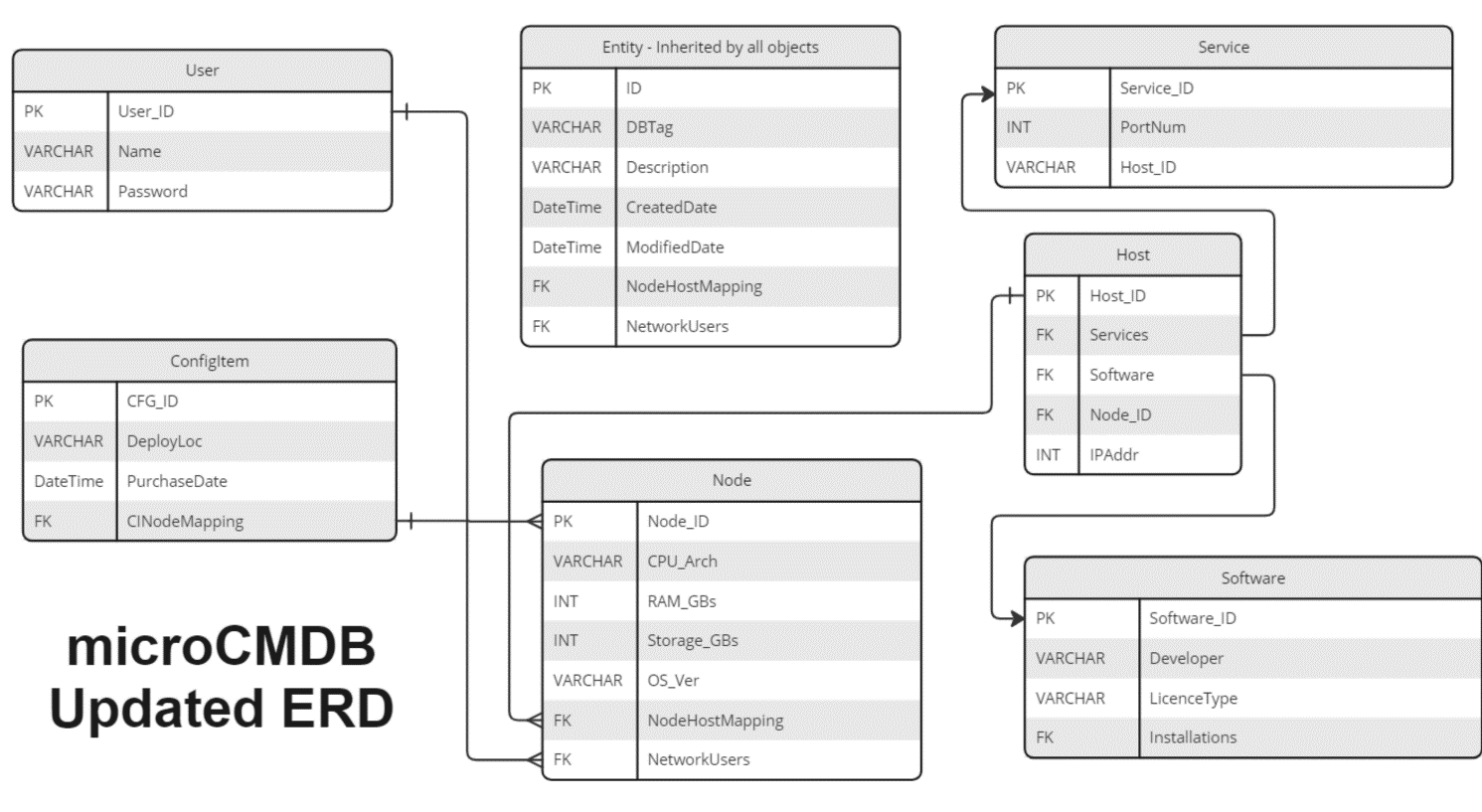


Figure 11: Finalised schema and entity relationship diagram

## Phase 4: Testing & Continuous Integration

Appveyor and C#’s MSTest unit testing suite were used to provide continuous testing and build feedback. As opposed to cloning (the process of downloading the latest source code from the repository), and compiling the configuration management database system, simply to check the build status of a single component, one could visit the Appveyor or GitHub project site and view the latest commit’s build status from there. Docker was also chosen to provide containerization support of the microCMDB system, further achieving one of the main objectives cross-platform support.

During the earlier stages of the project, it was obvious that the type of tests that will be required depended on the data types and processing methods that would capture input from a user. The extra tests that were developed included the parsing of integers from strings collected from user input, valid ranges for types such as DateTime (e.g., PurchaseDate cannot be in the future), and ensuring that alpha and symbol characters were not parsed as part of numeric data fields.

|  |  |
| --- | --- |
| ConfigItem | PurchaseDate\_SetValidValue\_ShouldSetPurchaseDate  PurchaseDate\_SetInvalidValue\_ShouldThrowException  DeployLoc\_SetValidValue\_ShouldSetDeployLoc  DeployLoc\_SetNullValue\_ShouldSetDeployLocToEmptyString  PrintInfo\_ShouldPrintCorrectInfo  ExportObject\_ShouldReturnCorrectString |
| Host | IPaddr\_SetValidValue\_ShouldSetIPaddr  IPaddr\_SetNullValue\_ShouldSetIPaddrToNull  PrintInfo\_ShouldPrintCorrectInfo  ExportObject\_ShouldReturnCorrectString |
| NetworkUser | Username\_SetValidValue\_ShouldSetUsername  Username\_SetNullValue\_ShouldThrowException  Email\_SetValidValue\_ShouldSetEmail  Email\_SetNullValue\_ShouldSetEmailToNull  Firstname\_SetValidValue\_ShouldSetFirstname  Firstname\_SetNullValue\_ShouldSetFirstnameToNull  Lastname\_SetValidValue\_ShouldSetLastname  Lastname\_SetNullValue\_ShouldSetLastnameToNull  PrintInfo\_ShouldPrintCorrectInfo  ExportObject\_ShouldReturnCorrectString |
| Service | Protocol\_SetValidValue\_ShouldSetProtocol  Protocol\_SetInvalidValue\_ShouldThrowException  PortNum\_SetValidValue\_ShouldSetPortNum  PortNum\_SetInvalidValue\_ShouldThrowException  URL\_ShouldReturnCorrectURL  PrintInfo\_ShouldPrintCorrectInfo  ExportObject\_ShouldReturnCorrectString |
| Software | Version\_SetValidValue\_ShouldSetVersion  Version\_SetNullValue\_ShouldThrowException  Developer\_SetValidValue\_ShouldSetDeveloper  Developer\_SetNullValue\_ShouldSetDeveloperToNull  LicenseType\_SetValidValue\_ShouldSetLicenseType  LicenseType\_SetNullValue\_ShouldSetLicenseTypeToNull  PrintInfo\_ShouldPrintCorrectInfo  ExportObject\_ShouldReturnCorrectString |

Table 13: List of required test cases for entity classes

## Phase 5: Review and Feedback

As GitHub was used as a hosting service to store the source code, it allows for advanced bug tracking and access for contributors to the project. This would likely be a highly utilised tool as GitHub is free for open-source, academic and personal use.

Using a cloud-hosted and freely available site to store the source code allows the source code to be easily downloaded and built on any development platform. While a professional non-academic project would depend on methods such as data collection and user analytics, as well as reviews and feedback, microCMDB currently has no deployments or current user. This could be remedied through GitHub’s issues tracker.

## Phase 6: Deployment and Delivery

The deployment and delivery of the microCMDB project differs depending on which platform it is to run on and in what form. When the system was deployed in the developer’s home environment, microCMDB was installed in a Docker container running on Ubuntu Server.

The technical specialist who is to be deploying the microCMDB system has the option of various virtualization methods: from running the bare-metal OS in a hypervisor (such as Oracle VirtualBox or VMWare Workstation), or by running the CLI on another virtualized system (Linux, Windows, macOS).

microCMDB source code is available on GitHub can be downloaded easily from <https://github.com/Arawn-Davies/microcmdb>. This will aid with the deployment and delivery of the system as it avoids the need for another physical medium storing the executables and source.

# Critical Analysis and Reflection

As with any project that is undertaken in this academic setting, there are many positive and negative aspects to all elements that the system comprises of. In this section, I will discuss the negative and positives regarding the completion and functionality of the technical system, as

## Negative elements of the development project

Regarding the technical elements of the system itself, there are several issues that have the potential prevent a successful deployment in its current state. These include scalability, performance, and security. The system does indeed meet the core functional requirements, however if an organization were to implement the microCMDB system on-premises in a secure environment, then the biggest flaws would be the lack of encryption and credential-based authentication. Performance-wise, the lack of multi-user access and efficient API access results in further dependence on the security features of an underlying operating system.

When the project was initially started, there were several issues which were foreseen and taken into consideration. At the beginning when the first deliverable and project contract was written, there were numerous challenges and disruptions to normal development abilities. Some of the challenges have been very personal, including a complete electrical rewiring of my home address (which is where I mainly reside) which took longer than expected, existing mental health issues, and the loss of both of my maternal grandparents in the first half of 2024.

While the above events were in the present and made up for the most amount of disruption, there were a few historic issues that further complicated the present difficulties. This included my dyspraxia which affects various aspects such as concentration, time management and social communication. These factors all played a significant part in timeline task allocation, and resulted in a lower performance and output than what would otherwise be expected.

## Positive elements of the development project

There are several areas which went very well considering the severe impact personal challenges had on the development of the system. As the initial system and concepts were first imagined in the placement prior to this final year, it was beneficial to have a head start at getting the initial plans drafted and research carried out.

One of the main objectives that were reached was cross-platform portability where a single codebase could be used across different architectures and platforms. While now the bare-metal operating system only runs on x86 platforms, the .NET compiled executable can be ran on any widely used computer of today (Microsoft, 2024).

Having previously worked in similar technical environments before, I already had sufficient knowledge with the abilities and limitations of the frameworks used. This resulted in minimal effort needed to create a strong codebase that meets the functional requirements of the system. While it isn’t always possible for developers to write solely in the language or use tools of their choosing, it is very important that one’s knowledge and familiarity of programming paradigms and tools is pushed where possible and appropriate.

## Lessons to be learned for the future

When reviewing the progress and completion of the overall project and the functional components, it would have been better to focus on a minimal framework that implements the main requirements, and then built on top of that. As the web user interface was a lower priority than the cross-platform command line interface, it would have been more beneficial on implementing a secure and robust system that could then be extended to allow communication via an API. As a developer I am much more comfortable with logic-based processing and low-level systems development as opposed to user interface, human interaction, and graphical layout development, so it would be wise to utilise the expertise of other parties when implementing these secondary requirements.

In retrospect, the opportunity to have developed in a programming environment of my choosing has dramatically aided in the work output and performance for the project. Had a different programming language been chosen (for example, I am unfamiliar with ‘WebAssembly’ and ‘Ruby on Rails’), then the completion of the project could potentially have been much lower.

It is very important to not forget that most of the disruptions and events over the past year were unforeseen and considerably life changing. While the best was done wherever possible, the full capabilities and rate of progress could have been reached, had more help and assistance been accessible. Had the extent of the disruption been known at the start, then it is likely that this entire project would have been postponed to a later stage when possible. More time was needed to have been spent on planning for any setbacks and emergencies.

In the future, I would dedicate more time in implementing a stable and efficient plan of action accounting for things that could have been avoided. This would have provided a much more stable working environment, making the project more reflective of my skills, knowledge, and abilities.

# Conclusion

## Technical conclusion on the microCMDB system

The microCMDB project has been a significant and tasking project to carry out due to the pressures of the academic project, and other difficulties faced throughout the process, but nonetheless it has been successful at meeting the core requirements and objectives set out at the beginning of the planning process.

Throughout the planning and development process, there were several key factors that were taken into consideration at each stage. These were simplicity, portability, and extensibility. Utilizing .NET and C# as a development framework and language, a successful and powerful command line interface was implemented, and the necessary models were created to manage configuration items, logical software assets, and other physical assets which are present on advanced networking and communication infrastructures.

Moving forward, it would be excellent to see microCMDB implement further additional features, such as remote connection utilities to access remote devices, status checking for hosts and assets which services may depend on, and better storage and data handling methods to ensure streamlined and error-free data migration between the three main systems.

## Non-technical conclusions on the overall project.

I believe had the past year been less eventful, then the project could have been much more successful than it already is. I would have made more use of projects I’ve created or contributed to that do provide various methods of meeting the functional requirements. In years prior, there has been much less pressure felt when I’ve worked on open-source hobbyist projects in my spare time, while also helping me sharpen my technical skills for any future career plans.

Even if everything hadn’t gone exactly as planned, every aspect of this project has been an enormous learning experience and opportunity to research and develop a useful tool that has many uses in my own network and potentially other organisations too. Developing microCMDB has allowed me to work on a project that I enjoy, linking to my own hobbies and other enthusiasts who work on a vast range of different platforms and technical environments.

After dedicating many years of study, research, and work on various aspects of computer science and different projects, developing microCMDB has provided me an opportunity to demonstrate everything I’ve learnt at De Montfort University. I would have very much liked to have completed the microCMDB project to its fullest completion, but it was important that feature creep did not happen, as this would have resulted in higher-priority tasks and features having lower-quality implementations.

# Appendix

## Appendix A – Miscellaneous Info

* Cosmos C# filesystem initialization code: microCMDB.OS has no underlying operating system or runtime other than the basic x86 functions provided by the BIOS and compiler. The OS must create a file allocation table (FAT) on the primary hard disk, which will then store the configuration management databases’ assets and other related items.
* Cosmos provides an example disk image (Filesystem.vmdk) which resides in “AppData\Roaming\Cosmos User Kit\Build\VMware\Workstation” in default installations.
* As the database already provides initial seed data, it is not strictly necessary to use physical storage media to store data.
* microCMDB GitHub repository site <https://github.com/Arawn-Davies/microcmdb>
* Cosmos GitHub repository site <https://github.com/CosmosOS/Cosmos>
* Cosmos Project Website <https://gocosmos.org/>
* Microsoft .NET SDK 6.0 Downloads. <https://dotnet.microsoft.com/en-us/download/dotnet/6.0>
* The system requirements for compiling the main system components depends on the system being deployed:
* Top-level dependencies:
  + .NET SDK
    - While all of the projects are built to support a minimum number of methods in the SDK, the version used depends on which system to be deployed
  + Visual Studio 2022 or latest supported version.
    - Community/Professional and Enterprise are all supported.
* microCMDB.OS:
  + VMWare Workstation
  + .NET SDK 6.0
  + VMWare Workstation Player
* microCMDB.Web:
  + .NET SDK 7.0
* microCMDB.CLI :
  + .NET SDK 6.0
* The Cosmos Developer Kit is a development build that includes the latest changes and features. It also depends on the following:
  + InnoSetup
  + Microsoft Visual C++ 2010 Redistributable
  + Visual Studio 2022 Workload: .NET Desktop Development

## Appendix B – Source listing

### Entity

// This abstract class is inherited by sub-classes to represent entities in the DB.

public class Entity() {

// Abstract property to be implemented by derived classes

// provides a unique prefix for the entity type

public abstract string DbTagPrefix { get; }

// The name of the entity

public virtual string Name { get; set; } = string.Empty;

// A unique identifier for the entity,

// generated based on the DbTagPrefix and a counter

public string DbTag { get; set; } = string.Empty;

// A description of the entity, used to provide additional information e.g. notes, known issues, etc.

public string Description { get; set; } = string.Empty;

// Date/Time the entity was created

public DateTime CreatedDate { get; set; } = DateTime.Now;

// Date/Time the entity was last modified.

public DateTime ModifiedDate { get; set; } = DateTime.Now;

// Dictionary to keep track of the number of entities,

// created for each specific type of entity.

private static Dictionary<string, int> DbEntityCounter =

new Dictionary<string, int>();

// Constructor for all microCMDB entities

public Entity() {

// If the specific prefix’s collection does not contain any values, set the

// initial prefix’s counter to one.

if (!DbEntityCounter.ContainsKey(DbTagPrefix)) {

DbEntityCounter[DbTagPrefix] = 1; }

// Generate the unique identifier based on the prefix and the counter

int nextId = DbEntityCounter[DbTagPrefix]++;

Id = nextId;

// Set the DbTag property to the generated identifier,

DbTag = DbTagPrefix + Id.ToString("D4");

// Add the new entity to the collection.

Db.CurrentDbContext.Entities.Add(this); }

// Using the custom table methods, print the top-level entity information:

public virtual void PrintInfo() {

Table.PrintRow("ID", DbTag);

Table.PrintRow("Name", Name);

Table.PrintRow("Notes", Description);

Table.PrintRow("Last updated", ModifiedDate.ToString());

Table.PrintRow("Created", CreatedDate.ToString()); }

// Abstract method that inherits PrintInfo but adds entity-specific properties:

public abstract string ExportObject(); }

### ConfigItem

public class ConfigItem : Entity {

public ConfigItem() : base()

{ Db.CurrentDbContext.ConfigItems.Add(this); }

/// <summary>

/// Overrides the DbTagPrefix property in the Entity class to set the prefix for ConfigItem objects

/// </summary>

public override string DbTagPrefix => "CFG";

/// <summary>

/// An optional field for the date the ConfigItem was purchased

/// </summary>

[Display(Name = "Date purchased")]

public DateTime PurchaseDate { get; set; }

/// <summary>

/// An optional field for the deployment location of the ConfigItem (e.g. datacenter, office, mobile etc.)

/// </summary>

[Display(Name = "Deployment location")]

public string DeployLoc{ get; set; } = string.Empty;

public CINodeMapping CINodeMapping { get; set; }

/// <summary>

/// Print the ConfigItem's information to the console

/// </summary>

public override void PrintInfo() {

base.PrintInfo();

Table.PrintRow("Purchase date:", PurchaseDate.ToString());

Table.PrintRow("Deployed:", DeployLoc); }

// Overrides the base entity's ExportObject string to include the custom ConfigItem properties.

public override string ExportObject() {

return $"{DbTag},{Name},{Description},{CreatedDate},

{ModifiedDate},{PurchaseDate},{DeployLoc}"; }

// ConfigItem object-specific constructor with parameters

public ConfigItem(string \_dbTag, string \_name, string \_description,

DateTime \_dt, string \_deployLoc) {

DbTag = \_dbTag;

Name = \_name;

Description = \_description;

PurchaseDate = \_dt;

DeployLoc = \_deployLoc; } }

### Node

public class Node : Entity {

public Node() : base() {

if (CINodeMapping != null) {

ConfigItemID = CINodeMapping.ConfigItemID; }

else { ConfigItemID = 0; }

Db.CurrentDbContext.Nodes.Add(this); }

public override string DbTagPrefix => "NOD";

// A required field for the operating system version

[Required]

[Display(Name = "Operating System")]

public string OS\_Version { get; set; } = string.Empty;

// A required field for the CPU architechture of the node

[Required]

[Display(Name = "CPU Architechture")]

public string CPU\_Arch { get; set; } = string.Empty;

// A required field for the amount of RAM. This is a double to allow for decimal

values as the default unit is megabytes.

[Display(Name = "RAM")]

public double RAM { get; set; }

// A required field for the amount of storage. This is a double to allow for decimal

values as the default unit is gigabytes.

[Display(Name = "Storage")]

public double Storage { get; set; }

// A required field for which ConfigItem the node is associated with

public int ConfigItemID { get; set; }

[Display(Name = "Linked Configuration Item")]

public CINodeMapping? CINodeMapping { get; set; }

public NodeHostMapping? NodeHostMapping { get; set; }

[Display(Name = "Installed Software")]

public ICollection<SoftwareInstallation> InstalledSoftware { get; set; } = new

List<SoftwareInstallation>();

public ICollection<NetworkUserMapping>? NetworkUsers { get; set; }

public override void PrintInfo() {

base.PrintInfo();

Table.PrintRow("OS Version:", OS\_Version);

Table.PrintRow("CPU Architecture:", CPU\_Arch);

if (RAM != null || RAM != 0) { Table.PrintRow("RAM", RAM.ToString()); } else {

Table.PrintRow("RAM", "N/A"); }

if (Storage != null || Storage != 0) { Table.PrintRow("Storage:",

Storage.ToString()); } else { Table.PrintRow("Storage", "N/A"); } }

public void PrintSoftware() {

Table.PrintLine();

Table.PrintRow("Installed Software:");

foreach (var software in InstalledSoftware) {

software.PrintInfo(); }

Table.PrintLine(); }

public override string ExportObject() {

// Return a string representation of the ConfigItem object

// containing every property

return $"{DbTag},{Name},{Description},{CreatedDate},{ModifiedDate},

{OS\_Version},{CPU\_Arch},{RAM},{Storage},{ConfigItemID}"; } }

### Software

public class Software : Entity

{

public Software() : base()

{

Db.CurrentDbContext.Software.Add(this);

}

public override string DbTagPrefix => "STW";

// A required field for the specific version of the software that is installed.

[Required]

[Display(Name = "Software Version")]

public string Version { get; set; } = string.Empty;

// An optional field for the developer or publisher of the software

[Display(Name = "Developer")]

public string? Developer { get; set; }

// An optional field for the type of license

[Display(Name = "License Type")]

public string? LicenseType { get; set; }

[Display(Name = "Installed On")]

public ICollection<SoftwareInstallation> Installations { get; set; } =

new List<SoftwareInstallation>();

public override void PrintInfo() {

base.PrintInfo();

Table.PrintRow("Version:", Version);

if (!String.IsNullOrEmpty(Developer)) {

Table.PrintRow("Developer:", Developer); }

if (!String.IsNullOrEmpty(LicenseType)) {

Table.PrintRow("License Type:", LicenseType); }

if (Installations.Count > 0) {

Table.PrintLine();

Table.PrintRow("Installed On:");

Table.PrintRow("Entity tag:", "Name:");

foreach (var installation in Installations) { Table.PrintRow(installation.Node.DbTag, installation.Node.Name); } }

}

public override string ExportObject() {

// Return a string representation of the Software object

// containing every property

return $"{DbTag},{Name},{Version},{Developer},{LicenseType}"; } }

### Service

public class Service : Entity {

public Service() : base() { Db.CurrentDbContext.Services.Add(this); }

// This method could take multiple forms (Dictionary<int,string>, switch/case,

// but the current implementation is a simple if/else statement to return the

// correct URI prefix based on the port number. It could be improved by allowing

// methods for adding custom service types.

private string URI\_PREFIX { Get { return Protocol.ToLower() + "://"; } }

public override string DbTagPrefix => "SVC";

[Required]

[Display(Name = "Service Protocol")]

public string Protocol {

Get {

if (PortNum == 80) { return "HTTP"; }

else if (PortNum == 443) { return "HTTPS"; }

else if (PortNum == 22) { return "SSH"; }

else if (PortNum == 21) { return "FTP"; }

else if (PortNum == 25) { return "SMTP"; }

else if (PortNum == 25565) { return "Minecraft"; }

else { return "N/A"; } } }

[Display(Name = "Location")]

public string URL() {

// Return the URL based on the associated host's IP address and the port number

return URI\_PREFIX + HostServiceMapping?.Host?.IPaddr + ":" + PortNum; }

[Required]

[Display(Name = "Port Number")]

public int PortNum{ get; set; }

public int HostId { get; set; }

public HostServiceMapping? HostServiceMapping { get; set; }

public override void PrintInfo() {

base.PrintInfo();

Table.PrintRow("Port", PortNum.ToString());

Table.PrintRow("Protocol:", Protocol);

Table.PrintRow("URL", URL()); }

public override string ExportObject() {

// Return a string representation of the Service object containing every property

return $"{DbTag},{Name},{Description},{CreatedDate},{ModifiedDate},

{PortNum},{Protocol},{HostId}"; } }

### Host

public class Host : Entity

{

public Host() : base()

{

Db.CurrentDbContext.Hosts.Add(this);

}

// Host entity tag prefix

public override string DbTagPrefix => "HST";

// An optional field for the IP address of the node, if applicable

public string? IPaddr { get; set; }

public NodeHostMapping? NodeHostMapping { get; set; }

public ICollection<HostServiceMapping> Services { get; set; } = new List<HostServiceMapping>();

public override void PrintInfo()

{

base.PrintInfo();

Console.WriteLine("IP Address:\t\t" + IPaddr);

Console.Write("Services:\t\t");

foreach (var service in Services)

{

PrintInfo();

}

}

public override string ExportObject()

{

// Return a string representation of the Host object containing every property

return $"{DbTag},{Name},{Description},{CreatedDate},{ModifiedDate},{IPaddr}";

}

}

### NetworkUser

public class NetworkUser : Entity

{

public NetworkUser() : base()

{

Db.CurrentDbContext.NetworkUsers.Add(this);

}

public override string DbTagPrefix => "USR";

[Required]

[Display(Name = "User Name")]

public string Username { get; set; } = string.Empty;

[Display(Name = "Email Address")]

public string? Email { get; set;}

[Display(Name = "First Name")]

public string? Firstname { get; set; }

[Display(Name = "Last Name")]

public string? Lastname { get; set; }

public ICollection<NetworkUserMapping> AllowedNodes { get; set; }

public override void PrintInfo()

{

base.PrintInfo();

Console.WriteLine("Username:\t" + Username);

Console.WriteLine("Email:\t" + Email);

Console.WriteLine("First Name:\t" + Firstname);

Console.WriteLine("Last Name:\t" + Lastname);

}

public override string ExportObject()

{

// Return a string representation of the NetworkUser object containing every property

return $"{DbTag},{Username},{Email},{Firstname},{Lastname}";

}

}

## Appendix C – Database seed data

* When the CMDB is started for the first time, it asks the user if they wish to seed it with some example data. This data I collected from my personal collection of computers, so while it is “accurate” for my own scenario, it is highly unlikely that an organization in 2024 has 1980s Z80-based computers.
* The following computers were used for the pre-seed data:
  + Amstrad CPC 6128
    - CPU: Zilog Z80A @ 4MHz
    - RAM: 128Kb
    - Storage: Gotek USB Floppy Emulator (~180Kb)
    - Deployed: Home
  + Commodore Amiga 500
    - CPU: Motorola 68030 @ 33MHz
    - RAM: 6MB (2MB Chip Mem, 4MB Fast Mem)
    - Storage: 4GB Compact Flash, Gotek USB Floppy Emulator (~880Kb)
    - Deployed: Home
  + Synology DS214play 2-bay Network Attached Storage
    - CPU: Intel Atom 32-bit
    - RAM: 2GB
    - Storage: 16TB SATA RAID
    - Deployed: Garage
  + Dell Optiplex 3020
    - CPU: Intel Core i3-4160
    - RAM: 4GB
    - Storage: 16TB SATA RAID
    - Deployed: Office
  + Raspberry Pi 4
    - CPU: Broadcom System-on-Chip (arm64 architecture)
    - RAM: 4GB
    - Storage: 32GB SanDisk microSD
    - Deployed: Home

// Would usually be specified by the user at data entry, random date used instead.

DateTime example = new DateTime(2018, 4, 27);

DateTime LastYear = DateTime.Now.AddYears(-1);

DateTime TenYearsAgo = DateTime.Now.AddYears(-10);

// Create ConfigItems

var cA500 = new ConfigItem { Name = "Amiga 500", PurchaseDate = example, DeployLoc = "Home", Description = "Amiga 500 Rev 5a with 68030 CPU and RAM upgrade", };

var cA6128 = new ConfigItem { Name = "Amstrad CPC 6128", PurchaseDate = DateTime.Now, DeployLoc = "Home", Description = "Amstrad CPC 6128 with Gotek floppy emulator" };

var cNAS1 = new ConfigItem { Name = "16TB Intel Atom NAS", PurchaseDate = TenYearsAgo, DeployLoc = "Garage", Description = "Synology DS214play NAS 8TB with 2x 4TB USB 3.2 HDD (16TB)"};

var cNAS2 = new ConfigItem { Name = "Ubuntu Server NAS", PurchaseDate = DateTime.Now, DeployLoc = "Office", Description = "Dell Optiplex 3020 with external 16TB HDD RAID array"};

var cRPI = new ConfigItem { Name = "Raspberry Pi 4", PurchaseDate = LastYear, DeployLoc = "Home", Description = "Raspberry pi 4 4GB with 32GB microSDHC storage"};

// Create Nodes

// Nodes are created for the Amiga 500, Amstrad 6128, and both 16TB NAS servers.

var nA500 = new Node { Name = "Amiga 500", OS\_Version = "AmigaOS", CPU\_Arch = "Motorola 68000", RAM = 1, Storage = 20};

var nA6128 = new Node { Name = "Amstrad CPC 6128", OS\_Version = "Amstrad CP/M", CPU\_Arch = "Zilog Z80", RAM = 0.5, Storage = 3 };

var nNAS1 = new Node { Name = "16TB Synology NAS", OS\_Version = "DSM Linux 3.2.101", CPU\_Arch = "Intel Atom", RAM = 16, Storage = 16000 };

var nNAS2 = new Node { Name = "16TB Ubuntu NAS", OS\_Version = "Ubuntu Server 20.04", CPU\_Arch = "Intel Core i7", RAM = 4, Storage = 16000 };

// Create Hosts. These have two properties: The hostname and IP address.

var hNAS1 = new Host { Name = "HOMENAS", IPaddr = "192.168.2.100"};

var hNAS2 = new Host { Name = "SHEDNAS", IPaddr = "192.168.2.200" };

// Create Network Users. These have two properties: the username, and email address.

// This will be improved to support passwords and other authentication methods.

var networkUser = new NetworkUser { Username = "p2626517", Email = "p2626517@my365.dmu.ac.uk" };

var networkUser2 = new NetworkUser { Username = "arawn", Email = "arawn.davies780@gmail.com" };

// Create Software

var DSM = new Software { Name = "Synology DiskStation Manager", Version = "7.1" };

var AWB = new Software { Name = "Amiga Workbench", Version = "3.1" };

var steamLauncher = new Software { Name = "Steam", Version = "Latest", Developer = "Valve", LicenseType = "Proprietary", Description = "Steam Games Launcher" };

// A SoftwareInstallation has two properties;

// the software to be installed and the node it is installed on.

SoftwareInstallation sDSM = new SoftwareInstallation { Node = nNAS1, Software = DSM };

SoftwareInstallation sAWB = new SoftwareInstallation { Node = nA500, Software = AWB };

nNAS1.InstalledSoftware.Add(sDSM);

DSM.Installations.Add(sDSM);

nA500.InstalledSoftware.Add(sAWB);

AWB.Installations.Add(sAWB);

// Create Services for a local website, Synology SFTP server and Ubuntu SSH server

var svcHTTP = new Service { Name = "Local web site", PortNum = 80 };

var svcFTP = new Service { Name = "Synology SFTP", PortNum = 22 };

var svcSSH = new Service { Name = "Ubuntu SSH", PortNum = 22 };

// Create initial CI Node Mappings – these entities have two properties:

// the ConfigItem and its accompanying Node.

var cA500Mapping = new CINodeMapping { ConfigItem = cA500, Node = nA500 };

var cA6128Mapping = new CINodeMapping { ConfigItem = cA6128, Node = nA6128 };

var cNASMapping1 = new CINodeMapping { ConfigItem = cNAS1, Node = nNAS1 };

var cNASMapping2 = new CINodeMapping { ConfigItem = cNAS2, Node = nNAS2 };

// Create initial Node Host Mappings – these entities have two properties:

// the Node and its accompanying Host.

// Not every Node necessarily has an accompanying Host as thin-clients and other low-compute devices only operate in guest mode.

var NHM1 = new NodeHostMapping { Node = nNAS1, Host = hNAS1 };

var NHM2 = new NodeHostMapping { Node = nNAS2, Host = hNAS2 };

// Map each Node and Host to each Node Host Mapping:

nNAS1.NodeHostMapping = NHM1;

hNAS1.NodeHostMapping = NHM1;

nNAS2.NodeHostMapping = NHM2;

hNAS2.NodeHostMapping = NHM2;

// Create initial Network User Mappings, these have two properties:

// The Node to be accessed and the User with access.

var NUM1 = new NetworkUserMapping { Node = nNAS1, NetworkUser = networkUser };

var NUM2 = new NetworkUserMapping { Node = nNAS2, NetworkUser = networkUser2 };

// Create initial Software Installations, these have two properties:

// The Node for the Software to be installed on.

var SI1 = new SoftwareInstallation { Node = nNAS1, Software = DSM };

var SI2 = new SoftwareInstallation { Node = nA500, Software = AWB };

// Create initial Host Service Mappings, these have two properties:

// The Host and its underlying Services.

var HSM1 = new HostServiceMapping { Host = hNAS1, Service = svcFTP };

var HSM2 = new HostServiceMapping { Host = hNAS2, Service = svcHTTP };

var HSM3 = new HostServiceMapping { Host = hNAS2, Service = new Service { Name =

svcSSH.Name, PortNum = svcSSH.PortNum } };

// Map each Service to the corresponding HostServiceMapping

svcFTP.HostServiceMapping = HSM1;

svcHTTP.HostServiceMapping = HSM2;

svcSSH.HostServiceMapping = HSM3;

## Appendix D – Test Suite

* Unit tests were used for the creation, reading, updating and deletion of database entries and assets. The unit testing framework used for this was MSTest as it was provided by the .NET SDK and didn’t require any further modifications to the projects.
* The Cosmos project also includes a testing suite that can be used with various tools, e.g. VMWare Player, QEMU, Bochs x86 Emulator, and RS232 serial on bare metal.
* For the microCMDB CLI project, unit tests were developed that will test against the create, read, update, and delete methods of the database. While the project was ultimately incomplete, once the codebases of the CLI and Web projects are unified then testing would become a lot more streamlined, there would be less redundancy and more efficient and extensive tests.

public void SoftwareStore()

{

// Create a new instance of the Db class

Db.CurrentDbContext = new Db();

Software egApp = new Software { Name = "TestApp", Version = "1.0" };

Assert.IsTrue(Db.CurrentDbContext.Software.First().Name == "TestApp");

}

public void ConfigItemStore()

{

Db.CurrentDbContext = new Db();

ConfigItem egConfigItem = new ConfigItem { Name = "TestConfigItem" };

Assert.IsTrue(Db.CurrentDbContext.ConfigItems.First().Name == "TestConfigItem");

}

public void NodeStore()

{

Db.CurrentDbContext = new Db();

Node egNode = new Node { Name = "TestNode" };

Db.CurrentDbContext.Nodes.Add(egNode);

Assert.IsTrue(Db.CurrentDbContext.Nodes.First().Name == "TestNode");

}

public void HostStore()

{

Db.CurrentDbContext = new Db();

Host egHost = new Host { Name = "TestHost" };

Assert.IsTrue(Db.CurrentDbContext.Hosts.First().Name == "TestHost", "Failed to store a Host");

}

public void ServiceStore()

{

Db.CurrentDbContext = new Db();

Service egService = new Service { Name = "TestService" };

Assert.IsTrue(Db.CurrentDbContext.Services.First().Name == "TestService", "Failed to store a Service");

}

public void NetworkUserStore()

{

Db.CurrentDbContext = new Db();

NetworkUser egNetworkUser = new NetworkUser { Name = "TestNetworkUser" };

Assert.IsTrue(Db.CurrentDbContext.NetworkUsers.First().Name ==

"TestNetworkUser", "Failed to store a Network User");

}

public void DeleteTest()

{

Db.CurrentDbContext = new Db();

Node egNode = new Node { Name = "TestNode", CPU\_Arch = "x86\_64", RAM = 32768,

Description = "Put a more descriptive description here"};

Console.WriteLine(egNode.ExportObject());

CLI.Util.IO.DeleteEntity(egNode.DbTag);

bool found = CLI.Util.Get.Find(egNode.DbTag);

Assert.IsFalse(found, "Failed to delete Node entity from database");

}

public void CITagTest()

{

Db.CurrentDbContext = new Db();

ConfigItem egConfigItem = new ConfigItem { Name = "TestConfigItem" };

Assert.IsTrue(egConfigItem.DbTag.StartsWith("CFG"));

}

public void NodeTagTest()

{

Db.CurrentDbContext = new Db();

Node egNode = new Node { Name = "TestNode" };

Assert.IsTrue(egNode.DbTag.StartsWith("NOD"));

}

public void HostTagTest()

{

Db.CurrentDbContext = new Db();

Host egHost = new Host { Name = "TestHost" };

Assert.IsTrue(egHost.DbTag.StartsWith("HST"));

}

public void ServiceTagTest()

{

Db.CurrentDbContext = new Db();

Service egService = new Service { Name = "TestService" };

Assert.IsTrue(egService.DbTag.StartsWith("SVC"));

}

public void SoftwareTagTest()

{

Db.CurrentDbContext = new Db();

Software egSoftware = new Software { Name = "TestSoftware" };

Assert.IsTrue(egSoftware.DbTag.StartsWith("STW"));

}

public void NetworkUserTagTest()

{

Db.CurrentDbContext = new Db();

NetworkUser egNetworkUser = new NetworkUser { Name = "TestNetworkUser" };

Assert.IsTrue(egNetworkUser.DbTag.StartsWith("USR"));

}

// Test the DbTag property for each entity type

public void DbTagTests()

{

CITagTest();

NodeTagTest();

HostTagTest();

ServiceTagTest();

SoftwareTagTest();

NetworkUserTagTest();

}

## Appendix E – Command Line Interface User Documentation

* The CMDB Command Line Interface (CLI) provides a convenient way to interact with the Configuration Management Database (CMDB) system. With the CLI, users can perform various operations such as retrieving, setting, creating, and deleting entities within the CMDB.
* Available Commands
  + General Commands
    - help: Display available commands and their descriptions.
    - get: Retrieve entities from the CMDB.
    - set: Set entities in the CMDB.
    - new: Create new entities in the CMDB.
    - find <dbtag>: Print information for a specific entity.
    - delete <dbtag>: Delete a specific entity.
    - export: Export entities from the CMDB.
    - exit: Exit the CMDB CLI.
  + Specific Commands
    - Retrieval Commands (get)
    - get configitems: Retrieve ConfigItems from the CMDB.
    - get nodes: Retrieve Nodes from the CMDB.
    - get hosts: Retrieve Hosts from the CMDB.
    - get services: Retrieve Services from the CMDB.
    - get software: Retrieve Software from the CMDB.
    - get networkusers: Retrieve NetworkUsers from the CMDB.
  + Setting Commands (set)
    - set configitems: Set ConfigItems in the CMDB.
    - set nodes: Set Nodes in the CMDB.
    - set hosts: Set Hosts in the CMDB.
    - set services: Set Services in the CMDB.
    - set software: Set Software in the CMDB.
    - set networkusers: Set NetworkUsers in the CMDB.
  + Creation Commands (new)
    - new configitem: Create a new ConfigItem in the CMDB.
    - new node: Create a new Node in the CMDB.
    - new host: Create a new Host in the CMDB.
    - new user: Create a new User in the CMDB.
    - new service: Create a new Service in the CMDB.
    - new software: Create a new Software in the CMDB.
  + Usage
    - To use the CMDB CLI, simply type the desired command followed by any necessary parameters. For example:
      * To display available commands: “help”
      * To retrieve Nodes from the CMDB: “get nodes”
      * To create a new Host in the CMDB: “new host”

# References

1. ‌Alawairdhi, M. (2016) ‘*Agile development as a change management approach in software projects: Applied case study’*. 2nd International Conference on Information Management (ICIM), London, UK, 2016, pp. 100-104, doi: 10.1109/INFOMAN.2016.7477541.
2. Atlassian (n.d.) *What is a Configuration Management Database (CMDB)?* Available at: <https://www.atlassian.com/itsm/it-asset-management/cmdb> (Accessed: 10/01/2024).
3. Chatziamanetoglou, D.; Rantos, K. *‘Blockchain-Based Security Configuration Management for ICT Systems.’ Electronics 2023, 12, 1879*. <https://doi.org/10.3390/electronics12081879>
4. Combodo (2012), *iTop, GitHub* Available at: <https://github.com/Combodo/iTop/blob/develop/license.txt> (Accessed at 19/01/2024)
5. Combodo (2023), *Service Management Module.* Available at: <https://www.itophub.io/wiki/page?id=latest:datamodel:itop-service-mgmt> (Accessed 20/01/2024)
6. CosmosOS (2022) *'Test Runner', C# Open-Source Managed Operating System*. Available at: <https://cosmosos.github.io/articles/Tests/TestRunner.html> (Last updated: 9 December 2022, Accessed: 10 March 2023).
7. CosmosOS (2024) ‘*IL2CPU Compiler’, Cosmos.* Available at: <https://cosmosos.github.io/articles/Compiler/il2cpu.html> (Last updated 24/03/2024, Accessed 03/04/2024).
8. Davies, A., GitHub (2023) *‘Medli Operating System’, GitHub*. Available at: <https://github.com/Arawn-Davies/Medli/tree/main/Medli> (Last updated: 16 September 2023, Accessed 17th March 2024).
9. Davies, A., GitHub (2018) ‘*Medli Classic’, GitHub.* Available at <https://github.com/Arawn-Davies/Medli-Classic> (Last updated: 19th October 2018, Accessed 20th March 2024).
10. Farayola O.A., et al. (2023) *Configuration management in the modern era: best practices, innovations, and challenges’, Computer Science & IT Research Journal,* 4(2), pp. 140–157. doi:10.51594/csitrj.v4i2.613.
11. Gupta, S., TrustRadius (2021) *IBM Tivoli CCMDB*. Available at: [https://www.trustradius.com/products/ibm-tivoli-change-and-configuration-management-database/reviews#reviews](https://www.trustradius.com/products/ibm-tivoli-change-and-configuration-management-database/reviews#reviewsA) (Accessed 15/01/2024)
12. Strehovsky, M (2022) *ZeroSharp Project Repository.* Available at: <https://github.com/MichalStrehovsky/zerosharp/tree/master/efi-no-runtime> (Accessed 31/03/2024).
13. IBM (n.d.). *Configuration Management Database.* Available at: <https://www.ibm.com/docs/en/control-desk/7.6.1?topic=overview-configuration-management-database> (Accessed 08/01/2024).
14. Krishnamurti, S., VMWare (2007) *'Get Juiced!', VMware's Executive Blog*. Available at: <https://archive.ph/20120708100444/http://blogs.vmware.com/console/2007/07/get-juiced.html> (Archived from the original on 8 July 2012). Accessed: 20 February 2023.
15. Lilly, I., ServiceNow (2023) *How HSBC is using ServiceNow Impact to scale its use of the Now platform.* Available at: <https://www.servicenow.com/community/servicenow-impact-events/how-hsbc-is-using-servicenow-impact-to-scale-its-use-of-the-now/ev-p/2674940?attachment-id=255739> (Accessed: 27/12/2023).
16. McDonald, P. (2009) *Best way to draw a table in a C# Console app.* Available at: <https://stackoverflow.com/questions/856845/how-to-best-way-to-draw-table-in-console-app-c> (Accessed: 15 February 2024).
17. Microsoft, et. Al. (2022) Dotnet/*NativeAOT, GitHub.* Available at: <https://github.com/dotnet/runtimelab/tree/feature/NativeAOT> (Accessed 15/03/2024)
18. RedHat (11/05/2022) *What is KVM?* Available at <https://www.redhat.com/en/topics/virtualization/what-is-KVM> (Accessed 18/03/2024)
19. SPOC.EU (n.d.) *6 Stories of implementing CMDB in ServiceNow*. Available at: <https://www.spoc.eu/ebooks/cmdb-implementation-case-studies-whitepaper/> (Accessed: 13/01/2024)
20. SUSE (2024) *SUSE Defines: JeOS (Just Enough Operating System)* Available at: <https://www.suse.com/suse-defines/definition/jeos-just-enough-operating-system/> (Accessed 20/03/2024)
21. TrustRadius (2021) *Combodo iTop*. Available at: <https://www.trustradius.com/products/combodo-itop/reviews#reviews> (Accessed 17/01/2024)
22. TrustRadius (2022) *ServiceNow IT Service Management.* Available at: <https://www.trustradius.com/products/servicenow-it-service-management/reviews#reviews> (Accessed 18/01/2024)
23. University of Warwick (n.d.) *'Dyspraxia', Neurodiversity Toolkit.* Available at: <https://warwick.ac.uk/fac/cross_fac/academy/activities/learningcircles/neurodiversity/toolkit/thebasics/dyspraxia/#:~:text=Students%20with%20Dyspraxia%20may%20struggle,deadlines%20as%20a%20beginning%20point> (Accessed: 17/04/2024).