Bachelor's thesis

Information and Communications Technology

2024

Santeri Sinisalo

User-centric Development of an Efficient Device Inventory Management Application for a University Lab

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2024 | 23 pages

Santeri Sinisalo

User-centric Development of an Efficient Inventory Management System for a University Lab

Inventory management is a standard activity for almost all organisations globally. While there are many software solutions for inventory management, many of them are too costly for small enterprises which then opt for manual spreadsheet tracking of their equipment and goods. In addition to cost factors, there is also little research carried out to show that inventory management software is, in fact, more efficient than spreadsheet tracking in small companies. The problem with spreadsheet systems, however, is that it opens the door for mistakes and the potential loss of valuable equipment. The Futuristic Interactive Technologies research group had been using a spreadsheet to manage their equipment loan system and wanted an efficient and user-friendly replacement software. The objectives of this thesis were to develop an inventory management application through a user-centric iterative development approach and to quantitatively test its efficiency. Observation was used during the development cycles to determine the user-friendliness while the efficiency was measured by conducting a direct comparison experiment with the spreadsheet. The experiment involved seven participants who borrowed and returned equipment three times respectively in both systems. The results indicated that the new application was significantly more efficient to use ($p<0.0001$ at $alpha = 0.05$). This thesis concludes that bespoke inventory management systems are more efficient than a spreadsheet for small companies. Furthermore, this thesis contributes theoretically showing how a qualitative iterative user-centric design and development approach can be used to optimize the quantitative efficiency of an inventory management software.

Keywords:

User-centered design, inventory management, user interfaces, software development

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintätekniikka

2024 | 23 sivua

Santeri Sinisalo

Tehokkaan inventaariojärjestelmän käyttäjäkeskeinen kehitys korkeakoululaboratoriolle

Inventaarion hallinta on yleinen tehtävä lähes kaikille yrityksille maailmanlaajuisesti. Vaikka monia ohjelmistoratkaisuja on saatavilla inventaarion hallintaan, monet niistä ovat liian hintavia pienille yrityksille. Sen sijaan yritykset päätyvät hallinnoimaan inventaariotaan taulukkolaskentaohjelmilla. Lisäksi tutkimuksia, jotka vertaavat inventaarionhallintaohjelmistojen ja taulukkolaskentaohjelmien tehokkuutta ovat vähäisiä. Taulukkojärjestelmien ongelma on, että niissä on vaikeaa pienentää käyttäjistä johtuvia virheitä. Virheet voivat johtaa arvokkaiden laitteiden menetykseen, joten niiden vähentäminen on tärkeää. Tulevaisuuden interaktiiviset teknologiat -tutkimusryhmä on käyttänyt taulukkolaskentaohjelmaa laitelainauksiensa hallitsemiseen ja heillä on tarve korvata se tehokkaalla ja helppokäyttöisellä ohjelmistolla.

Tämän opinnäytetyön tavoite oli kehittää inventaarionhallintasovellus käyttäen käyttäjäkeskeistä iteratiivista kehitystapaa ja samalla kvantitatiivisesti testata sen tehokkuutta. Järjestelmän käyttäjäystävällisyyttä arvioitiin kehityskierroksien aikana havainnoimalla käyttäjiä. Tehokkuutta mitattiin tekemällä vertailukoe sovelluksen ja taulukkolaskentaohjelman välillä. Tutkimukseen osallistui seitsemän käyttäjää, jotka tekivät kolme lainausta ja kolme palautusta molemmilla järjestelmillä. Tutkimuksen tulokset osoittivat, että uusi sovellus oli huomattavasti tehokkaampi ($p<0.0001$, $alpha = 0.05$).

Tämän opinnäytetyön johtopäätös on, että erikoisvalmisteiset inventaarionhallintajärjestelmät ovat tehokkaampia pienemmille yrityksille, kuin taulukkolaskentajärjestelmät. Tämän lisäksi opinnäytetyössä osoitetaan, miten kvalitatiivista ja iteratiivista käyttäjäkeskeistä suunnittelu- ja kehittämistapaa voidaan käyttää optimoimaan inventaariohallintasovelluksen kvantitatiivista tehokkuutta.

Asiasanat:

Käyttäjäkeskeinen suunnittelu, varastonhallinta, käyttöliittymät, ohjelmistokehitys

Contents

Figures

Tables

List of abbreviations

ATM	Automated Teller Machine
CSS	Cascading Style Sheets
DBMS	Database Management System
ER	Entity-Relationship
FIT	Futuristic Interactive Technologies
GUID	Globally Unique Identifier
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
MVP	Minimum Viable Product
RFID	Radio Frequency Identification
TUAS	Turku University of Applied Sciences

1 Introduction

Keeping track of inventory is a common task in many organizations. To help with this task, various software solutions exist for inventory management, the market for which is valued at 2.8 billion dollars (Business Research Insights, 2024). These software solutions range from simple asset tracking applications (itemit, 2024) to large software suits that include warehouse management, ecommerce integration, manufacturing materials planning and more (Cin7, 2024). The pricing of these software solutions also varies. The pro tier of Asset tracker itemit starts at $1299 per year for 500 tracked items (itemit, 2024). The Cin7 standard plan is $349 per month (Cin7, 2024). The Katana starter plan is $179 per month (Katana, 2024). Small companies with small inventories may look at these prices and decide that the cost to benefit ratio is not in their favor and opt instead for keeping track of their inventory using a spreadsheet.

The Futuristic Interactive Technologies research group (FIT), within Turku University of Applied Sciences (TUAS), is one such entity maintaining a small-scale inventory. In addition to their research tasks, they are responsible for providing the students at TUAS with devices for use with their projects. These devices include laptops, Virtual Reality headsets, cameras, and many other valuable items that students are not expected to purchase themselves. Thus far, multiple methods have been used to track who has borrowed what, including spreadsheets, Trello boards, and sticky notes. None of these solutions have been deemed satisfactory by the engineers working at FIT, with all of them containing different or contradictory information and being inconvenient to use.

In order to solve these problems, FIT commissioned this thesis. The objectives of this thesis were to create an inventory management application that uses a central database that can be accessed with an easy-to-use frontend and to test its efficiency through a direct comparison experiment with their current spreadsheet system. The application should have the functionality to use a camera or scanner to scan barcodes to help automate the loaning process. The

backend should be able to store information about the devices in a database as well as their current status.

To create an application that would be as efficient and easy to use as possible, an iterative development cycle would need to be used where a minimum viable product (MVP) would be created and then given to the engineers for testing. During the testing phase, the engineers would give feedback on the application and their behavior using it would be observed. Once the application would be in a satisfactory state, a study would be conducted where the participants' time taken to conduct regular loaning activities using the new application and the spreadsheet they are currently using would be measured and compared. This thesis compares the times of 42 loaning activities across 7 participants to determine which of the two inventory systems is most efficient. The thesis further reports on the qualitative strengths and weaknesses of both systems. The expected finding would be that the purpose made inventory application would be more efficient than the spreadsheet system.

The rest of this thesis is structured as follows: Chapter 2 of the thesis provides a theoretical framework of user centric design and literature review of studies related to the efficiency of inventory management systems. Chapter 3 gives a detailed description of the research prototype is provided. Chapter 4 covers the research methodology used in the study, the results of which are covered in Chapter 5. Chapter 6 discusses the results and their implications and finally Chapter 7 concludes the thesis and gives suggestions for future research.

2 Theoretical framework and literature review

When designing the user interface, the design principles laid out in Donald A. Norman's The design of everyday things were used. In it, Norman advocates for the idea of user centric design which prioritizes usability over aesthetics. Norman gives seven principles of design which are as follows:

1. "Use both knowledge in the world and knowledge in the head."
2. "Simplify the structure of tasks." Individual tasks should not be too complex and if they are, they should be broken down into multiple, simpler tasks.
3. "Make things visible." The system should inform the user about its state and should not hide crucial information.
4. "Get the mappings right." The placement of objects should feel natural, and controls should do what the user thinks they do.
5. "Exploit the power of constraints." The user should feel like there is only one possible thing to do, which is the right thing to do.
6. "Design for error." Assume that any error that can be made, will be made, so the design should accommodate user error.
7. "When all else fails, standardize." If a standard exists, use it. (Norman, 1988, pp. 188-203)

Studies into the benefits of using inventory management software have been done in the past. A study by Dennert et al. (2021) found that using a custom-made Microsoft Access database produced significant improvements to their overall workflow and outcomes and prevented thousands of dollars' worth of additional storage purchases. They also saw a reduction in time spent searching for items and filling out paper documents. A different study by Atieh et al. (2016) found that replacing a manual management system with an automated system resulted in a more reliable and efficient system. A study by Nisha (2018) into replacing a libraries barcode system with a Radio Frequency Identification (RFID) system concluded that the RFID system was a more efficient, effective, user friendly system in comparison to the barcode system.

However, they also concluded that the RFID system was costly and did not make economic sense to a small library.

While these studies show that an inventory management system improves efficiency, only one of them provides a comparative quantitative measure against non-database solutions. Extensive literature searching did not provide any other studies that reported on the time efficiency aspect of custom inventory management systems. This indicates that organisations most likely buy into the sales pitches of large corporations that claim great time savings when using their systems.

This thesis aims to make an empirical contribution to the scientific knowledge base on the time efficiency of small-scale inventory management. What also sets this thesis apart from existing work, is its theoretical contribution. This thesis uses a qualitative iterative user-centric design and development approach to optimize the quantitative efficiency of the inventory management software. The goal of this thesis is not only to have an efficient inventory management system, but also a highly usable one.

3 Application description

The plan in the beginning was to create an Android application that could read barcodes using the phone's camera. Each loanable device would have a sticker with a barcode that the application could read and then mark it as borrowed or available. Through the course of development, the project's scope expanded, and new features were added. From this point forward the application will be referred to as Borrowtool.

Borrowtool consists of three parts: the database, backend, and frontend. In planning the application, it was important to keep the following in mind: First was that the database and backend would be hosted on Azure meaning that when picking a database management system (DBMS) or framework it would have to be supported by Azure. Second was that the engineers working at FIT have a bachelor's degree in information and communication technology from Turku University of Applied Sciences with a specialization in Game and Interactive Technologies. In this programme the main programming language they are taught is C# and many of their projects are made with the Unity game engine (Turku University of Applied Sciences, 2024). To play into the engineers' strengths, C# should be used as the programming language when possible so that they can easily continue developing the application. Third was that whatever would be used for the frontend must be compatible with Android.

3.1 Database

The database was made using MySQL and has four tables: items, statuses, users, and itemstatuslogs (Figure 1).

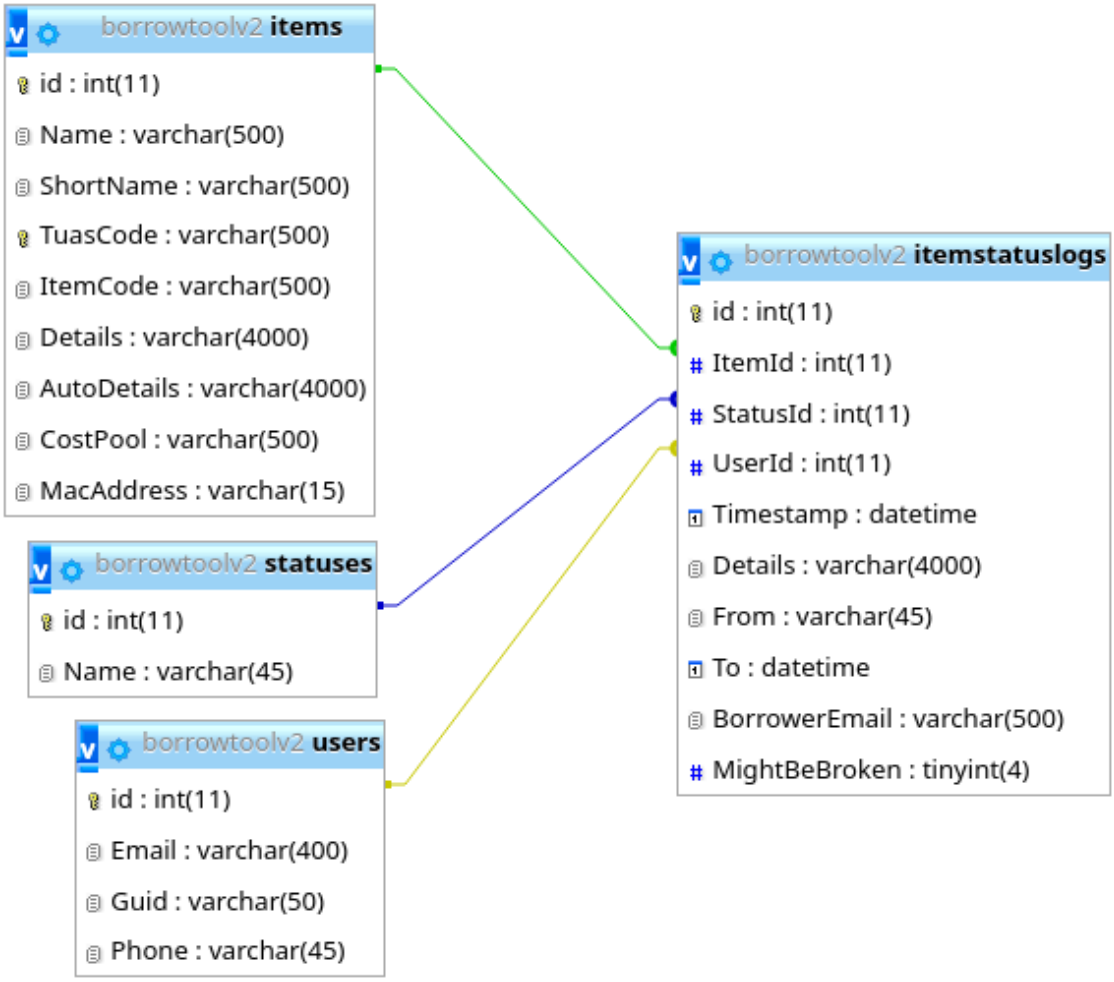

Figure 1. Borrowtool backend entity-relationship (ER) diagram.

The Items table contains data about the items such as its name, barcode, and serial number. The Statuses table has all the possible statuses an item can have, namely: available, borrowed, and has problems. The Users table contains data about users like their email and work phone number. The Users table is not meant to be used as an account but instead it is used for identifying which engineer is using the application. The users are identified by their device's globally unique identifier (GUID) instead of having them log in. Finally, there is

the itemstatuslogs table which is where the items' logs are stored. Every time an item's status is changed, like going from available to borrowed, a new log is added with that status. It also stores the user who made the log, a timestamp, miscellaneous details about the log, and possible return date and borrower email.

3.2 Backend

The backend was made with ASP.NET Core which is a framework for building web services (Microsoft, 2023a). An important reason for picking ASP.NET Core is that it uses C#. Manipulating the database happens through the Entity Framework (Microsoft, 2024a).

The backend is separated into models and controllers. Models contains the scaffolded entity type classes from the database schema as well as classes for Hypertext Transfer Protocol (HTTP) requests and responses. Controllers contains methods for processing HTTP requests such as adding items.

To use the database with the backend, the database schema needs to first be scaffolded into C# classes. This can be achieved with a command line tool called Scaffold-DbContext (Microsoft, 2023b). Entity framework can then use these scaffolded classes to manipulate the database. While these scaffolded classes can be serialized into JavaScript Object Notation (JSON) and sent to the frontend, it can often have issues. As an example, the scaffolded class Itemstatuslog (Figure 2) has a variable that references a scaffolded class Item.

```
using System;
using System.Collections.Generic;

namespace BorrowToolCore.Models.Borrowtool;

public partial class Itemstatuslog
{
    public int Id { get; set; }

    public int ItemId { get; set; }

    public int StatusId { get; set; }

    public int UserId { get; set; }

    public DateTime Timestamp { get; set; }

    public string? Details { get; set; }

    public string? From { get; set; }

    public DateTime? To { get; set; }

    public string? BorrowerEmail { get; set; }

    public sbyte? MightBeBroken { get; set; }

    public virtual Item Item { get; set; } = null!;

    public virtual Status Status { get; set; } = null!;

    public virtual User User { get; set; } = null!;
}

```

Figure 2. Scaffolded class Itemstatuslog.

The scaffolded class Item on the other hand has a variable that references the scaffolded class Itemstatuslog. This means that when trying to serialize Itemstatuslog and its referenced Item the JSON serializer will get stuck in an infinite loop. The scaffolded class can also contain data that you do not want to serialize, or they can be in the wrong format. For these reasons I used separate classes for serialization which I called DataModels. These DataModels can then be used with Request and Response classes. As an example, the DataModel

for Itemstatuslog is ItemstatuslogData (Figure 3) which lacks the scaffolded class's Id variables, references to other classes, and converts the MightBeBroken sbyte to a bool.

```
using BorrowToolCore.Models.Borrowtool;

namespace BorrowToolCore.Models.DataModels
{
    public class ItemstatuslogData
    {
        public DateTime Timestamp { get; set; }
        public string? Details { get; set; }
        public string? From { get; set; }
        public DateTime? To { get; set; }
        public bool MightBeBroken { get; set; }
        public string? BorrowerEmail { get; set; }

        public ItemstatuslogData ItemstatuslogToJson(Itemstatuslog itemstatuslog)
        {
            return new ItemstatuslogData
            {
                Timestamp = itemstatuslog.Timestamp,
                Details = itemstatuslog.Details,
                From = itemstatuslog.From,
                To = itemstatuslog.To,
                MightBeBroken = Convert.ToBoolean(itemstatuslog.MightBeBroken),
                BorrowerEmail = itemstatuslog.BorrowerEmail
            };
        }
    }
}

```

Figure 3. Serializable class ItemstatuslogData derived from Itemstatuslog.

The backend has four controllers for handling requests and responses: InventoryController, BorrowController, LogController, and UserController. InventoryController contains methods for adding, editing, and removing items. BorrowController contains methods for borrowing, returning, and changing the status of items. LogController contains methods for getting an item's logs and for searching. UserController has methods for adding, editing, and getting a user. Each of the controller methods take an object inherited from the Request class as an input. The Request class contains the requester's GUID, a timestamp, and the version number of the frontend.

3.3 Frontend

Many factors had to be considered when picking what to use for developing the frontend. First was that the frontend should be written in C#. Secondly, the application needs to run on Android, but it should not only run on Android. Third was that it needs to support the use of Android's camera. Fourth was that the application should be easy to port to a different operating system without the need for a rewrite.

These requirements filter out many commonly used tools for creating graphical applications. GTK (GTK Team, 2024) and Windows Forms (Microsoft, 2023c) cannot be used because they do not support Android. Qt does not officially support C# (Qt Wiki, 2023) and the third-party language bindings have not been updated in years (Qml.Net, 2020). The .Net Multi-platform App UI does not support Linux (Microsoft, 2024b).

Since none of the options seemed satisfactory, it was decided that using a game engine should be considered. While game engines are not generally meant for application development, they do offer many features that are useful for this project and the FIT engineers are very familiar with using them. The main game engines that were considered were Unity and Godot. Both support C#, cameras, multiple operating systems like Android, Linux, and Windows, and offer tools for creating graphical user interfaces. In the end Unity was chosen because it is what the engineers are the most familiar with and because of Unity's UI Toolkit which offers similar functionality as standard web technologies like HyperText Markup Language (HTML) and Cascading Style Sheets (CSS) (Unity Technologies, 2024).

3.3.1 Layout

The goal when planning the layout was to mimic the layouts of Windows applications which have had a common design since the 1990's. Windows applications usually have the title of the window at the top left, minimize,

maximize, and close buttons at the top right, content in the middle, and miscellaneous information at the bottom. An example of this is FileZilla (Figure 4), which has the window title and controls at the top, content in the middle, and a file transfer log at the bottom.

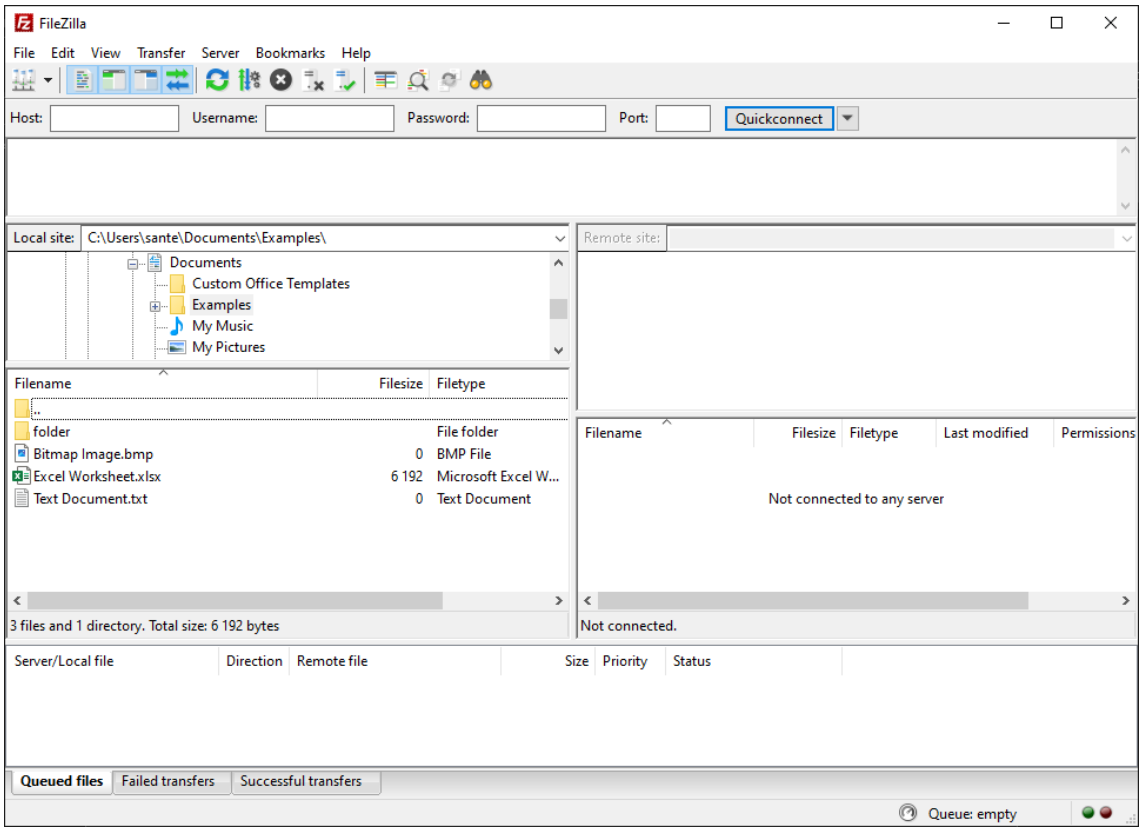

Figure 4. FileZilla layout example.

The layout of the user interface (Figure 5) for this study is separated into three containers: header container, content container, and log container. The header container is at the top of the screen and contains the title of the current menu, my email address for sending feedback, the application's version number and two buttons, one for toggling dark mode and one for returning to the main menu. The main menu button is meant to have similar placement and functionality as the close button on Windows. The content container is in the middle and has the menu selections and content of the selected menu. The log container is at the bottom and shows the latest debug log message and a button that opens a scrollable full screen window with a list of the previous debug messages. The debug messages can be informational messages or error messages like if a

HTTP request was successful or if a user's input is valid e.g. an email input is in a valid format. The header and log containers are set to take up 10% of the screen each and the content container expands to fill the rest.

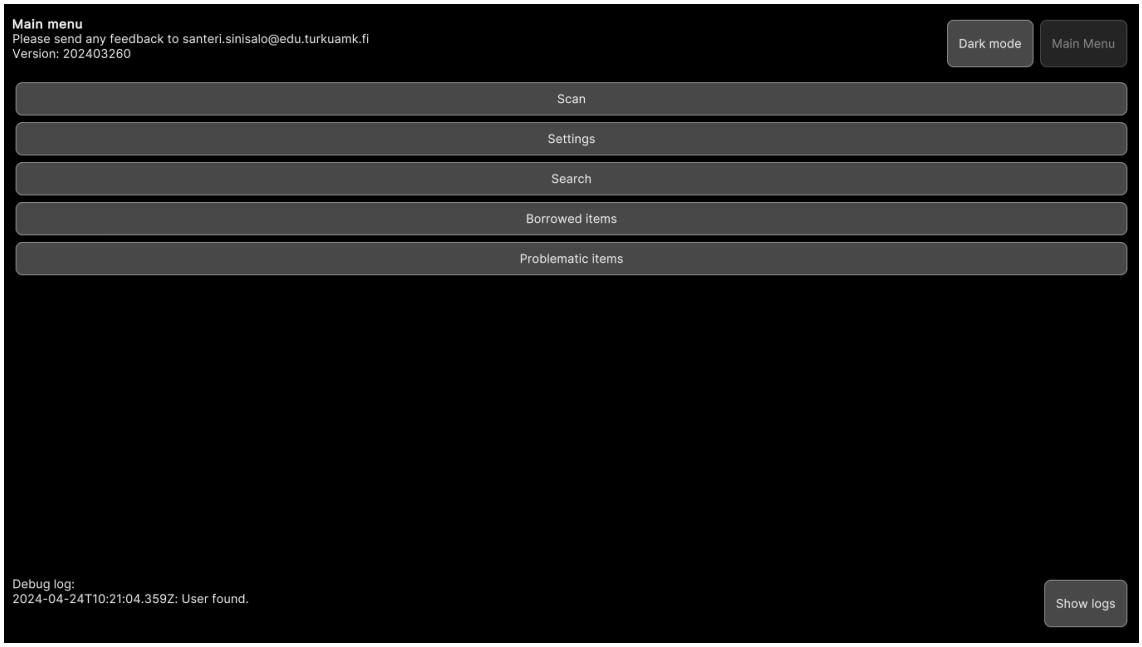

Figure 5. Borrowtool main menu.

3.3.2 Functionality

The main feature of the frontend is the ability to scan barcodes. This can be done with a USB barcode scanner or with a camera (Figure 6). When using a camera, a plugin called ZXing is used for decoding the barcode (Jahn, 2024). If the scanned code is not in the database, the user will be prompted to add the device. If the code is found in the database, the user will be given information about the device and a list of options like check device's logs, borrow or return the device, change device's status, edit device, and remove.

Figure 6. Mobile camera scan with scanned item.

The second major feature of the frontend is the ability to get data about the devices from the database. There are four different scenarios where one can get device data from the database, each of which uses the same menu (Figure

7) but with different parameters and criteria. The first way is through the search menu which will retrieve the latest log of each device that matches the parameters given by the user. These include filtering by status, device name, borrower, and costpool. The second way is to get all currently borrowed devices which has a dedicated button on the main menu. This method has a feature where it will show a device in red if it is due and has not yet been returned. The third way is to get all problematic devices, meaning all devices that are marked as broken or in need of repair. Finally, there is a way to check a specific item's logs which gives an item's history.

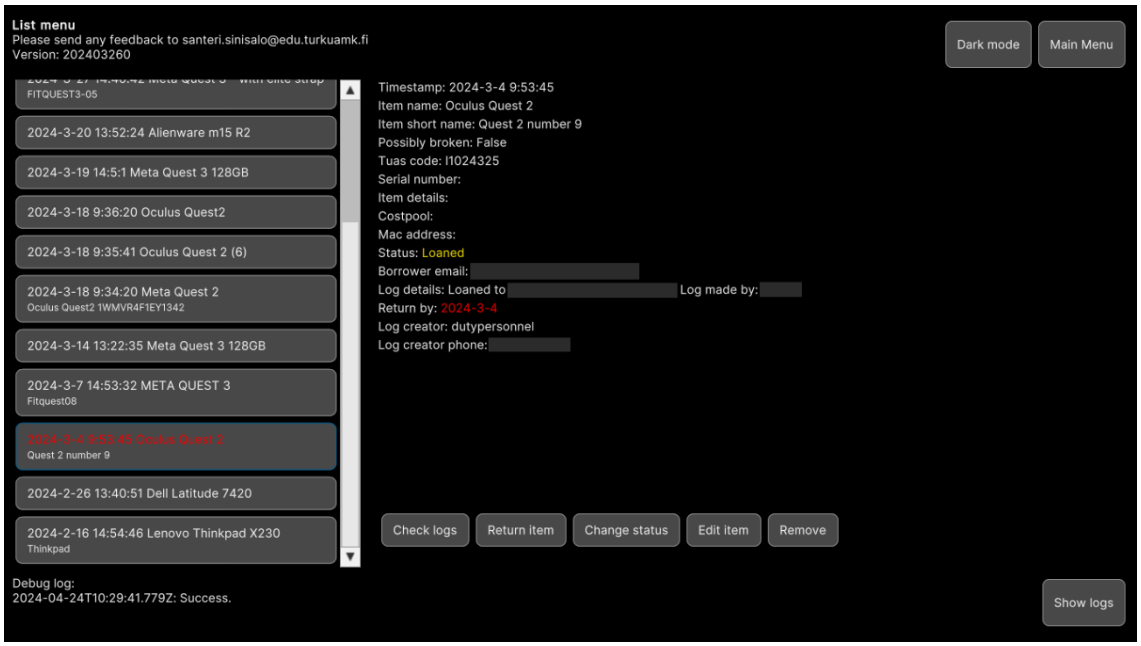

Figure 7. List menu showing currently borrowed devices.

3.3.3 Versions

There are three versions available of the frontend. The first is the Android version which can use a camera to scan codes. The second is a Windows desktop version which can use either a USB webcam or a USB barcode scanner to scan codes. Finally, there is the Linux kiosk version (Figure 8) which is different from the others in that it is meant to be used in a multiuser environment. This version is meant to be installed on a dedicated computer that

is only meant for running the frontend similar to an automated teller machine (ATM) or a library self-service kiosk. In the kiosk version, every time a method is executed that stores something in the database, the user is required to input their name. This is done because the device GUID cannot be used for identification since all requests are sent from the same device and not the user's personal device.

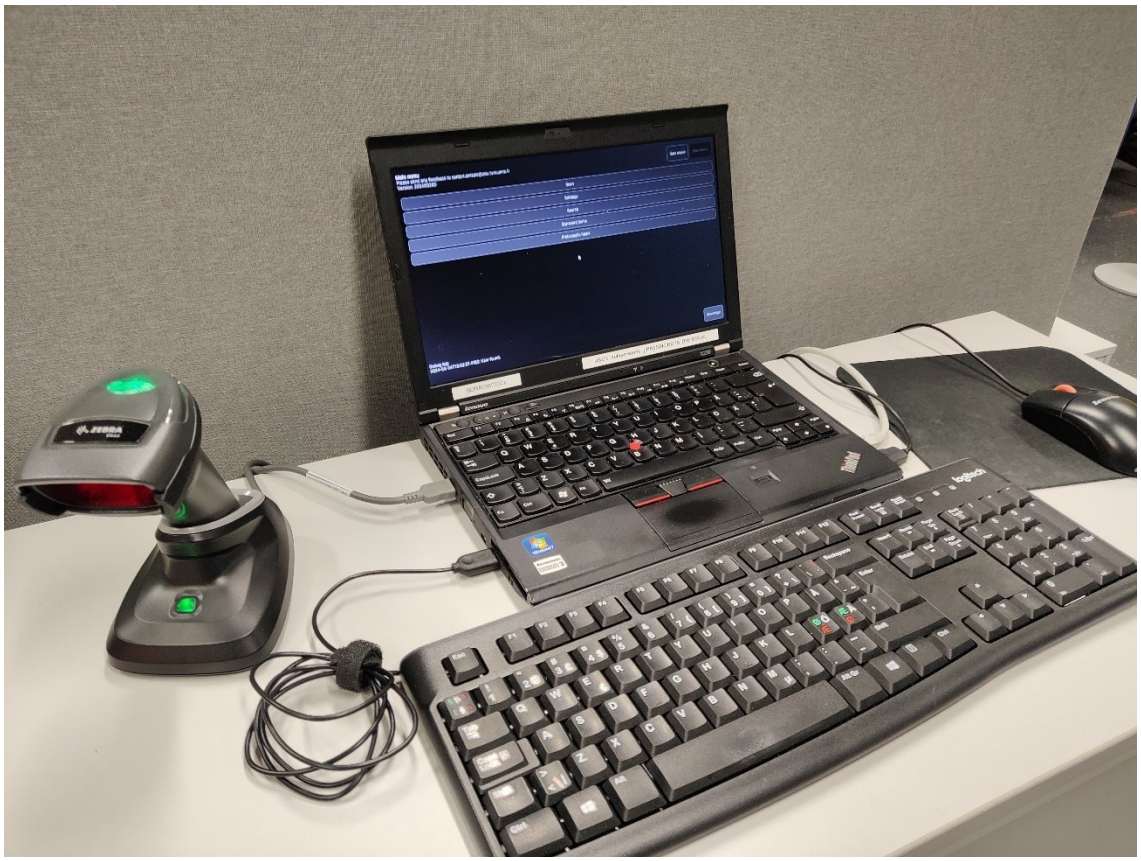

Figure 8. Linux kiosk.

4 Research methodology

In order to compare the efficiency of Borrowtool with the spreadsheet, a user centric comparison study between the two was conducted.

Before the study, Borrowtool went through a month of testing. The engineers used the application to loan out devices, report bugs, and request features. The engineers' behavior using the application was observed, noting how they tried to use the application, what they found frustrating, if they understood what the buttons did and what the input fields wanted them to input, and if the given instructions were clear. The observations and reports were marked in a text file and the application would then be iterated upon these observations and reports.

Seven participants were part of the study, of whom six were FIT engineers and one was a student assistant. These participants are a convenience sample of people who will in the future use Borrowtool on a daily basis. The engineers' work experience at FIT varied from six months to two years. All of them had used the spreadsheet in the past but the amount they had used it varied. Some, but not all of them, had tried Borrowtool during its testing phase but none of them were fully aware of its complete feature set. The one student assistant did not have any experience with the spreadsheet or Borrowtool.

Each of the participants tested the spreadsheet and Borrowtool separately. When testing Borrowtool they used the Linux kiosk version. They were given three items that they would loan and return with the spreadsheet and Borrowtool. Two of these items were already added to the spreadsheet and the database. One of them was not in the spreadsheet or the database, meaning the participants had to figure out that the item was not added, fill out its information, and then loan it. The participants were timed on how long it took them to loan each individual item. Their times were marked in a spreadsheet and analyzed. Figure 9 gives an overview of the iterative development cycles and expriment phases of the study.

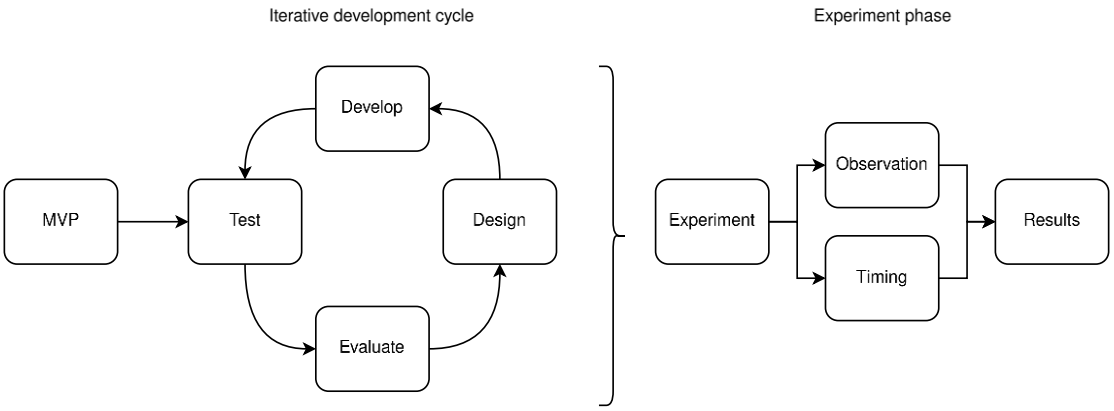

Figure 9. Overview of development cycle and experiment phase.

5 Results

The goal of this research was to compare the efficiency of the loaning application with a spreadsheet system. The results showed that on average Borrowtool was faster than the spreadsheet both when borrowing and returning items.

When looking at the individual results, the borrow times were always faster on Borrowtool than with the spreadsheet. Figure 10 shows the average borrow times taken by the seven participants for each of the items. Figure 11 shows the distribution of the borrow times.

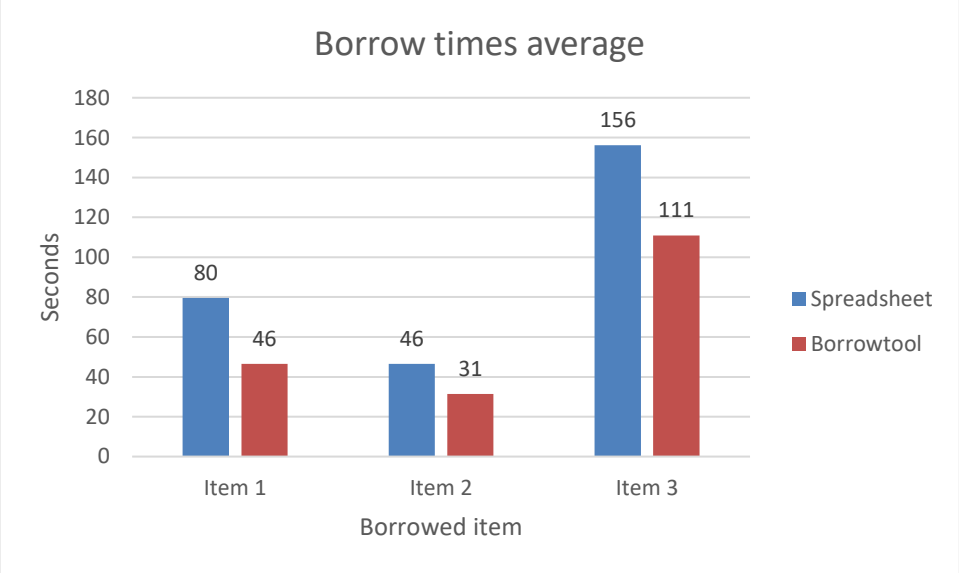

Figure 10. Borrow times average.

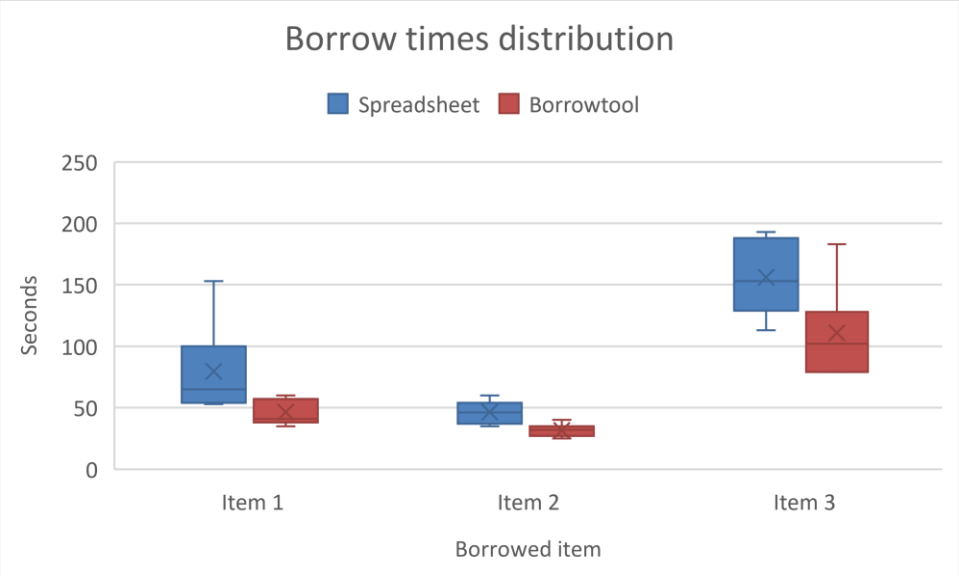

Figure 11. Borrow times distribution.

With returns the averages were also better with Borrowtool than with the spreadsheet. However, when comparing individual participants' times returning with Borrowtool and the spreadsheet, four out of 21 cases were faster with the spreadsheet. Figure 12 shows the average return times taken by the seven participants for each of the items. Figure 13 shows the distribution of the return times.

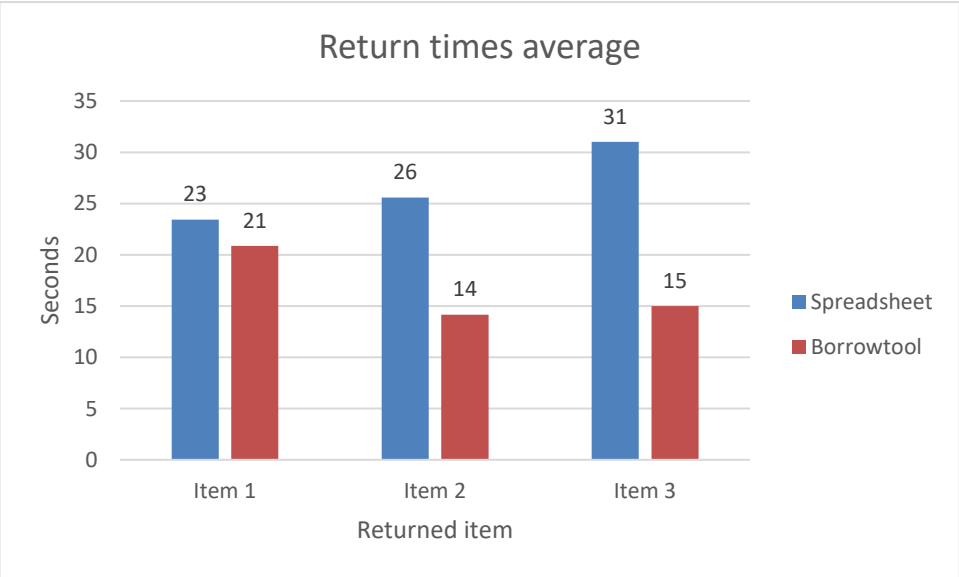

Figure 12. Return times average.

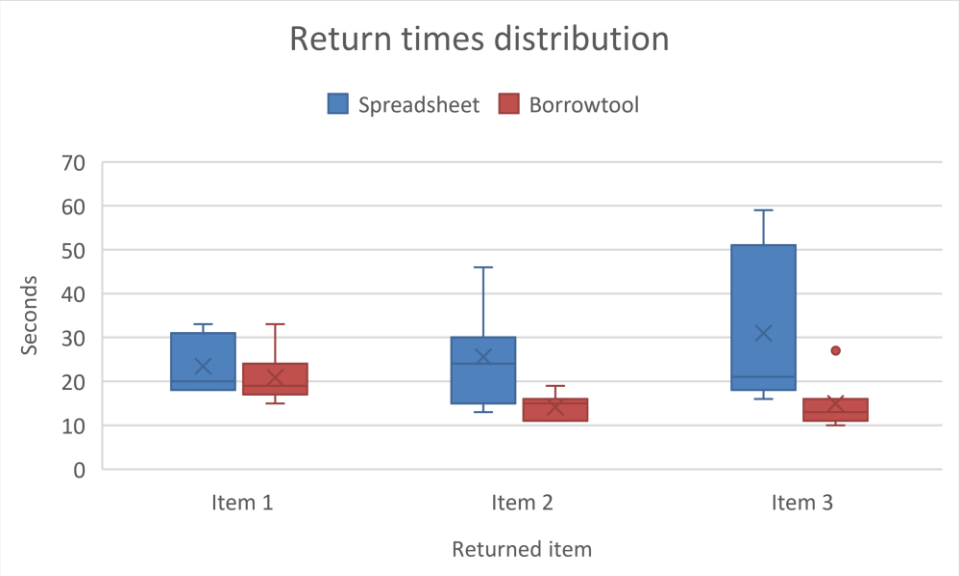

Figure 13. Return times distribution.

Paired single-tail t-tests to test the hypotheses that the borrow and return times of Borrowtool are faster than the spreadsheet system gives p-values of 0,0001 for the average borrow time and 0,0038 for the average return time. This indicates that the borrow and return times are significantly less with Borrowtool at alpha = 0,05. Table 1 summarizes the paired t-test for the overall average (combining borrow and return times) of using the two systems, confirming that using Borrowtool is significantly more efficient than using the spreadsheet.

Table 1. Summarized t-test for loaning out and returning.

t-Test: Paired Two Sample for combined borrow and return means		
	Spreadsheet	*Borrowtool*
Mean	60,35714286	39,78571429
Variance	2677,06446	1381,977352
Observations	42	42
df	41	
t Stat	4,991497034	
P(T<=t) one-tail	5,76359E-06	
t Critical one-tail	1,682878002	

6 Discussion

The results indicate that Borrowtool is faster to use than the spreadsheet which is in line with this study's hypothesis. The few cases where the spreadsheet was faster than Borrowtool were from participants who were very familiar with the spreadsheet and had a lot of experience using it. As the engineers keep using Borrowtool and better learn how it works, the results could improve even more.

What the data does not show is whether the participants made any mistakes. When using the spreadsheet, many of them forgot to fill in some columns, did not know if they should fill them, or did not even realize the columns existed since they were off their screen. This could result in a scenario where an item is loaned to someone, but the borrower is still marked as the previous borrower because an engineer forgot to overwrite it causing them to not have any way to contact the borrower. With Borrowtool, the required input fields are clearly marked and if they made a mistake or forgot to fill something in, the application would refuse their input and tell them where they made a mistake. This ensures that important information is never left out and that all of it is up to date.

It is important to note that while this study focuses on comparing the speed of Borrowtool to the spreadsheet, it is not the only benefit Borrowtool has over the spreadsheet. Borrowtool keeps a history of each item, allowing users to see who has previously loaned the item or see if the item has needed repairs in the past. With the spreadsheet, every time an item is loaned, the previous loan gets overwritten causing this information to be lost. The spreadsheet lacks the ability to easily search for items or get lists of borrowed or broken items which can lead to those items being forgotten about or lost. The spreadsheet lacks any error checking, and a user can accidentally delete data without realizing, potentially losing a borrower's contact information. These factors along with the speed improvements are why Borrowtool is superior to the spreadsheet and why it was created in the first place.

7 Conclusion

The first objective of this study was to develop the application through an iterative development cycle. Once the application was developed, the second objective was to compare its efficiency with the spreadsheet system.

The results of this study demonstrate that using the inventory management application called Borrowtool is more efficient for the FIT engineers than using a spreadsheet to manage inventory. The use of an iterative development cycle that combined both observation of users and their feedback helped create an application that was efficient to use and easy to understand. The qualitative user centric design and development optimized the quantitative efficiency of the software. The hypothesis was that the application would be more efficient to use than the spreadsheet and the results proved it correct. To gain a better understanding of Borrowtool's ease of use and efficiency, a further study should be conducted using participants who have no prior experience using either Borrowtool or the spreadsheet. This type of study would have less bias towards the spreadsheet and it could also show which of the two tools is faster to learn.

References

Atieh, A. M. et al., 2016. Performance Improvement of Inventory Management System by an Automated Warehouse Management System. *Procedia CIRP,* Volume 41, pp. 568-572.

Business Research Insights, 2024. Inventory Management Software Market Size, Share, Growth, And Industry Analysis, By Type (Manually Managed Inventory System, Barcode Scanning System, and Advanced Radio Frequency System (RFID)), By Application (Tablet, Mobile Phone, and Desktop), Regional. [Online]
Available at: https://www.businessresearchinsights.com/market-reports/inventory-management-software-market-110159
[Accessed 4 June 2024].

Cin7, 2024. *Cin7 solutions.* [Online]
Available at: https://www.cin7.com/solutions/core/
[Accessed 4 June 2024].

Dennert, K., Friedrich, L. & Kumar, R., 2021. Creating an Affordable, User-Friendly Electronic Inventory System for Lab Samples. *SLAS Technology,* 26(3), pp. 300-310.

GTK Team, 2024. *GTK project documentation.* [Online]
Available at: https://www.gtk.org/docs/installations/index
[Accessed 4 June 2024].

itemit, 2024. *itemit pricing.* [Online]
Available at: https://itemit.com/pricing/
[Accessed 4 June 2024].

Jahn, M., 2024. *ZXing.Net.* [Online]
Available at: https://github.com/micjahn/ZXing.Net
[Accessed 4 June 2024].

Katana, 2024. *Katana pricing.* [Online]
Available at: https://katanamrp.com/pricing/
[Accessed 4 June 2024].

Microsoft, 2023a. *Overview of ASP.NET Core.* [Online]
Available at: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0
[Accessed 4 June 2024].

Microsoft, 2023b. *Scaffolding (Reverse Engineering).* [Online]
Available at: https://learn.microsoft.com/en-us/ef/core/managing-schemas/scaffolding/?tabs=dotnet-core-cli
[Accessed 4 June 2024].

Microsoft, 2023c. *Windows Forms .NET documentation overview.* [Online]
Available at: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-8.0
[Accessed 4 June 2024].

Microsoft, 2024a. *Entity Framework documentation hub.* [Online]
Available at: https://learn.microsoft.com/en-us/ef/
[Accessed 4 June 2024].

Microsoft, 2024b. *Supported platforms for .NET MAUI apps.* [Online]
Available at: https://learn.microsoft.com/en-us/dotnet/maui/supported-platforms?view=net-maui-8.0
[Accessed 4 June 2024].

Nisha, F., 2018. Implementation of RFID Technology at Defence Science Library, DESIDOC : A Case Study. *DESIDOC Journal of Library & Information Technology,* 38(1), pp. 27-33.

Norman, D. A., 1988. *The design of everyday things.* 2002 ed. New York: Basic books.

Qml.Net, 2020. *Qml.Net - Qt/QML integration/support for .NET.* [Online]
Available at: https://github.com/qmlnet/qmlnet
[Accessed 4 June 2024].

Qt Wiki, 2023. *Language Bindings.* [Online]
Available at: https://wiki.qt.io/index.php?title=Language_Bindings&oldid=40847
[Accessed 4 June 2024].

Turku University of Applied Sciences, 2024. *Turku University of Applied Sciences Study Guide.* [Online]

Available at: https://opinto-opas.turkuamk.fi/21632/fi/21715/21719/1415/607
[Accessed 4 June 2024].

Unity Technologies, 2024. *Unity - manual: UI toolkit.* [Online]
Available at: https://docs.unity3d.com/Manual/UIElements.html
[Accessed 4 June 2024].