UiA University of Agder

Bachelor

OVERSEER, A 3D-PRINTER MANAGEMENT SYSTEM

A custom system to oversee and control the usage of 3D-printers for Signature Makerspace UiA.

ROBIN HANSEN & CHRISTOPHER HØGBERG HOLVIK

SUPERVISOR

Christian Auby

University of Agder, 2022

Faculty of Engineering and Science

Department of information and communication technology

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	**Ja**
2.	**Vi erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	**Ja**
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	**Ja**
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	**Ja**
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	**Ja**
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	**Ja**
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	**Nei**

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	**Ja**
Er oppgaven båndlagt (konfidensiell)?	**Nei**
Er oppgaven unntatt offentlighet?	**Nei**

Chapter 1

Abstract

Maintenance and usability are essential for 3D-Printing. 3D-Printers are quite expensive, so ensuring they last their full life time is desirable. This is especially true if someone is responsible for a multitude of 3D-Printers, such as makerspaces. Additionally makerspaces greatly value the quality of service they provide. Being useful and making sure their services are appreciated is highly advantageous. Overseer is a product meant to strengthen these attributes.

Overseer is a printer management system meant to make life easier for both managers and users of 3D-printers in makerspaces. It aims to assist managers to take good care of makerspaces and supervise use. Overseer also aims to give users useful information and facilitate better communication between users and admins.

Contents

List of Figures

Chapter 2

Glossary

.NET - ".NET is a free, cross-platform, open source developer platform for building many different types of applications"[1]

API - Application Programming Interface, allows for communication between software applications [2]

API consumption - Using an API [3]

ASP.NET core - ASP.NET is a popular web-development framework for building web apps on the .NET platform [4]

Back-end - The software part the user can't see. Focuses on giving functionality, storing data and making the application "run" correctly

Endpoint - "API endpoints are the specific digital location where requests for information are sent by one program to retrieve the digital resource that exists there"[5]

Enterprise Architect - A visual modeling and design tool [6]

Entity framework - "Entity Framework is an open-source ORM framework for .NET applications supported by Microsoft" [7]

Framework - Gives functionality and gives a solid foundation to build upon. Avoid starting from scratch

Front-end - Usually the visual portion of a product and the part the user is able to interact with

Gantt diagram - popular way to represent planned time usage[8]

Git - Version control, keeps track of changes in code [9]

Github - Used together with Git, extends version control over several users [10]

GPIOs - general purpose input/output

GUI - Graphical User Interface

IDE - Integrated Development Environment, Facilitates software development with code editor, debugger and much more

Jira - Project management tool [11]

JSON - JavaScript Object Notation, data-interchange format [12]

Makerspace - A place/environment where people can gather to create and design [13]

MAUI - Multi-platform App UI, a cross-platform framework for creating apps [14]

MicrosoftSQL - Database system created by Microsoft [15]

Non-functional requirements - Criteria that can be used to judge the use of a system, rather than the system itself [16]

Nswag - A Swagger/OpenAPI 2.0 toolchain [17]

Open-source - A code base the general public has access to

ORM - Object–relational mapping

Quality Attribute - "A quality attribute (QA) is a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders" [18]

REST APIs - An API following a set of agreed upon standards and styles. REST stands for "representational state transfer" [19] [20]

SQL - Structured Query Language

SSMS - SQL Server Management Studio

STL - Standard Tessellation Language, it is the file-type 3D-printers use to produce items

Swagger - API developer tool that allows for easy design, consumption and testing APIs [21]

TOS - Terms of service

UI - User Interface, software interacting with the user

Use-Cases - A scenario for a program to simulate usefulness [22]

User-stories - An informal explanation of how a user might want to use a system [23]

Visual studio - An IDE for Microsoft's .NET platform [24]

YAML - "YAML is a data serialization language that is often used for writing configuration files." [25]

Chapter 3

Introduction

The subsequent report entails the process of choosing, planning and developing the product "Overseer" for Signature Makerspace, UiA, as well as the final result. Overseer would help both managers and users of Makerspaces oversee certain information more efficiently.

3.1 Background

While 3D-Printers are far from readily available in every household today, they have been steadily increasing in popularity as the technology has been furthered developed and adopted by consumers. As such, a trend in recent years have become to bundle a collection of 3D-Printers for people to create objects for their work, hobbies or studies. These work spaces are most often refereed to as "makerspaces". Makerspaces can vary widely in size and complexity, being found in libraries, universities and privately owned businesses, but most of them contain some form of 3D-Printing. [13]

Signature Makerspace is a student organisation within UiA and like most other makerspaces hosts a variety of different 3D-printers, laser cutters and other high-tech equipment [26]. Maintaining this equipment and being informed of exactly who is using what equipment is quite important. Knowing which users are using specific equipment is quite useful in case there is a need to contact that person. For instance: if their 3D print falls over, stops or if the user is miss-using the equipment.

The printer management system employed at Signature Makerspace was a whiteboard, hanging next to the printers, for users to write their names, Printer-ID and contact information whenever they used the printers. This system mostly went under-utilized and simply ignored over time. Users did not really seem gravitate towards the system, managers had little to no idea when certain entries on the board were made. The system also lacked a backlog of usages of the 3D-printers over a reasonable amount of time.

3.2 Project Objective

With the previous mentioned deficiencies in mind, a solution was needed to remedy the situation. The solution in-mind would firstly have a clear focus on overview of printers and being consistently able to know which printers were available and which were unavailable. This would be of great help to the student-admins managing the system, letting them know exactly when and which printers were in use. This might also facilitate a more efficient usage of the printers, with users having a clear summary of all printer's status and being able to plan accordingly.

Secondly, it would require an easy method for notifying students using specific printers. The reason for this is; as 3D-prints usually take a long time to complete, the user normally is not overseeing the actual work. Therefore if the print were to fall over (Which is more common than you think according to Signature Makerspace) or otherwise stops, the user would not be able fix this issue while they are away doing something else. Being able to notify the user before they return to find a unsuccessful 3D-print would be highly beneficial for both the student-admins managing the system and those using it. Another beneficial side-effect would be the immediate notification towards users miss-using printers. If a particular users approach to the 3D-Printers was unwanted or their prints displeasing to management, the print could be stopped while also notifying the user and the reason(s) for its removal.

The last primary goal of the solution would be a sufficient printer-usage backlog. With students responsible for managing the system not being able to oversee every print that occurs, a backlog would be needed. This way if an error detrimental to the 3D-printers were to happen during an unsupervised print, the managers would be able to look at the printer's history and get a better idea of exactly who was responsible. The person in question could then be informed of their mistake and/or warned concerning their future prints.

3.2.1 Employer requirements

The employer had a set of specific requirements they wished completed, and a set of requests they wanted, but did not consider essential. After conversing with the employer and working out a possible solution that included the most amount of their wishes, we ended up agreeing to these essential features:

- User-management
- Log-in system (with student-cards if possible).
- History over 3D-printer usage.
- Custom UI using Signature Makerspace's colors.
- The ability to add/customize/remove printers and users.

Chapter 4

Method

From the inception of the project it became clear that we would need a lot information to formulate a sufficient plan. While we were excited to work with the makerspaces and 3D-Printers, we had little to no prior experience with either. This was also one of the first projects directly made for a "customer" and not directly for ourselves. We knew from the start that it would test our ability to understand our customers wishes and we would need to formulate a plan accordingly.

Our first priority was to accumulate enough information about 3D-Printing, makerspaces and Signature Makerspace's wishes for the assignment. While 3D-Printing as a concept was known to us, the intricacies of exactly how it works were unknown. Questions such as: "How does a person print out an object?", "Do users deliver the file over the internet, Bluetooth or physically?", "How customizable is an average 3D-Printer?", "What exactly is a Makerspace?" and many more questions lingered.

Most of the questions were answered with a quick search, but we still had great lapses in knowledge. Specifically pertaining to what Signature Makerspace explicitly wanted out of the project. While the initial project description explained the problems they were experiencing, it was not clear as to exactly how they wanted those issues fixed. The text made clear wishes for a potential scanner-terminal, information of students using the printers and time-remaining, but not exactly how they wanted it implemented. Do they wish for a solution that is strict, one which locks the printer usage until students opt into the management system, or is a incentivized system desirable?

The focus in the beginning was to gather information on both the technology used and what the end-user would potentially want. Considering we were not an average user of makerspaces, this was mostly done by discussing with managers of Signature Makerspace.

Out of these discussions with the managers and ourselves, grew a set of requirements, functions and ideas of prototypes. After getting a sense of what they envisioned, we eventually made clear what was out-of-scope for the project, discussed our ideas and eventually began working on testing out our ideas. Ideas was mostly tested by creating diagrams, wire-framing or models to essentially test the ideas for free.

With the ideas becoming more clear with each iteration, we then had to decide how we would use them going forward. Our first priority would be to make a set of goals to follow throughout the entire implementation process, these would function as a good base to rely on. This foundation would in turn make sure we always stayed on track.

Our implementation phase would be structured by splitting it into individual weeks, with each week having two parts. One part planning and the other part working while we made sure to follow the initial plan we agreed on. Since we would be meeting our supervisor every week to show of progress and get feedback, we decided to do our planning on the same day as well. After getting feedback and a general sense of the previous week's timeline, we would plan the next week's period accordingly. Either adding new features to start work on, or finishing incomplete ones.

While we initially considered following a similar implementation method just with a bi-weekly format, we eventually settled on weekly schedules. Structuring for individual weeks would allow us to be much more flexible when handling tasks. Firstly, splitting assignments into smaller and more achievable tasks to fit into our week long period would simplify the process. Additionally, our view was that the feedback from our supervisor would be more valuable if it was given on a weekly plan, instead of a bi-weekly plan. If feedback was given every week on a bi-weekly plan, repetition of similar topics might be more likely as feedback would be given on the same plan two weeks in a row.

Chapter 5

Process

This chapter will go into detail about why we chose the tools and methods in the planning phase for this project.

5.1 Project method

While there are quite a few types of project methods we could have chosen, in the end we decided upon the Agile method. The reason behind this choice was being able to push functionality to the user faster and being more flexible to change. We did not want to work on the software solely "in house" and receiving minimal amount of outside feedback before one big final release. The project would instead have incremental releases to show of to the users for feedback.

5.2 Agile method

The upside to Agile is being able to push functionality to the user faster and being more flexible to change. Sprints are set dates in which certain features are planned to be finished and shipped to the users. This ensures good flexibility in projects.

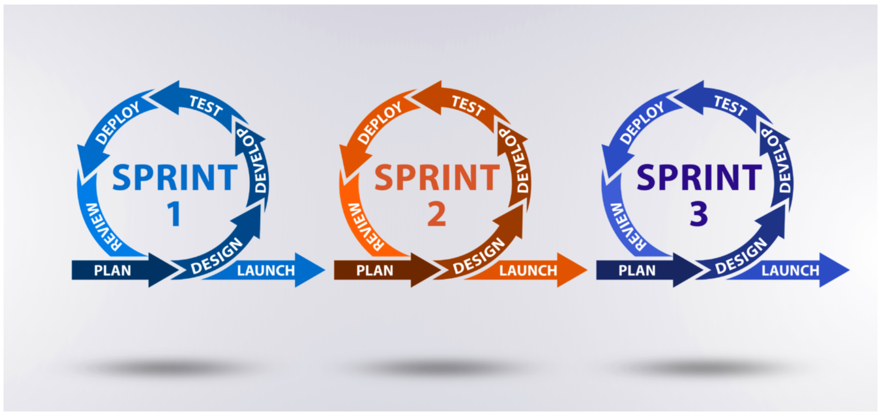


Figure 5.1: Visualization of the agile method

Image taken from [27]

Though some of the downsides of the Agile method may be dampened with experience, outside factors or good leadership, they will still impact the product in some way. Some of the biggest disadvantages of the Agile method could be [27] [28]:

- **No finite end** - Since the Agile method does not set a final end date and stays more fluent, it can be harder for users to keep focused on what the end project would be like. With no finite time limits users can easily find themselves going on long side-tangents in the project, focusing on something which in the grand scheme of things might no be so important. It is also more difficult to imagine what the final product may look like when there is no immediate end date.

- **Fragmented products** - Since the product is being worked on in several smaller "chunks" and not as one large project, the final version may feel somewhat fragmented. Several, individual pieces of the program will most likely have been created separately from each other, leaving the program with far lower cohesion than a normal program might have.

- **Unexpected difficulty and difficult planning** - Since the project gets worked on in smaller "chunks" it can be hard to accurately judge the difficulty and time requirement on some sub-tasks. This can lead to major delays in the project, especially if later tasks are dependent on the sub-task causing the delay.

- **Limited and last minute documentation** - Since user documentation happens whilst working on the project instead of before the project even starts, this can lead to less detailed documentation that more often than not fall on the backlog to either be done after the fact or never at all.

5.2.1 Jira

We agreed to use Jira for our project management tool early on in the planning process. We had extensive experience with the platform as a result of previous tasks and projects, additionally we had quite a positive opinion on Jira. While other options such as "Asana [29]" were weighed, Jira was quickly utilised for further planning and management due to our familiarity and satisfaction with the product.

5.2.2 Time-Tracking

The group would time-track and use time estimation for sprints and tasks when ever possible. This was to help the development phase stay on track and make issue assignment more realistic and effective.

5.3 Git/GitHub

The group would utilize Git, and a shared private GitHub repository. Git is an extremely popular tool in programming used for version control. Git would allow us the use of saving file-states in repositories with the use of "branches". All repositories have a main branch being the main version of the repository, but it also allows for the creation of other branches. The other branches do not effect each other and allows for separation of states. This would be quite useful when implementing features, because if something goes wrong we would always have our main branch to fall back on. Since Git also allows for the merging of branches, if a branch used for development was a success and stable, we can simply merge this branch into main or any other branch the we might desire. [9]

Github extends this functionality to involve multiple developers by having them work on a remote repository hosted on their website. We maintain the same functionality as previously mentioned, but instead of merging our branch onto another local branch, we can instead create a pull request for merging our branch onto one of our remote branches. As well as allowing us to merge into the remote main branch. The pull request also allows for other functionality as code review. Before the branch is merged into main, we can review that branch to ensure that the quality is acceptable.[10]

5.4 Enterprise Architect

As we were following the Agile method [27], a lot of diagrams would have to be made. For this we would utilise Enterprise Architect (EA). While we could have made the diagrams in writing or drawing, we thought it would be more appropriate and easier to use EA. EA would allow us to quickly test out diagrams and share these with other developers or our employer. EA also contains a lot of templates that would prove useful throughout the planning process [6].

5.5 IDE

Visual studio is a free IDE for Microsoft's .NET platform. It is used for many different purposes such as websites or apps. It is easy to use, is well supported with a large amount of documentation on the internet, the community version is free to use and much more [24]. Visual studio offers several different programming languages to work with, such as CSS, C#, JavaScript and more [30].

5.6 Gantt schema

A Gantt schema is popular way to sort tasks. It displays tasks on the y-axis, against time displayed on the x-axis. It serves as a useful tool to manage time, both in preparation and retroactively [8]. This project utilized a Gantt schema.

Chapter 6

Requirements

With the employers wishes in mind, we then began formulating concrete requirements that the project had to fulfill for it to be considered done. Doing so took a fair amount of time and great amounts of discussion in the group, but led to a solid foundation of requirements which we could use for the future.

6.1 User-stories, use-cases & non-functional requirements

6.1.1 Customer collaboration

To be able to fully understand and fulfill the customer's wishes in regards of the project, a great deal of collaboration and revision of ideas would have to be done. This came in the form of meetings and discussion of ideas, desires and what current deficiencies the project would ideally solve. After which these would be focused down to more clearly defined user stories, use-cases and requirements.

Once both developers and customers were satisfied with all functional requirements, we set out to decide upon the quality attributes that would be our non-functional requirements. With our goal being to uphold these requirements at every step throughout our project. To ensure this, we defined our quality attributes into actually testable non-functional requirements

6.1.2 User-stories

Following the Agile method [27] we first tried to emulate the mindset of an average user, and what systems the user might interact with. In doing so we created several user-stories based on what the average user would do in order to operate our system. After reaching five user-stories we decided to change the perspective from that of an average user, to that of an admin. We then attempted to create several user-stories that only apply to the admins, in which we created six user-stories. This was done under the assumption that everything an average user might want to do, the admin is also able to do. Therefore the list of admin duties and requirements where far larger than that of the average user. Finally we tried to find edge cases, something the average user would not normally interact with, such as refilling plastic or re-calibrating the printer and then creating the final user-stories with these in mind.

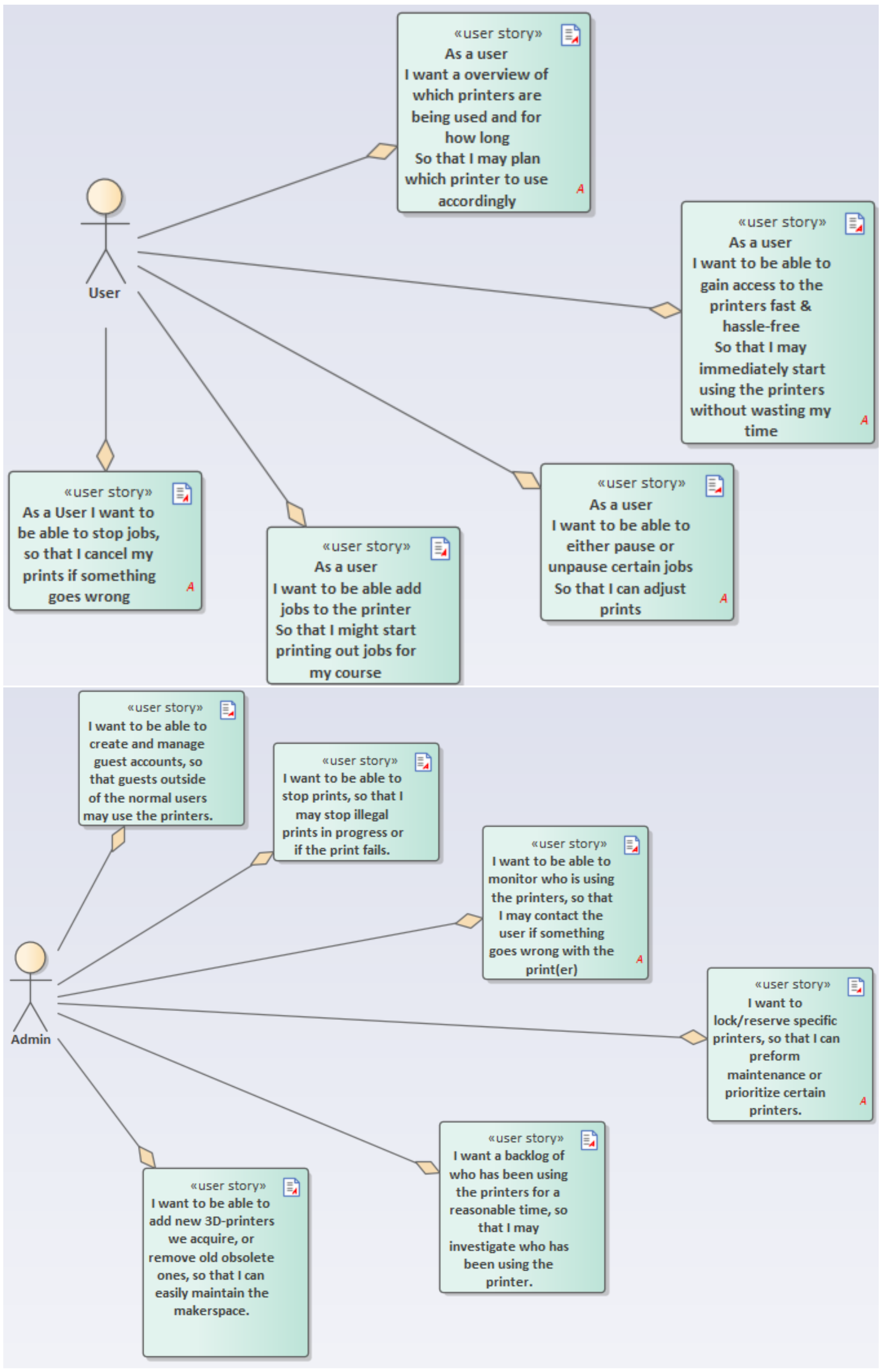

User
«user story» As a user I want a overview of which printers are being used and for how long So that I may plan which printer to use accordingly
«user story» As a user I want to be able to gain access to the printers fast & hassle-free So that I may immediately start using the printers without wasting my time
«user story» As a User I want to be able to stop jobs, so that I cancel my prints if something goes wrong
«user story» As a user I want to be able add jobs to the printer So that I might start printing out jobs for my course
«user story» As a user I want to be able to either pause or unpause certain jobs So that I can adjust prints
Admin
«user story» I want to be able to create and manage guest accounts, so that guests outside of the normal users may use the printers.
«user story» I want to be able to stop prints, so that I may stop illegal prints in progress or if the print fails.
«user story» I want to be able to monitor who is using the printers, so that I may contact the user if something goes wrong with the print(er)
«user story» I want to lock/reserve specific printers, so that I can preform maintenance or prioritize certain printers.
«user story» I want a backlog of who has been using the printers for a reasonable time, so that I may investigate who has been using the printer.
«user story» I want to be able to add new 3D-printers we acquire, or remove old obsolete ones, so that I can easily maintain the makerspace.

Figure 6.1: User-stories

6.1.3 Use-cases

Using a similar method to what we did when creating user-stories. We created a base diagram in Enterprise architect and used our user-stories as a foundation as to what we would use for the use-case diagram. We divided all possible actions into two groups, "user" and "admin". Then, using Enterprise architect, we created the full diagram containing all the user-stories that were created earlier in the project.

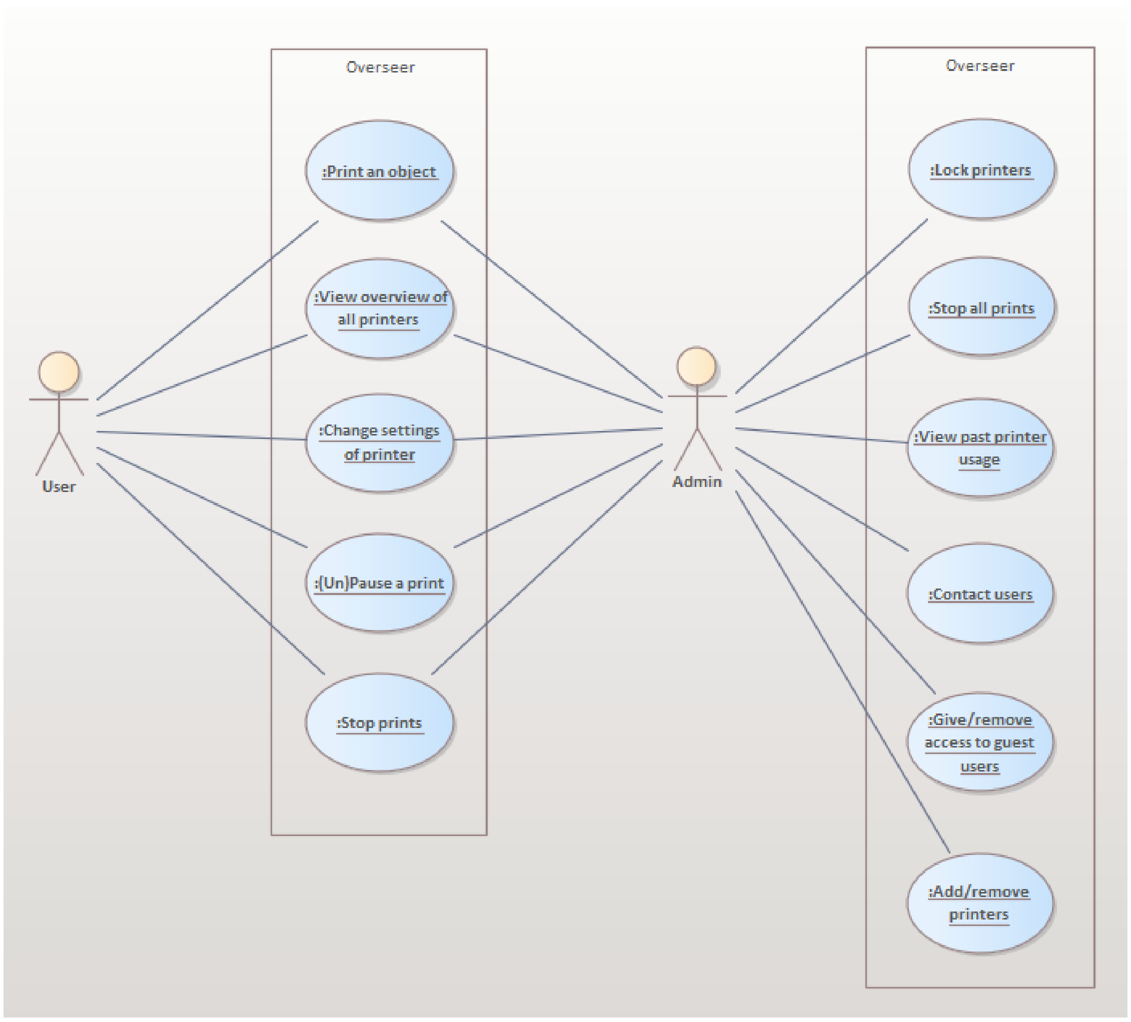
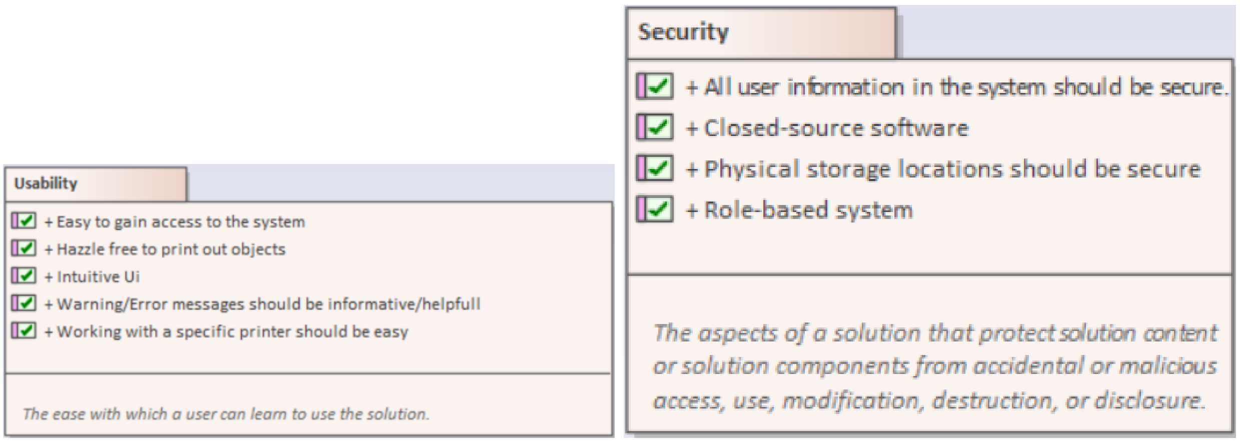


Figure 6.2: Use-cases

6.1.4 Non-functional requirements

The last set of product requirements initially decided upon were non-functional requirements. These "quality attributes", are almost as important to get right as functional requirements as they dictate how well the end product ought to satisfy the user. These were decided by discussing what quality attributes the end project would ideally have, which of these attributes we would be prioritising in development and what non-functional requirements would be necessary to appease the customer.



Usability

- + Easy to gain access to the system
- + Hazzle free to print out objects
- + Intuitive Ui
- + Warning/Error messages should be informative/helpfull
- + Working with a specific printer should be easy

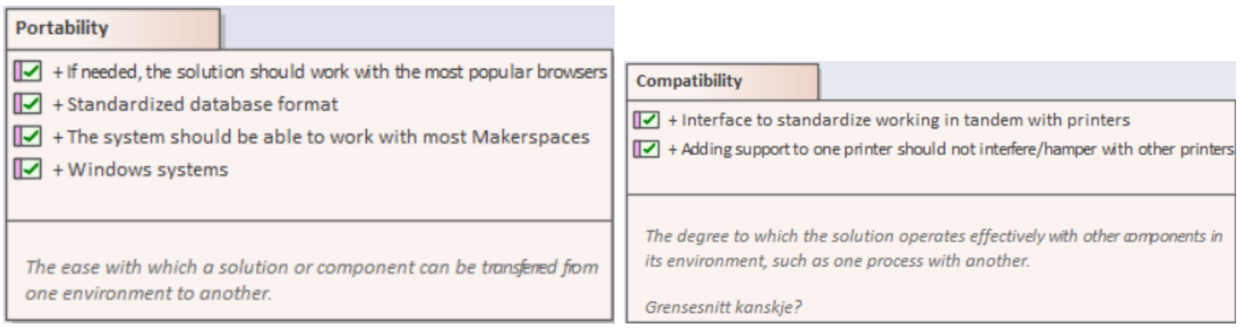
The ease with which a user can learn to use the solution.

Security

- + All user information in the system should be secure.
- + Closed-source software
- + Physical storage locations should be secure
- + Role-based system

The aspects of a solution that protect solution content or solution components from accidental or malicious access, use, modification, destruction, or disclosure.

Figure 6.3: Non-functional requirements usability and security



Portability

- + If needed, the solution should work with the most popular browsers
- + Standardized database format
- + The system should be able to work with most Makerspaces
- + Windows systems

The ease with which a solution or component can be transferred from one environment to another.

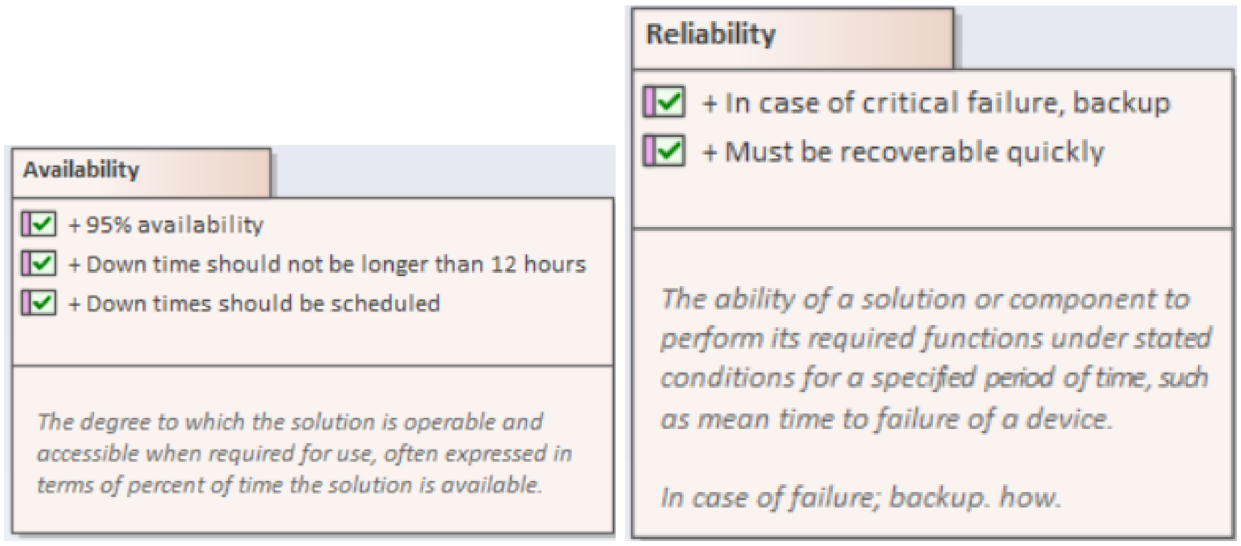
Compatibility

- + Interface to standardize working in tandem with printers
- + Adding support to one printer should not interfere/hamper with other printers

The degree to which the solution operates effectively with other components in its environment, such as one process with another.

Grensesnitt kanskje?

Figure 6.4: Non-functional requirements portability and compatibility


Availability

- ✓ + 95% availability
- ✓ + Down time should not be longer than 12 hours
- ✓ + Down times should be scheduled

The degree to which the solution is operable and accessible when required for use, often expressed in terms of percent of time the solution is available.

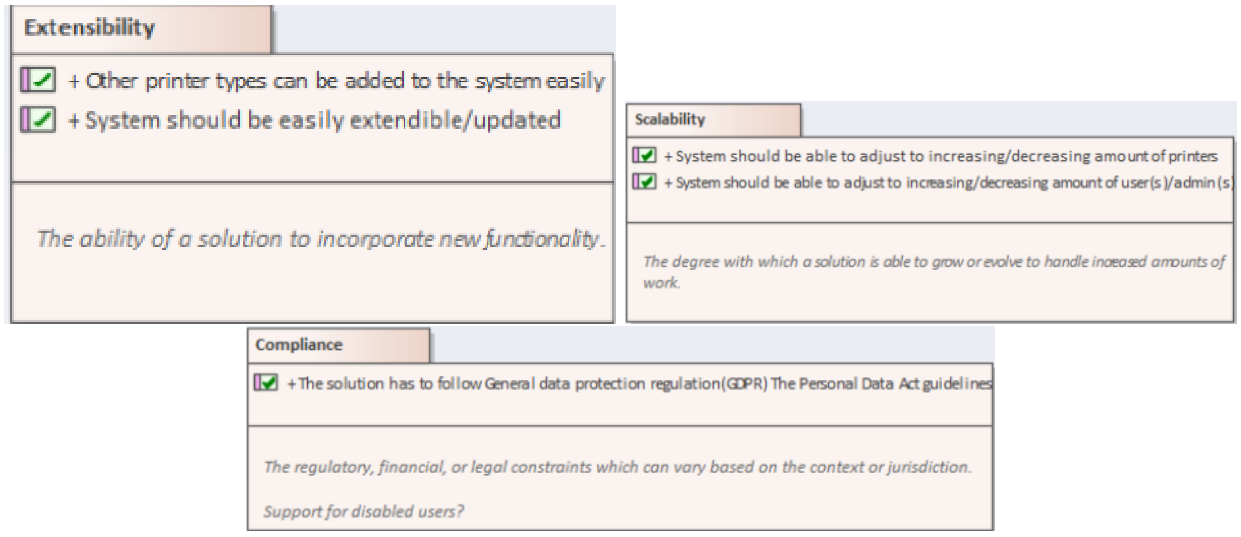
Reliability

- ✓ + In case of critical failure, backup
- ✓ + Must be recoverable quickly

The ability of a solution or component to perform its required functions under stated conditions for a specifed period of time, such as mean time to failure of a device.

In case of failure; backup. how.

Figure 6.5: Non-functional requirements availability and reliability


Extensibility

- ✓ + Other printer types can be added to the system easily
- ✓ + System should be easily extendible/updated

The ability of a solution to incorporate new functionality.

Scalability

- ✓ + System should be able to adjust to increasing/decreasing amount of printers
- ✓ + System should be able to adjust to increasing/decreasing amount of user(s)/admin(s)

The degree with which a solution is able to grow or evolve to handle incresed amounts of work.

Compliance

- ✓ + The solution has to follow General data protection regulation(GDPR) The Personal Data Act guidelines

The regulatory, financial, or legal constraints which can vary based on the context or jurisdiction.

Support for disabled users?

Figure 6.6: Non-functional requirements extensibility, scalability and compliance

After deciding upon non-functional requirements, there was an additional demand to further elaborate on them. This would be obtained by transforming our non-functional requirements goal-like format into a more testable format to have a less abstract and more obtainable end-goals.

6.1.5 Testable usability non-functional requirement

There are too many testable non-functional requirements to show of each one. Instead we will specifically examine the testable "usability" requirements, to give you a better idea of how we created our testable non-functional requirements.

- **Easy to gain access to the system** - The average time used to gain access to the system should not exceed 2 minutes.
- **Hazzle free to print out objects** - A prepared user should be able to login and print out their object within 5 minutes on average. A prepared user is defined as a user with previous experience and a functional .stl file. Experience is defined as having used the system at least 3 times.
- **Intuitive UI** - A user after being asked to find a specific function within the system, should not spend more than 1 minute find it and how to utilize it. This 'specific function' could for instance be a highly used setting within the program, the overview or print. Essentially highly elements within the system.
- **Warning/Error messages should be informative/helpful** - A user should within 1-2 minutes of being presented an error message know what the issue is.
- **Working with a specific printer should be easy** - The added time needed to work with a specific printer(Assuming it does not need calibration, refill or change of settings.), in contrast to working with any printer at all, should not exceed a 20 seconds increase in time.

6.2 Layout

For the layout of our program we decided upon a simple and rather minimalist design. To navigate the program a user would simply use buttons located at the top of the page to switch between the different views we implemented. After a bit of testing and designing we eventually came up with this simplistic design:

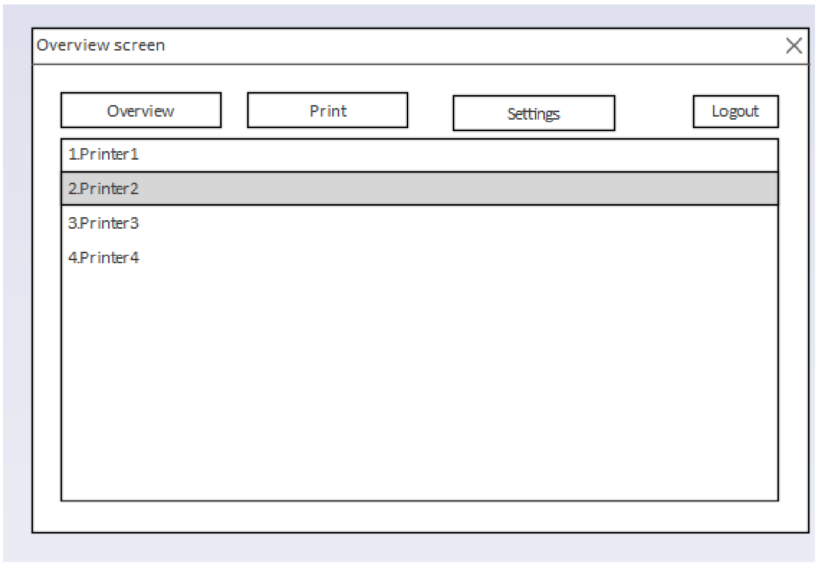


Figure 6.7: Concept for the "overview" page

Of course this was just an example of what the final product could look like, but this was more like the "skeleton" we used as a starting point when we began designing our program. This design forgoes a more fanciful style in favor of a clear and easily readable design. We wanted to avoid over designing the page to the point where a user might find it confusing or cumbersome to find the function they are looking for. We specifically wanted the user to only spend a maximum of 2-3 clicks to find said function. We hoped that the UI would be intuitive enough for the average user to be able to use our program without having to ask for assistance.

6.3 Project priorities

We intended to do the job in four phases. We used phases to indicate both our progress and priority during our work. We found it was more beneficial to finish phases in the correct order. We would be pushing out prioritized functionality to the end-user and making sure it working as intended, instead of adding less prioritized features and jumping straight to the total end goal. We intended to go "step by step", thus ensuring we would be able to deliver a finished product even if it had less functions. This ensures that we do not work on larger amounts of work than we could handle at once, which would lead a messy and most likely dysfunctional final product. This clearly reflects the agile method, which the project is built around, focusing on the functionalities that is needed first and then adding additional functionalities later.

6.4 GDPR, TOS and the privacy act

One of our employer's requirements was a backlog of users that has used Overseer. To do this we decided to keep a log of which students used what printer and at what time it was used. Since the users have to register with their email, name and phone-number we would simply tell the system to save the exact time the printer was set to "active". However keeping personal information of users have important laws behind them [31] [32] [33]. General Data Protection Regulation (GDPR) is a regulation for keeping the personal information of users safe by enforcing laws and rules on whom exactly can keep personal information, and for how long [34]. Therefore it was important that our system follow all Norwegian laws centered around privacy and the right to deletion [35]. The right to deletion ensures that programs/companies/businesses/etc is not allowed to keep personal information on the user if the user requests the information to be deleted. There are exceptions to this rule, but generally users have a right to be forgotten if they so wish. We intend to create a "Terms of service (TOS)" that allows us to keep the users information stored until the users requests it deleted from the system. Additionally GDPR protects the user from being "profiled". Profiling entails companies taking the personal information of users and either selling them out to companies which then create personalized adds, or by singling out individuals based on race/religion/gender/etc. This has been heavily restricted in the EU since 2018 and safeguards the users against malicious use of their information [36].

We intend to keep the account-details for a maximum of one year assuming the user agrees to the TOS. Alternatively the user may delete their account at any time. Additionally whenever a user uses a printer, the usage of said printer along with their information will be stored separately for one week. After one week, that printer usage record would be deleted from the system permanently. Storing printer usages for one week is done so that if the user somehow destroys/damages a printer or equivalent equipment, they will be logged in our system. This safeguards the makerspace admins against sabotage, both accidental and predetermined. Additionally we need to be able to contact the user is something goes awry with their print, for example it might fall over.

The information we save is:

- **Name** - Our basis for keeping the user's name is so that we can tell who is who in our data-system, more specifically who is responsible for a specific print. We cannot allow the creation of an account unless we know who agreed to the TOS [31].
- **Email** - Our basis for keeping the email is so that we may contact/notify the user. Since management is an important part of Overseer, the ability to communicate with its users is paramount.
- **Phone-number** - Our basis for keeping the user's phone-number is so that we may contact the user considerably faster and more efficiently than using the email. However the phone-number is not required and is an option for users who desire faster contact/warning.

Chapter 7

Detailed planning

With requirements set, phases decided upon and frameworks chosen, we could finally plan how we wanted the physical setup to look like for each phase.

7.1 Phase one implementation

Phase one focused on creating an incentivized system that does not connect to the printers in any way, simply incentivizing the users to sign into the system before using the printers. What we mean by incentivized is simply a system where nothing directly forces the user to interact with the framework. Instead simply incentivizing the use of the framework through signs, messages from the admins or light punishment for not going through the proper channels. In theory nothing is stopping the user from simply using the printers as normal, but administrators will monitor and ensure the system gets used to log and oversee printer activity. Therefore the program will simply be a logging service on the side, more as a management system to help the administrators. For users to log-in for documentation, we

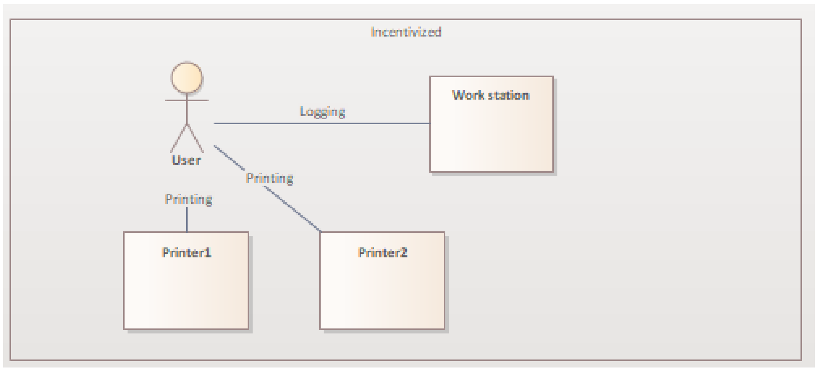


Figure 7.1: Phase one - Incentivized

would have a terminal in the Makerspace room, next to the printers. This would either be a simple PC/laptop or a Raspberry Pi connected to a screen, keyboard and mouse for users to interact with the application. This terminal would be running our front-end .NET MAUI application, allowing users to see the printer overview, register printer-usage or report issues.

The .NET MAUI application would further communicate with our back-end ASPNET Core hosted on a remote server. This way no user could simply steal the PC/Pi and gain sensitive user data in the process, since that data would be stored securely elsewhere.

An additional Raspberry Pi could be mounted elsewhere, connected to another screen. This setup would function as the central overview screen for students and users to check all printer's current status. This setup would most likely have no keyboard/mouse connected, as its purpose would be to simply show the overview portion of the application.

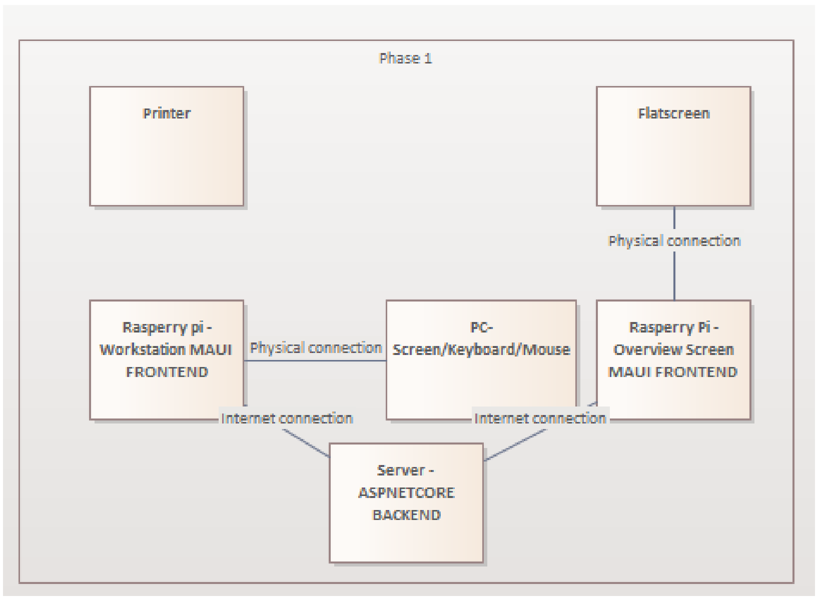

Phase 1
Printer
Flatscreen
Physical connection
Rasperry pi - Workstation MAUI FRONTEND
Physical connection
PC- Screen/Keyboard/Mouse
Rasperry Pi - Overview Screen MAUI FRONTEND
Internet connection
Internet connection
Server - ASPNETCORE BACKEND

Figure 7.2: Physical setup of phase one

7.2 Phase two implementation

Phase two will connect the printer's storage system to our system, but it will not physically stop the users from using the printers. Connecting the printers to our system will allow the users to sign into Overseer, and then send the desired "STL" file to the printer, from there they may start the print at the printer. This improves upon phase one by allowing the users to access the printer's storage from the workstation, instead of manually connecting their storage device to the printer itself. This could theoretically decrease the amount of work needed to get a printer booked and started, and it would be far easier for the administrators to oversee the system as a whole. However, outside of physically blocking access to the printers SD-card/flash drive input nothing is stopping the user from accessing the printers as normal, without logging it in the system. Such behaviour would be de-incetivized by the administrators cancelling any prints currently in progress that is not registered in the system.

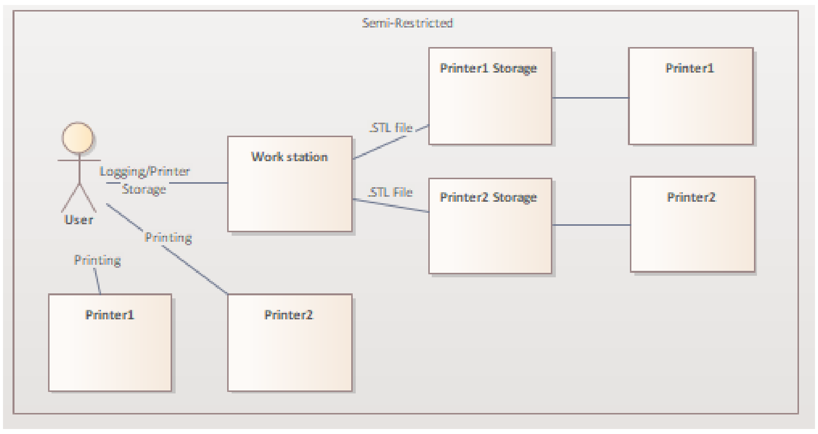


Figure 7.3: Phase two - Semi-restricted

We wanted a general solution for interacting with most printers, no matter if they were new, old or different models. Thankfully most printers have a way of connecting a storage device to them, mostly SD-cards or flash-drives. Our plan would be for the user to insert their own flash-drive/SD-card in the terminal when selecting a printer for usage. The user would then be able to choose that file when selecting files to print with the 3D-Printer. We had 3 options in mind to carry out this plan.

- **Physical external storage** - The easiest solution we came up with would be an external hard-drive which all printers would be connected to. When selecting .STL files to print with specific printers, users would pick from this storage device. Our system could then manipulate what files would be stored on this external device, based of the files given by the users in the terminal.

- **SD-cards/flash drives with Bluetooth connection** - Another idea we came up with would be either SD-cards/flash drives with the possibility for Bluetooth connection to the terminal. When a user selected a printer and gave a .STL File, our system would then send that file to the SD-card/flash drive corresponding to the corresponding printer.

- **SD-cards/flash drives with WiFi connection** - Lastly was the idea to use SD-cards/flash drives with the possibility for WiFi connection. Allowing us to use the same setup idea as Bluetooth. With a user selecting a printer and giving a .STL file, our system would then send that file to the SD-card/flash drive corresponding to the correct printer.

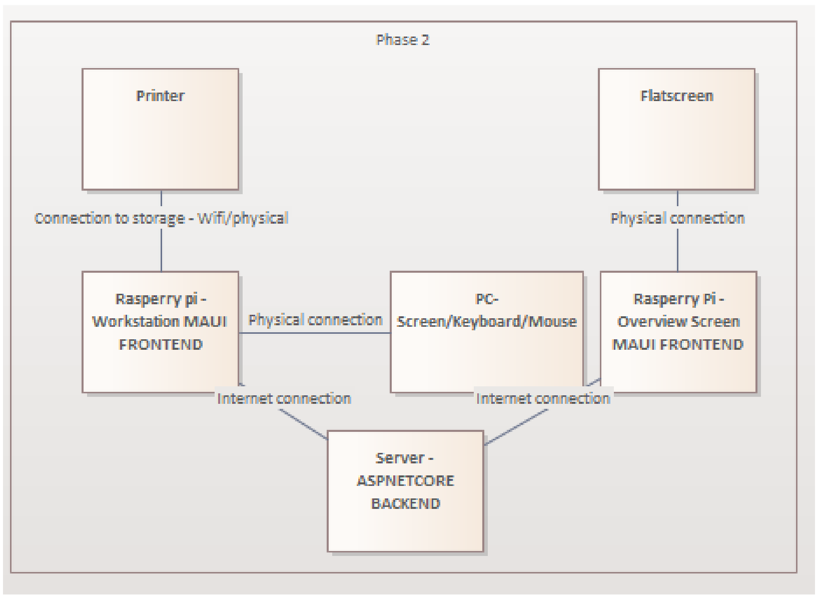

Phase 2
Printer
Flatscreen
Connection to storage - Wifi/physical
Physical connection
Rasperry pi - Workstation MAUI FRONTEND
Physical connection
PC- Screen/Keyboard/Mouse
Rasperry Pi - Overview Screen MAUI FRONTEND
Internet connection
Internet connection
Server - ASPNETCORE BACKEND

Figure 7.4: Physical setup of phase two

7.3 Phase three implementation

Phase three is theoretically the "smallest" phase in its scope. This phase aims to add a web-page where users and admins can check the system from any browser on any computer. Users would also be able to reserve printers so they know they are ready for when they need them.

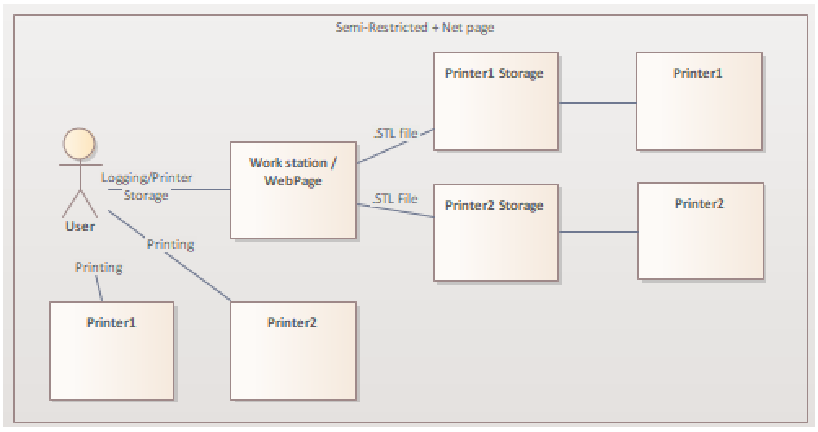


Figure 7.5: Phase three - Semi-restricted + web-page

We were not quite satisfied as to how we would be able to show overviews of printers. If we wanted to show a current status of printers anywhere else not near the makerspace, we would have to install our software onto a machine. This seemed a bit to restricting in terms of making the information as readily available as possible.

The purpose of the web page would be largely to replace the overview screens running the application. Instead of having to install the application on these machines, they could instead simply open up a web-page to show the current status of printers. This web-page would be very simple in form and design, it's main intention being to give information on printers. Since it is a web page this could be accessed by anyone, anywhere. Users could check availability of printers while walking to school for instance.

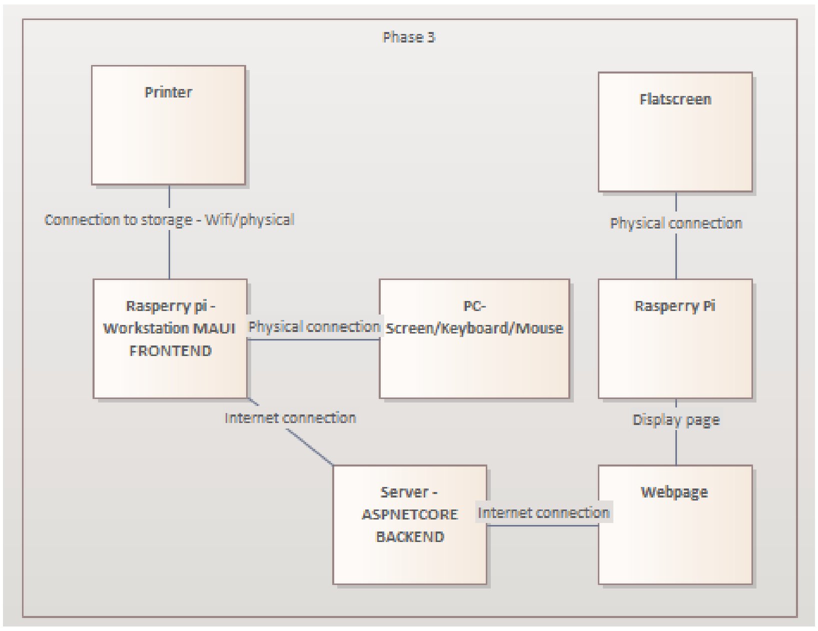


Figure 7.6: Physical setup of phase three

7.4 Phase four implementation

Phase four will build upon phase two and three by actually stopping users from accessing and using the printers outside of our system. This would force users into using the system, thus ensuring full oversight over any usage of the printers. As of right now phase four seems unlikely, due to limitations and restrictions with the printers themselves(and variations in models), so our focus will mainly be with phase one, two and three.

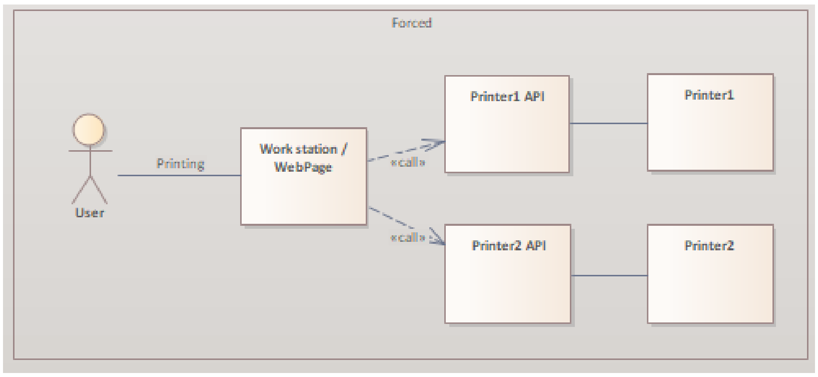


Figure 7.7: Phase four - Forced

This phase would directly connect our PC/Pi to each printer. Users would still select which printer to use, and which file to print, but instead of starting the print on the printers themselves, our program would do that for them. Working with the Printer's API we would start the print for the user and remote-control everything about printing.

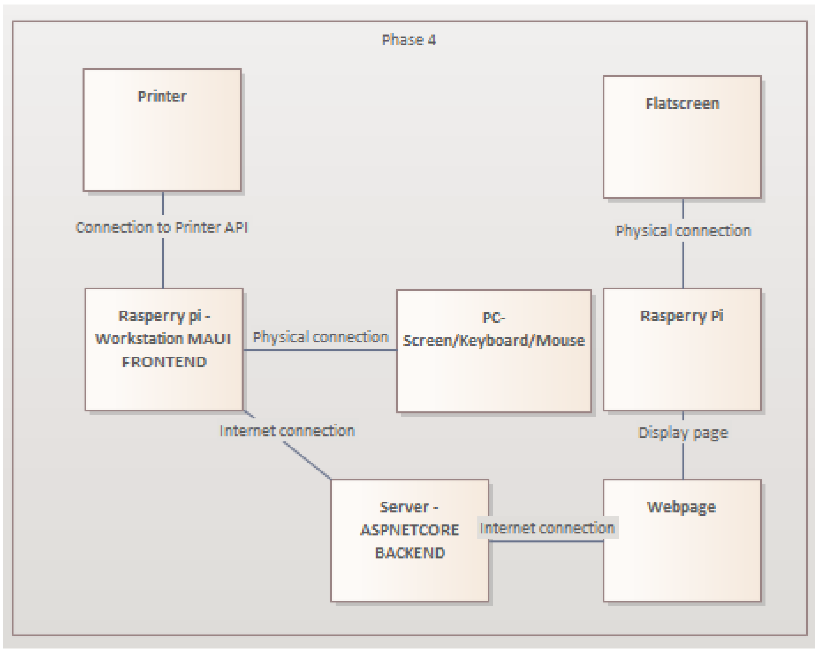


Figure 7.8: Physical setup of phase four

Chapter 8

Technical Design

8.1 Code structure

After discussion between group members, it was decided we would have clear distinctions between the front-end and back-end in the project. Having only visual elements in our front-end and processing all information and logic in a back-end working independently from the front-end. The biggest reason for this was obviously security. If the program was ever to process user information in the front-end, that information might be readily available for anyone using this application. This would severely hamper what we would be allowed to do. It also structures the application in a neat way. Clearly separating functionality from the visuals.

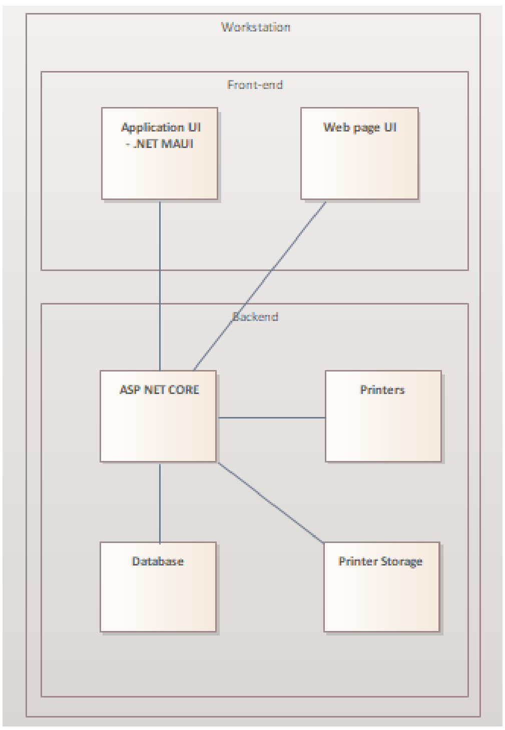


Figure 8.1: Initial concept of finished product structure

It was also decided that any interaction with a database, separate printers, printer storage or with a web page would be done through the back-end handling all the logic.

8.1.1 Chosen frameworks and database system

The front-end section would use .NET MAUI as a foundation. While .NET MAUI is still in pre-release which could have posed a problem, we saw quite a lot of positives with the framework. The .NET MAUI framework could allow for cross-platform development so that the program would run on multiple different operating systems [14]. We also wanted a framework that would be supported and be able to function for a long time. Our own experience with Microsoft products is that they tend to be supported for a long time and we trust that .NET MAUI will last for many years to come.

In the back-end section ASPNET Core would be used. We had a couple of requirements: a user-management system, the possibility to interact with the database/front-end and it would have to function across the internet. ASPNET Core fulfilled these and more. It had a built-in identity with ASPNET users [37]. We knew it most likely would function well with .NET MAUI and Microsoft SQL server, since Microsoft services tend to work well with each other. Lastly ASPNET Core is first and foremost designed for Internet-connected apps [38]. It was decided that we would use ASPNET Core as a web-api, since our system is designed around the front-end and back-end being physically separated from each other.

As a consequence of using ASPNET Core as a web-api we would have to authenticate the usage of it. While the application was in development the ASPNET Core would strictly run on localhost, but once development was over the endpoints would be vulnerable for use. As such we would utilise Json Web Tokens (JWT) to authenticate use of the API to ensure no unauthorized usage of API calls.

There was a lot of discussion in the group on which database system to use, but eventually we settled on "Microsoft SQL Server Express". We found this system to have the best qualities for our needs. Since the rest of our frameworks/systems was designed and/or owned by Microsoft we believed that keeping this consistency into the chosen system for our database would be a good idea. Additionally, it was free, it supported potentially larger databases if needed [15], and it worked similarly to MySQL which we already had some familiarity with.

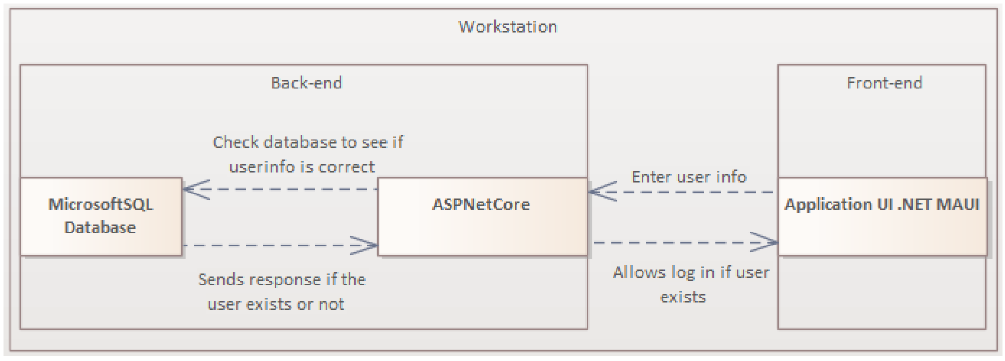

Workstation
Back-end
Front-end
Check database to see if userinfo is correct
MicrosoftSQL Database
ASPNetCore
Enter user info
Application UI .NET MAUI
Sends response if the user exists or not
Allows log in if user exists

Figure 8.2: Login example

8.2 Framework alternatives

A criticism of our framework choices could be our heavy focus on Microsoft products. .NET MAUI, ASPNET Core and Microsoft SQL Server are all Microsoft services. If changes in frameworks ever needed to be made, it could prove more difficult than previously thought. This could make us more reliant upon Microsoft than we would have optimally liked.

However we believe that these services were the right ones for us, fitting all the requirements we set out for them. We also believed all of them would work well together, since they are all from Microsoft, making the whole process more manageable. We also believe that these frameworks will last a long time before potentially being phased out, adding to the longevity of the project.

8.2.1 Front-end

There are some alternative frameworks that could have been utilized, but did not quite match our needs.

- **Windows Presentation Foundation (WPF)** - WPF does not support cross platform development and only runs on windows machines [39]. In the end it was decided that the ability for Overseer to run on multiple systems would future-proof the program more and WPF was therefore dropped. We believe that since WPF and .NET MAUI offer the same opportunities but with or without cross-platform development respectively then most likely .NET MAUI will replace WPF in the future.
- **Universal Windows Platform (UWP)** - UWP could theoretically work well for us, but since we concluded that having the ability for the program to run on all systems, not just windows platforms (PC, Xbox, Mixed-reality headset, and so on.), UWP would not cut it [40].

- **Xamarin** - Xamarin, just like UWP could theoretically work really well for our needs, but it was decided that the possibility of a user using MacOS or Linux was not zero, and so we decided to drop Xamarin in favor of its more updated and inclusive evolution "MAUI" [41].

- **Windows Forms** - Windows Forms is the oldest candidate [42], which is exactly why it was not chosen. Although Windows Forms has a lot of documentation on the internet, Microsoft has been trying to move away from Windows Forms in favour of newer frameworks. Therefore it was decided that developing the app on a framework that was "dying" would just lead to problems in the future as eventually Windows Forms would no longer be supported or usable [43].

8.2.2 Back-end

While there are many alternatives for back-end framework, we decided upon ASPNET Core being the framework of choice early on. Not only did we have experience with the framework from previous projects, we also found it had all the features needed for our specific needs. Lastly since both our front-end and database was developed and maintained by Microsoft, ASPNET Core would function well with them.

8.3 Database alternatives

Though MicrosoftSQL was more compatible with the project's needs, there were other alternatives.

- **PostgreSQL** - PostgreSQL was a contender for our database system of choice, but lost out to Microsoft-SQL due to PostgreSQL being slower than its competitors [44] [45]. Additionally the compatibility between Microsoft-SQL and the rest of our frameworks/systems made work easier than learning PostgreSQL.

- **Oracle** - Oracle was not chosen due to a bad track record from the company, as we were very unsure on the longevity of the system as the Oracle company has shown a lack of support and interest in continuing previous systems (At least the public opinion appears so) [46].

- **MySQL** - MySQL was not chosen due to limitations of the system such as not supporting large databases and being prone to data corruption [47] [45], making it unsuitable for our needs. Additionally, MySQL is currently owned by Oracle, which in it self is a negative, MySQL was bought by Oracle in 2010 [48].

- **SQLite** - SQLite was not chosen dues to its limited database size, as well as its inability to connect to servers over the internet. SQLite works best when used as a localhost database for smaller and less cross-platform heavy systems [49] [45].

Chapter 9

Theory

In this section we will present and explain the various tools, frameworks or technologies we used during this project. In explaining these technologies it will become clear why they were chosen based on previous requirements.

9.1 Enterprise Architect

Enterprise Architect is a Unified Modeling Language(UML) modeling platform used to create everything from flow-charts to Ui wireframing. It allows for easy oversight and planning and help give a clear overview over IT-architectures [6]

9.2 .NET framework

.NET is a framework initially developed and released 20 years ago by Microsoft. Since then it has continued to be one of the most widely used developer platforms. Its designed purpose is to create a large multitude of applications. Everything from programs for desktop, mobile apps, or web development. While other programming languages can be used (F#, Visual Basic), it mostly utilises C#. The newest version of the original .NET Framework is 4.8 [50] [51] [1].

9.2.1 .NET Core

.NET Core was introduced as a new open-source alternative to .NET in 2016 and for a time worked side-by-side. .NET Core would have more focus on cross-platform access and cloud-functionality, while missing some functionality like Windows Communication Foundation. The newest version of .NET Core was 3.1

These would for a time be separate from each other, until Microsoft decided to combine both of them into a new version of .NET Core called .NET 5.0, creating a common platform (.NET 4.0 was skipped to avoid confusion with .NET Framework 4.8). The newest stable version is .NET 6.0. This version would continue the unifying process, add new features and previews. Most importantly, a preview of .NET MAUI is set to become available when .NET 7.0 releases.[52] [53] [54] [55]

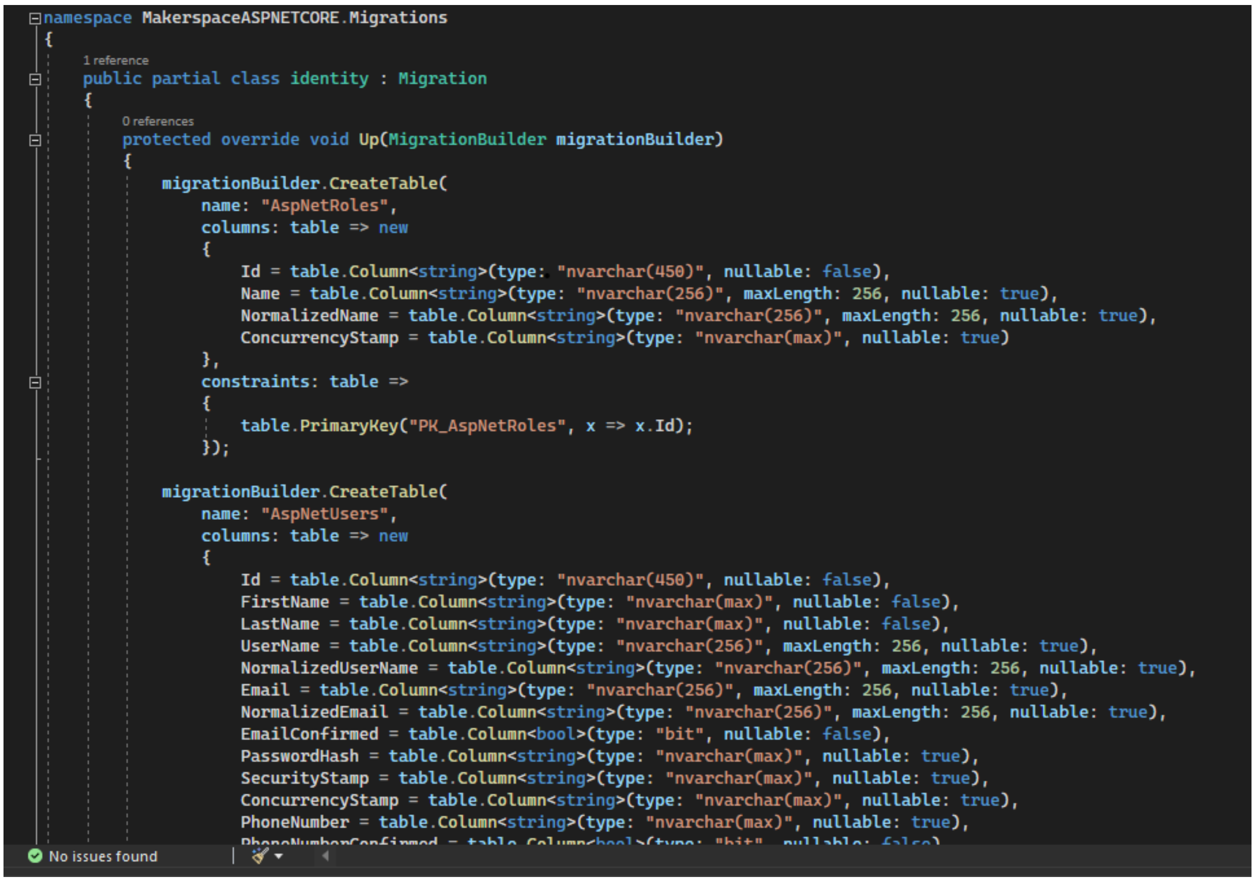
9.3 ASPNET Core

ASP NET Core is an open-source framework within the .NET Platform. This framework focuses on web-development of web applications and APIs. ASP NET Core offers: creating web pages with c# using Blazor, implementation of libraries and features such as EF and ASP.NET Core Identity, while mostly following a Model-View-Controller(MVC) format. [4]

MVC is a design choice for separating the application into three parts: The view containing the visual portions of the app, the model used for standardising data and the controller handling any requests made by the user and ultimately decided which view is rendered. [56]

9.4 Entity framework

Entity Framework (EF) is an incredibly useful framework whenever a developer wishes for a database that correctly reflects the models they have created. In an ASPNET project for instance, EF is able to analyse the models the developer has created and then automatically infer the necessary database structure. The necessary database structure is then created by a "migration" file. The migration file is the necessary SQL-code to create said database structure [7] [57] [58].



```
namespace MakerspaceASPNETCORE.Migrations
{
    1 reference
    public partial class identity : Migration
    {
        0 references
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "AspNetRoles",
                columns: table => new
                {
                    Id = table.Column<string>(type: "nvarchar(450)", nullable: false),
                    Name = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    NormalizedName = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetRoles", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "AspNetUsers",
                columns: table => new
                {
                    Id = table.Column<string>(type: "nvarchar(450)", nullable: false),
                    FirstName = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    LastName = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    UserName = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    NormalizedUserName = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    Email = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    NormalizedEmail = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    EmailConfirmed = table.Column<bool>(type: "bit", nullable: false),
                    PasswordHash = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    SecurityStamp = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    PhoneNumber = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    PhoneNumberConfirmed = table.Column<bool>(type: "bit", nullable: false)
```

Figure 9.1: Migration file of our ASPNET users

9.5 .NET MAUI

.NET Multi-platform App UI (.NET MAUI) is a cross-platform framework for creating native mobile and desktop apps with C# and XAML [14]. The most unique and standout feature of MAUI is the fact that cross-platform programming very simple. Using .NET MAUI, you can develop apps that can run on Android, iOS, macOS, and Windows from a single shared code-base. .NET MAUI is an evolution of Xamarin.forms, which worked similarly, but were only compatible with IOS, Android and Windows. As of right now .NET MAUI is in pre-release and will be fully release along with .NET 7.

.NET Maui works by unifying Android, IOS, MacOS and windows APIs into a single API that "runs anywhere". Allowing the developer to design and create without the worry on what system the user might employ.

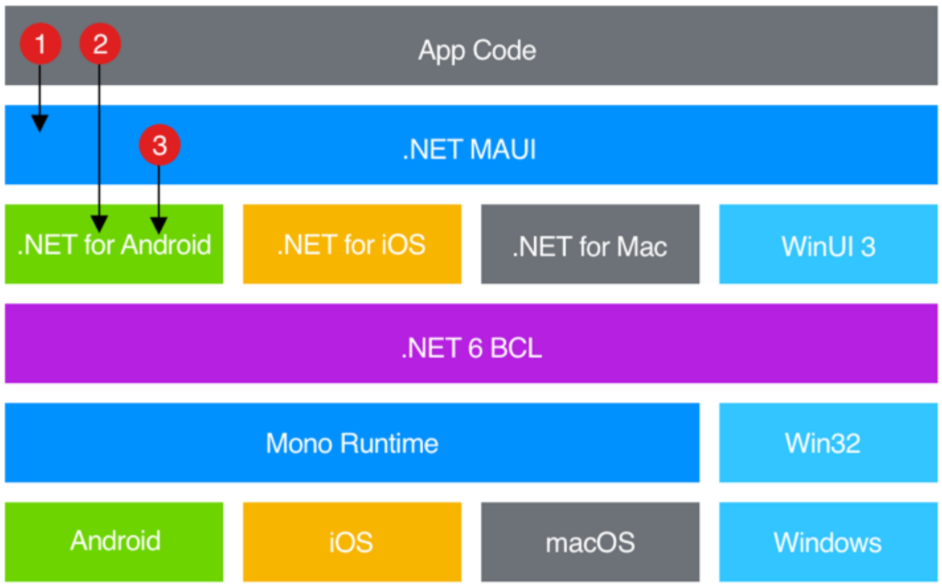


Figure 9.2: Illustration of a high-level view of the .NET MAUI app.
Image taken from [14].

"In a .NET MAUI app, you write code that primarily interacts with the .NET MAUI API (1). .NET MAUI then directly consumes the native platform APIs (3). In addition, app code may directly exercise platform APIs (2), if required." [14]

9.6 Swagger

Swagger adds the potential for added functionality to an existing API. If Swagger is added to a API, Swagger will return a specification of the entire API in a YAML/JSON format. Many other tools rely on this specification [59], such as:

- **Swagger UI** - Used in API documentation and testing. Opens a browser with all current API Controllers functions. This site is interactive, letting you change values to see the response from the API. [60]

- **Swagger Codegen** - "Swagger Codegen can simplify your build process by generating server stubs and client SDKs for any API" [61]

9.7 NSwag

NSwag is described as a "Swagger/OpenAPI toolchain for .NET, ASP.NET Core and TypeScript" [62] and is very useful when it comes to API-consumption. NSwag is capable of creating C# clients based of your API-controllers with just a few configurations. If the specific project is set to use Swagger for instance, NSwag is able to utilise the JSON schema from Swagger to create a C# client for API-consumption [63] [62].

9.8 MicrosoftSQL

MicrosoftSQL is a database service provided by Microsoft that is free, and easy to use. To manage the database we used SSMS. "SQL Server Management Studio (SSMS) is an integrated environment for managing any SQL infrastructure" [15] and allows users to query, design and manage databases [64].

9.9 Jira

Jira is an management system for teams or developers working in an agile method. Jira digitises the whole process, instead of agreeing on issues and sprints through post-it notes or writing. Most features a developer would want are present in Jira, but also extra features such as time-tracking, reports, Gantt schemata or even integration with GitHub or Bitbucket [65].

9.10 Raspberry Pi

A Raspberry Pi is very useful for when you want most of the functionality of a computer, but not necessarily the size. A Raspberry Pi is essentially an extremely tiny and highly customizable computer [66]. It comes with Linux, and provides GPIOs for additional customizability, however many people use it simply as a small and discrete controller [67].

9.11 Json Web Tokens

Developers require a method for authenticating who is allowed to utilise their API. One such method is using Json Web Tokens (JWT) for authentication. [68] [69]

A JWT is split into 3 sections:

- Header - Contains information about the token itself. This metadata would identity that it is a JWT token and what signing algorithm it uses, which becomes important in the signature. [70][71] [72]
- Payload - Contains claims. Claims are essentially information pertaining to an object. These claims could for instance be a users email and password.[70][71] [72]
- Signature - Used to validate the token. Using the signing algorithm previously mentioned, the combination of header and payload encoded is signed using a secret string. A secret string which is only known to the sender and receiver.[70][71] [72]

JWT tokens can be sent between web-server and client to communicate what a client is then authorized to do using claims.

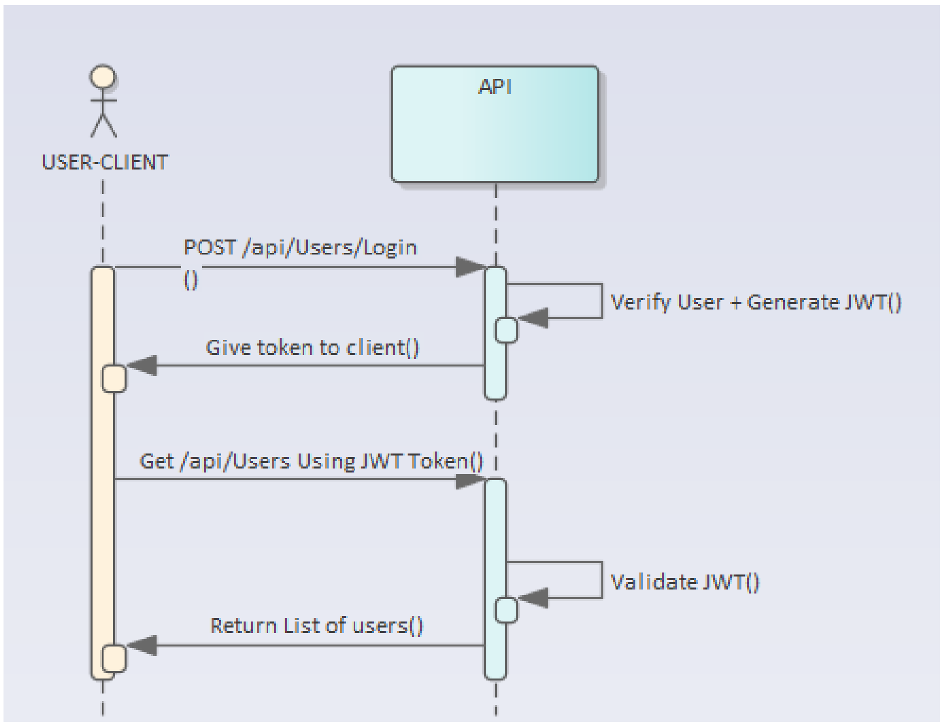


Figure 9.3: JWT authentication to log-in and get a list of all users in the system. Used as reference for diagram: [73] & [74]

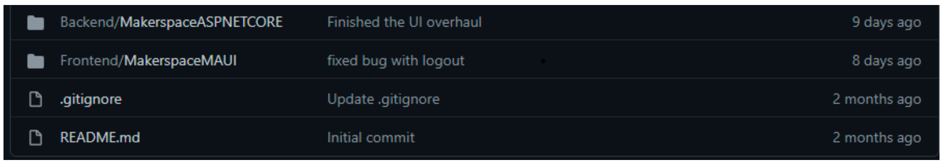
Chapter 10

Solution

This chapter will go through the result of our project and will serve as a general overview of the solution. This Chapter will show of different parts of the software, including code-snippets, front-end UI and parts of the API

10.1 Git & GitHub

For version control of the project, Git [9] and GitHub [10] is utilized. We started by creating a private repository called "MakerspaceRepo" and started adding in the skeletal structure. The skeletal structure was composed of two files; "Frontend" and "Backend".



Backend/MakerspaceASPNETCORE	Finished the UI overhaul	9 days ago
Frontend/MakerspaceMAUI	fixed bug with logout	8 days ago
.gitignore	Update .gitignore	2 months ago
README.md	Initial commit	2 months ago

Figure 10.1: GitHub file structure

This was done to clearly distinguish between the two parts of the project. As a consequence, managing what parts of the project was visual and which parts was functional was be a lot easier.

10.2 Current situation

At this time the project is still very much in phase one, as we have not implemented any interaction with the printers. So far the program contains:

- A general overview of printer status, based of information in the back-end.
- A user management system, with login/register.
- A printer-usage page, for whenever using a printer.
- An empty settings page.
- A page for users to submit issues to admins.
- An admin screen for adding/updating/deleting printers. Admins are also able to view printer history based on the activity in the printer usage page. Lastly they are able to review issues submitted to them by their users.

The first section will show how the program looks thus far in our .NET MAUI. We will take a look at the functionality of the application and lastly we will do a deep-dive on how our register page is setup to give a better idea of how the application functions.

10.3 Login and register page

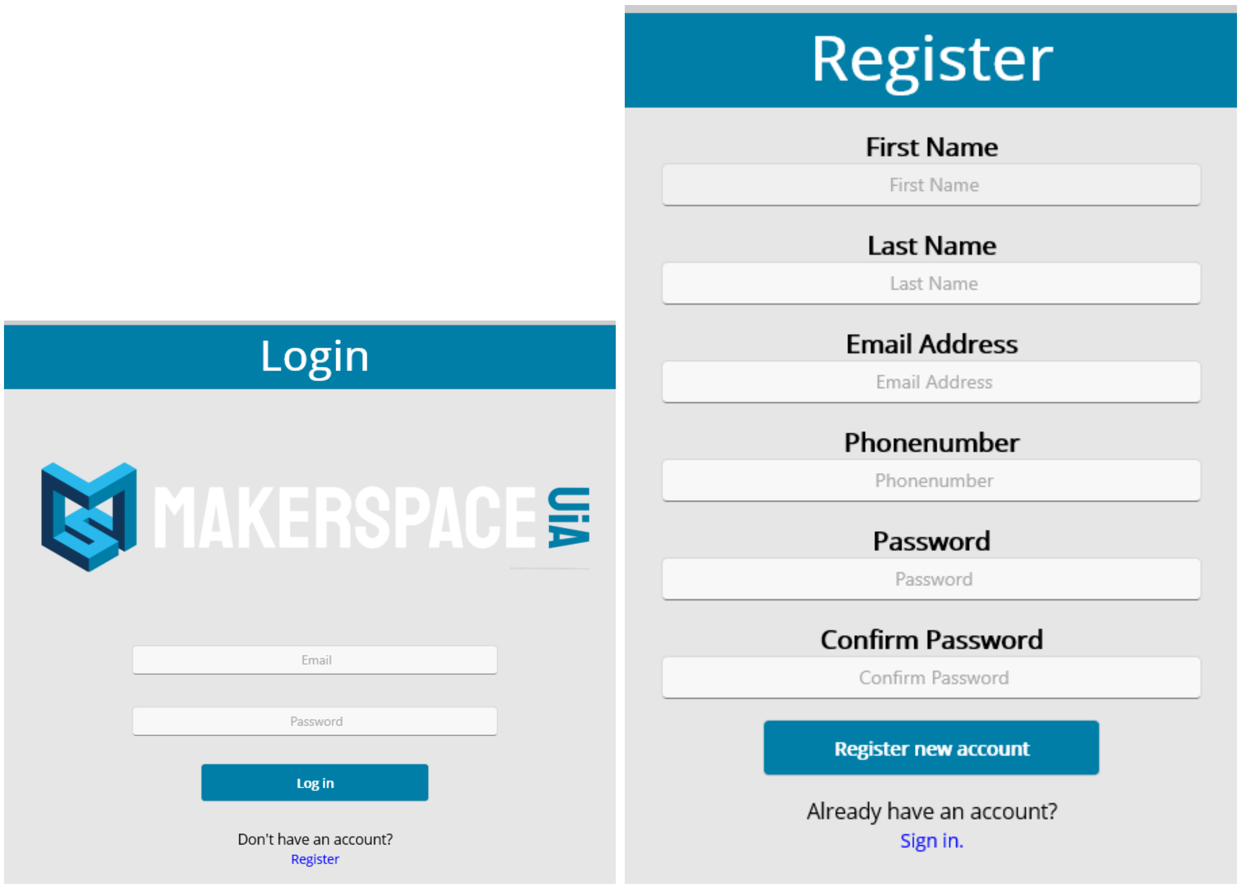
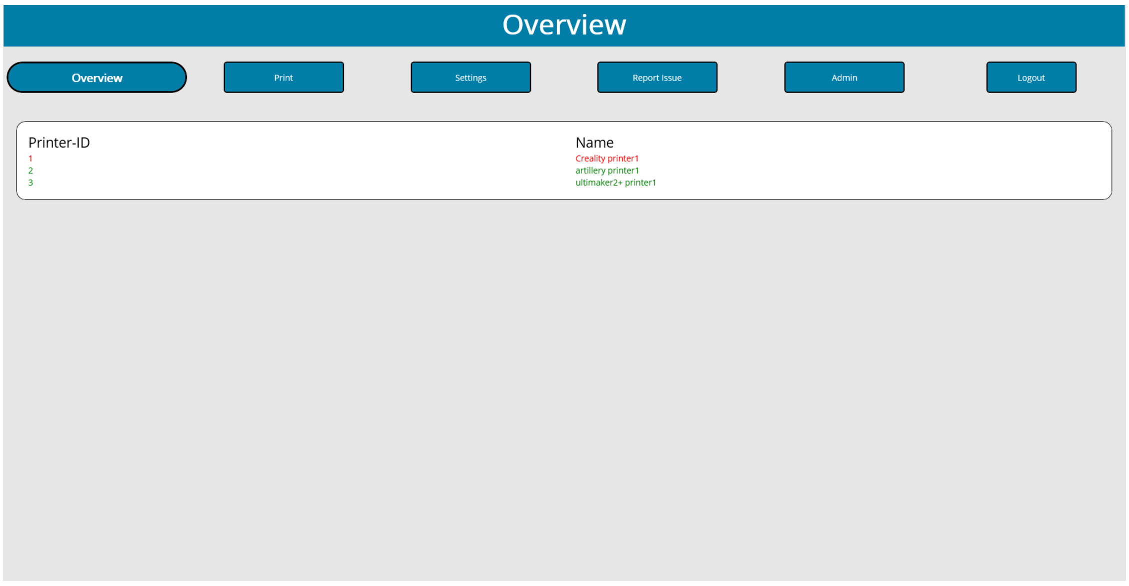


Figure 10.2: Log-in and register page

The register page allows users to insert their information to create a new account in our systems, as long as all the information they put in is correct (email address follows correct format, first/last name is at least two letters etc.).
The login page allows users to insert their email and password. If the email and corresponding password is correct, it will allow the user to go to the Overview Page.

10.4 Overview page

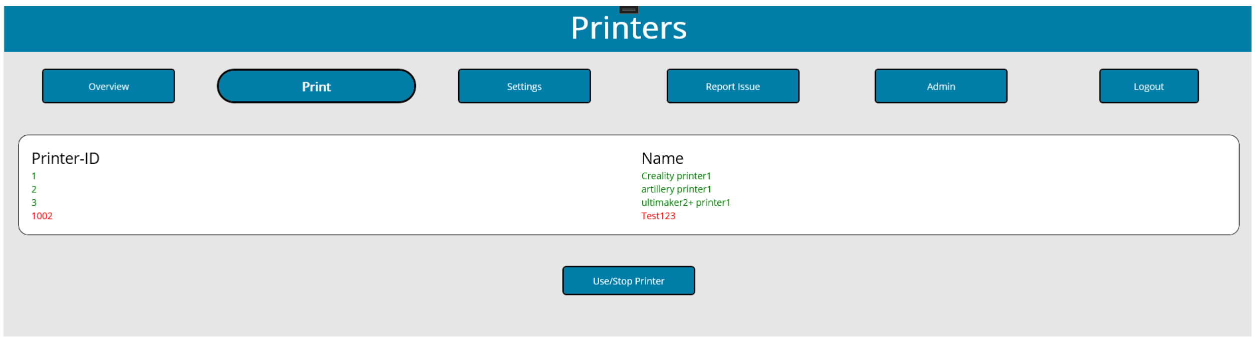


Printer-ID	Name
1	Creality printer1
2	artillery printer1
3	ultimaker2+ printer1

Figure 10.3: Printer overview

The overview page is generally considered the main page of the application. The overview screen consists of a list of printers with their ID and names present. It also allows for seeing which printers are available based upon whether they are green (available) or red (unavailable).

10.5 Printer usage page



Printer-ID	Name
1	Creality printer1
2	artillery printer1
3	ultimaker2+ printer1
1002	Test123

Figure 10.4: Select printer for use

The print screen allows all the functionality of the overview screen, but also allows users to select printers for use. If they do so, the application will update the availability in the back-end and there after update the view. Turning either the printer red or green depending on it's previous status.

10.6 Settings page

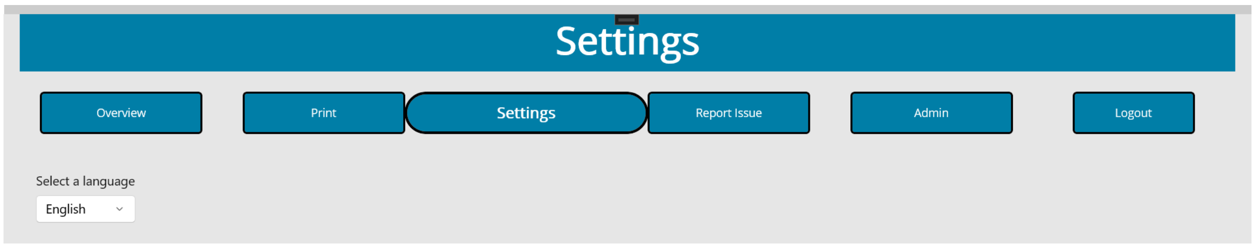


Figure 10.5: Settings

Settings currently is populated by a picker for languages. This functionality is currently not implemented properly and thus does not work.

10.7 Report issue page

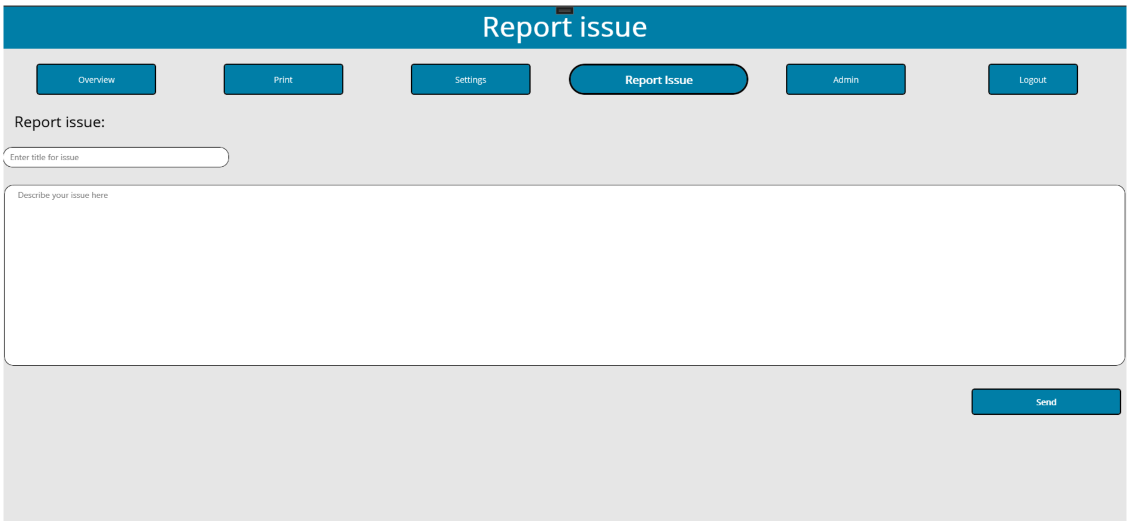
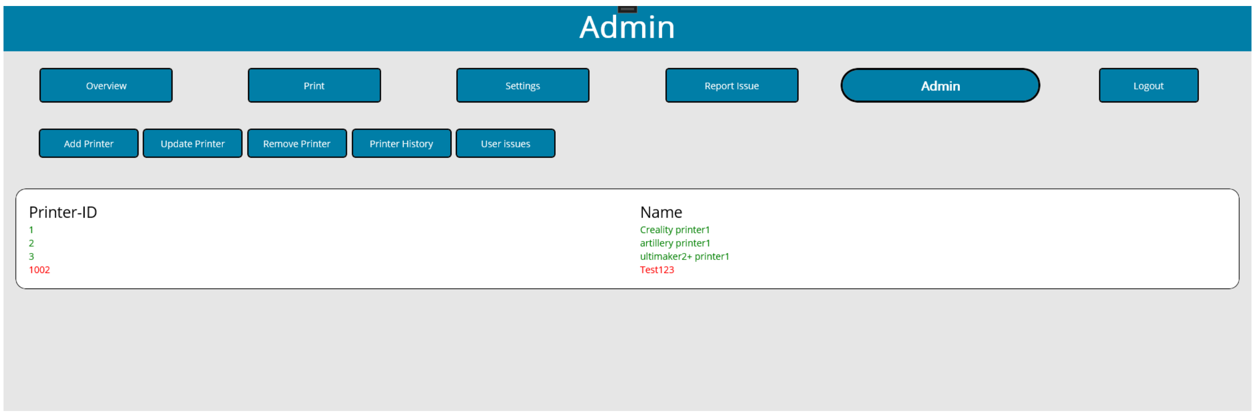


Figure 10.6: Report issue

The report view contains two entries the user can write text in. The first one being the title of the issue and the second being the actual body of the issue. If both the title and body is not empty when the user presses send, the program will create a new issue with the values given by the user. This issue will now appear in the "user issues" tab in the admin and database view.

10.8 Admin page



Printer-ID	Name
1	Creality printer1
2	artillery printer1
3	ultimaker2+ printer1
1002	Test123

Figure 10.7: Admin Issue

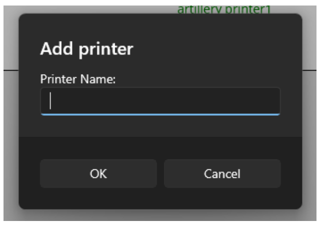
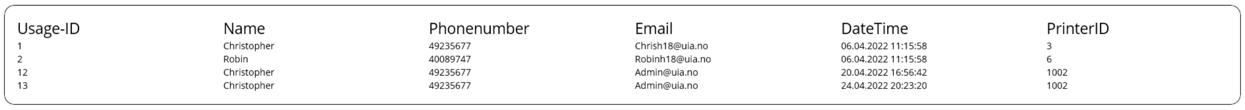


Figure 10.8: Adding new printer. updating/deleting looks very much the same

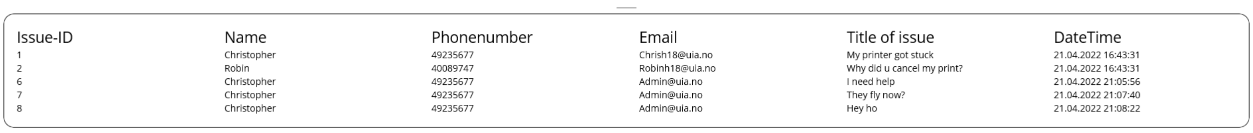
The admin view holds the most functionality of the program. The admin view contains the printer-list in the same format as Overview and Print, but admins are able to add/update/delete printers as they see fit. After doing so, the printer-list will update accordingly. The admin view also contains the option to view the printer history and user issues pages.

10.9 Printer history and user issue pages



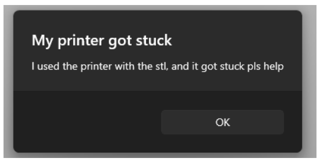
Usage-ID	Name	Phonenumber	Email	DateTime	PrinterID
1	Christopher	49235677	Chrish18@uia.no	06.04.2022 11:15:58	3
2	Robin	40089747	Robinh18@uia.no	06.04.2022 11:15:58	6
12	Christopher	49235677	Admin@uia.no	20.04.2022 16:56:42	1002
13	Christopher	49235677	Admin@uia.no	24.04.2022 20:23:20	1002

Figure 10.9: Printer history page



Issue-ID	Name	Phonenumber	Email	Title of issue	DateTime
1	Christopher	49235677	Chrish18@uia.no	My printer got stuck	21.04.2022 16:43:31
2	Robin	40089747	Robinh18@uia.no	Why did u cancel my print?	21.04.2022 16:43:31
6	Christopher	49235677	Admin@uia.no	I need help	21.04.2022 21:05:56
7	Christopher	49235677	Admin@uia.no	They fly now?	21.04.2022 21:07:40
8	Christopher	49235677	Admin@uia.no	Hey ho	21.04.2022 21:08:22

Figure 10.10: User issues page



My printer got stuck

I used the printer with the stl, and it got stuck pls help

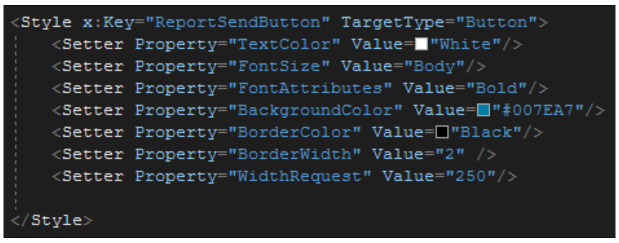
Figure 10.11: Prompt when selecting specific issue.

Printer history contains information with every record of printer-usage. It shows who used the printer, which printer and when. Printer issue contains every issue submitted by the users. Every entry contains the name of who submitted the issue, title of the issue and when it was made. If you double click an issue, a prompt will appear with the issue's title and body.

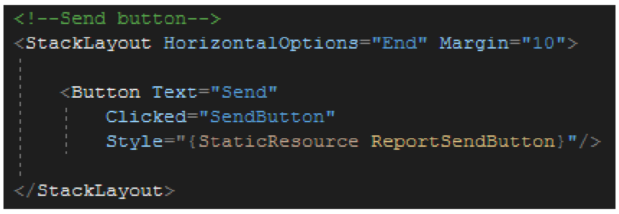
10.10 General overview

The UI design was to be created using XAML, and was to closely match the example UI created in Enterprise architect. Users can easily understand and easily navigate the program. All features can be found within three clicks maximum. XAML was the most optimal way for us to create the UI, and in addition to "styles" is the method chosen to build the UI.

In XAML you are able to use styles to easily standardize the output of your UI, creating a set "style" that might be referenced later on [75]. An example of a style could be a set flair for buttons which then is called when creating the button in question:



```
<Style x:Key="ReportSendButton" TargetType="Button">
    <Setter Property="TextColor" Value=⬜"White"/>
    <Setter Property="FontSize" Value="Body"/>
    <Setter Property="FontAttributes" Value="Bold"/>
    <Setter Property="BackgroundColor" Value=🟦"#007EA7"/>
    <Setter Property="BorderColor" Value=⬛"Black"/>
    <Setter Property="BorderWidth" Value="2" />
    <Setter Property="WidthRequest" Value="250"/>

</Style>
```



```
<!--Send button-->
<StackLayout HorizontalOptions="End" Margin="10">

    <Button Text="Send"
        Clicked="SendButton"
        Style="{StaticResource ReportSendButton}"/>

</StackLayout>
```

Figure 10.12: Example of an XAML style, style code

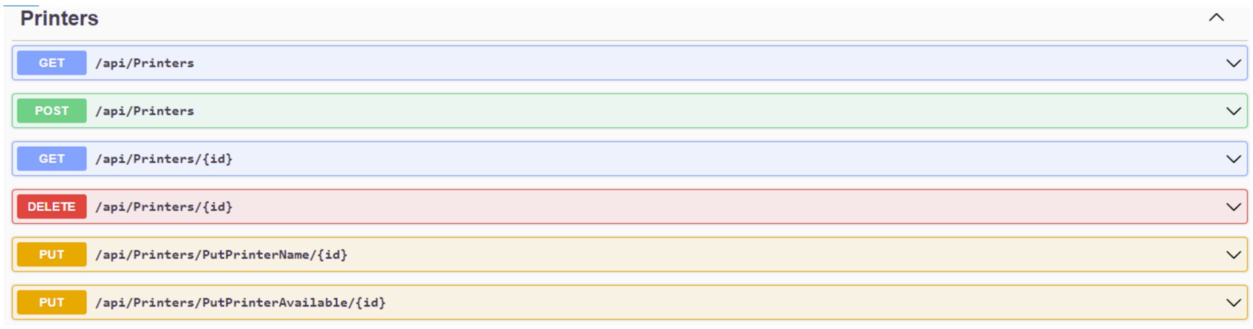
Now with the style created, we may call the style on any button desired, saving work having to re-create the same "look" for all our buttons. Unlike what was presented in the "Requirements" section of this rapport (6.2) the final UI turned out more rounded, larger, and with less features than planned. This design, which uses "signature Makerspace's" colors, easily portrays all options in a clear and concise way. The large amount of empty space is there in case a large amount of printers is added to the system, during testing we only had 4 test-printers and thus the page looks quite empty. Feature wise we were able to add most features, but the program is currently missing one important feature, that being log-in recognition. As is, there is a separate tab specifically for admin features, which we wanted to be automatic based on what account was logged in. Regretfully this feature was far more difficult the we anticipated and thus we did not distribute enough time for it. Additionally, several smaller features is currently missing from our initial draft such as a functional settings screen, estimated time remaining and multiple languages.

Despite the smaller changes we are overall happy with the UI, and if the missing log-in recognition were taken care of we would be very happy with how the UI turned out. As of right now the user is able to change a printer from available to unavailable and back, students are able to register a user which will be recognized in the database, and the user will have to enter the correct information to be able to log in in the first place.

10.11 API

Currently the API functionality is quite large with four controllers with their corresponding model (Printers, PrinterUsage, Users and UserIssues), but does not invlove JWT authentication like planned. It contains too many functions to show of each one, instead we will examine 'PostUser' closer in 9.2.

So far we have these in our printers controller so far:



Printers

GET /api/Printers

POST /api/Printers

GET /api/Printers/{id}

DELETE /api/Printers/{id}

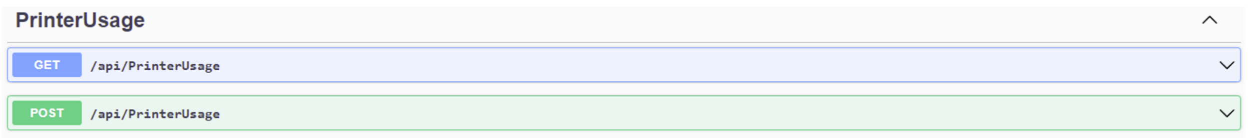
PUT /api/Printers/PutPrinterName/{id}

PUT /api/Printers/PutPrinterAvailable/{id}

Figure 10.13: Documentation of all current API calls for Printers.

- GET /api/printers - Used to populate our printer overviews with lists of printers and their current status
- POST /api/printers - Used in the admin screen for the "Add printer" button. Adds printer based on printer name given.
- Get /api/printers/id - Not currently in use.
- DELETE /api/printers/id - Adds functionality to the "Delete Printer" button in admin screen. Deletes printer with the corresponding ID.
- PUT /api/Printers/PutPrinterName/id - Adds functionality to the "Update Printer" button in admin screen. Updates name to the printer with the ID given.
- PUT /api/Printers/PutPrinterAvailable/id - Adds functionality to the "Use/Stop Printer" button in print view. Updates the "Available" bool to the printer selected in the list.

We currently only have two functions in PrinterUsage, but plan for more. Specifically a DELETE function to delete entries 7 days after they were created.


PrinterUsage

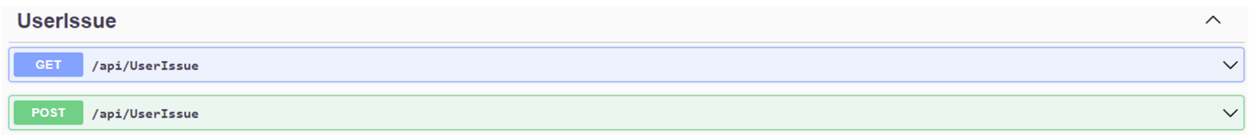
GET /api/PrinterUsage

POST /api/PrinterUsage

Figure 10.14: Documentation of all current API calls for PrinterUsage.

- GET /api/PrinterUsage - Used to populate the printer history in admin view with lists of printer usages.
- POST /api/PrinterUsage - This call happens every time a user uses the "Use/Stop Printer" button. Adds a new entry in PrinterUsage with the printer used, who used it and the current time.

The userIssue controller also contains two functions. Similarly to printer usage, we plan to add a DELETE function to remove entries 7 days after they were created.


UserIssue

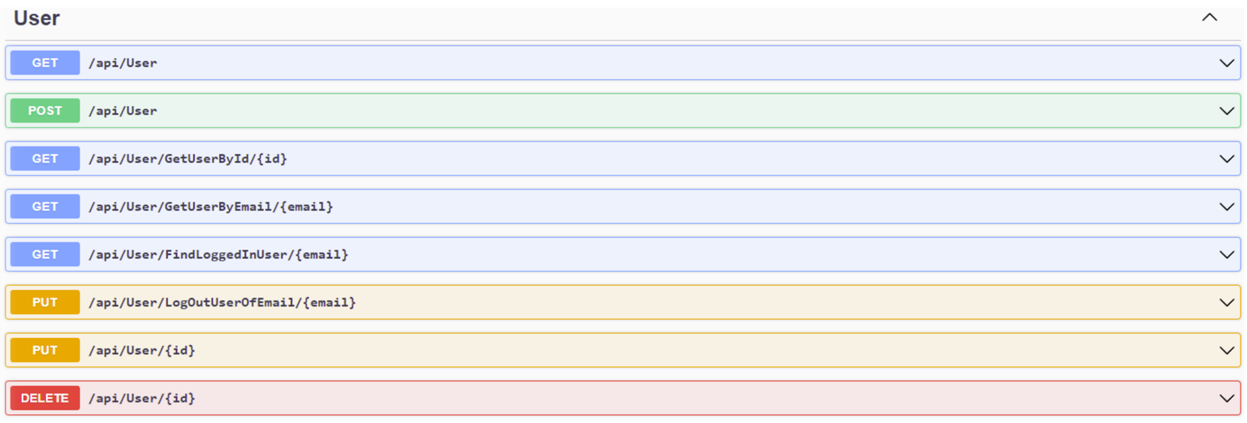
GET /api/UserIssue

POST /api/UserIssue

Figure 10.15: Documentation of all current API calls for UserIssues.

- GET /api/UserIssue - Used to populate our user issues in admin view with lists of user issues.
- POST /api/UserIssue - Adds functionality in the "Report Issue" screen. Adds a new entry of user issue containing the user information, issue-title, issue-body and the time when sent.

The User controller is definitely one of the largest controllers so far, adding a lot of functionality to the application


User

GET /api/User

POST /api/User

GET /api/User/GetUserById/{id}

GET /api/User/GetUserByEmail/{email}

GET /api/User/FindLoggedInUser/{email}

PUT /api/User/LogOutUserOfEmail/{email}

PUT /api/User/{id}

DELETE /api/User/{id}

Figure 10.16: Documentation of all current API calls for Users.

- GET /api/User - Not currently in use in our application.
- POST /api/User - Adds functionality to our register page. Takes the account information given by the user to create a new one.
- Get /api/User/GetUserById/id - Not finished.
- GET /api/User/GetUserByEmail/email - Used in Login screen. Takes in email and password. Checks if account with email exists, then check if password matches accounts password. Returns boolean.
- GET /api/User/FindLoggedInUser - Not finished.
- PUT /api/User/LogOutUserOfEmail - Not finished.
- PUT /api/User/id - Not currently in use. planned for settings screen to update account settings.
- DELETE /api/User/id - Not currently in use. Planned for the deletion button in settings screen, pending further development.

10.12 Register page insight

To get a sense of how the service operates, we will take a closer look at how registering functions. When the program starts, users are presented with the login screen. They can choose to either login with existing accounts or create new accounts

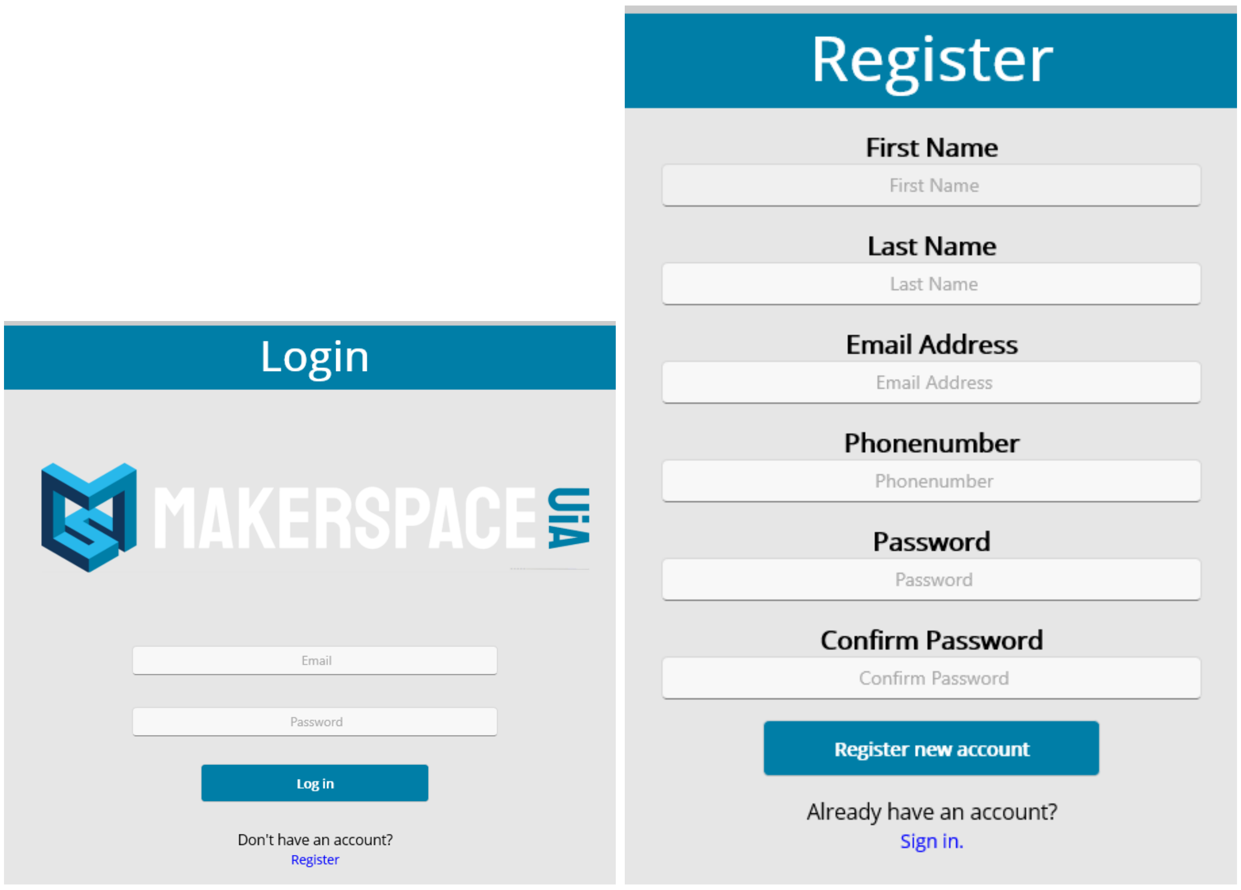
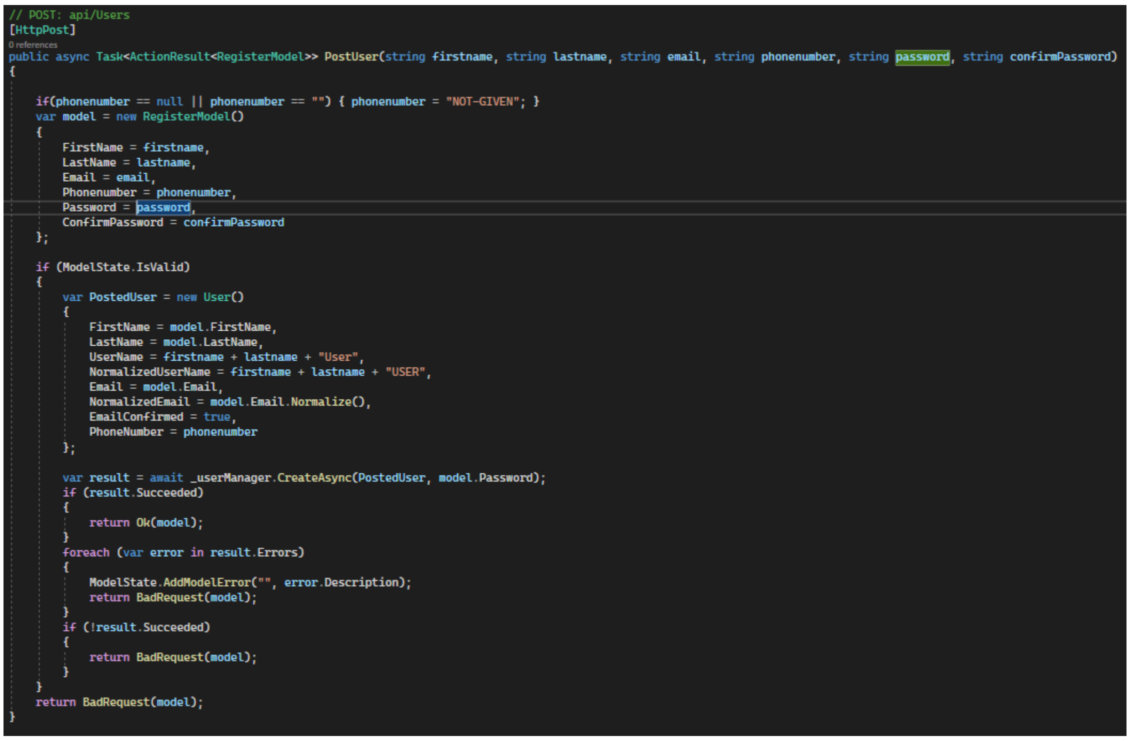


Figure 10.17: Log-in and register page

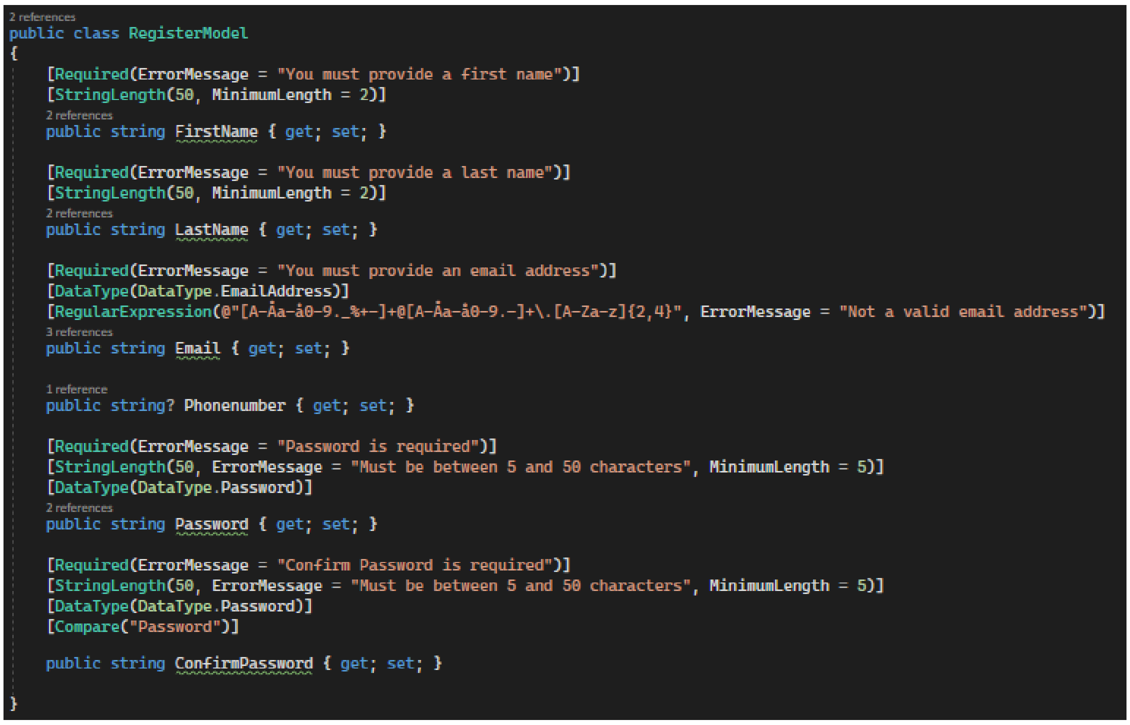
The Register page makes use of our back-end API, specifically our UserController. In User-Controller we have the POST function 'PostUser'. Which takes in the necessary information, confirms that the format is valid and creates an ASPNET user with the information given.



```csharp
// POST: api/Users
[HttpPost]
0 references
public async Task<ActionResult<RegisterModel>> PostUser(string firstname, string lastname, string email, string phonenumber, string password, string confirmPassword)
{

    if(phonenumber == null || phonenumber == "") { phonenumber = "NOT-GIVEN"; }
    var model = new RegisterModel()
    {
        FirstName = firstname,
        LastName = lastname,
        Email = email,
        Phonenumber = phonenumber,
        Password = password,
        ConfirmPassword = confirmPassword
    };

    if (ModelState.IsValid)
    {
        var PostedUser = new User()
        {
            FirstName = model.FirstName,
            LastName = model.LastName,
            UserName = firstname + lastname + "User",
            NormalizedUserName = firstname + lastname + "USER",
            Email = model.Email,
            NormalizedEmail = model.Email.Normalize(),
            EmailConfirmed = true,
            PhoneNumber = phonenumber
        };

        var result = await _userManager.CreateAsync(PostedUser, model.Password);
        if (result.Succeeded)
        {
            return Ok(model);
        }
        foreach (var error in result.Errors)
        {
            ModelState.AddModelError("", error.Description);
            return BadRequest(model);
        }
        if (!result.Succeeded)
        {
            return BadRequest(model);
        }
    }
    return BadRequest(model);
}
```

Figure 10.18: UserController PostUser function

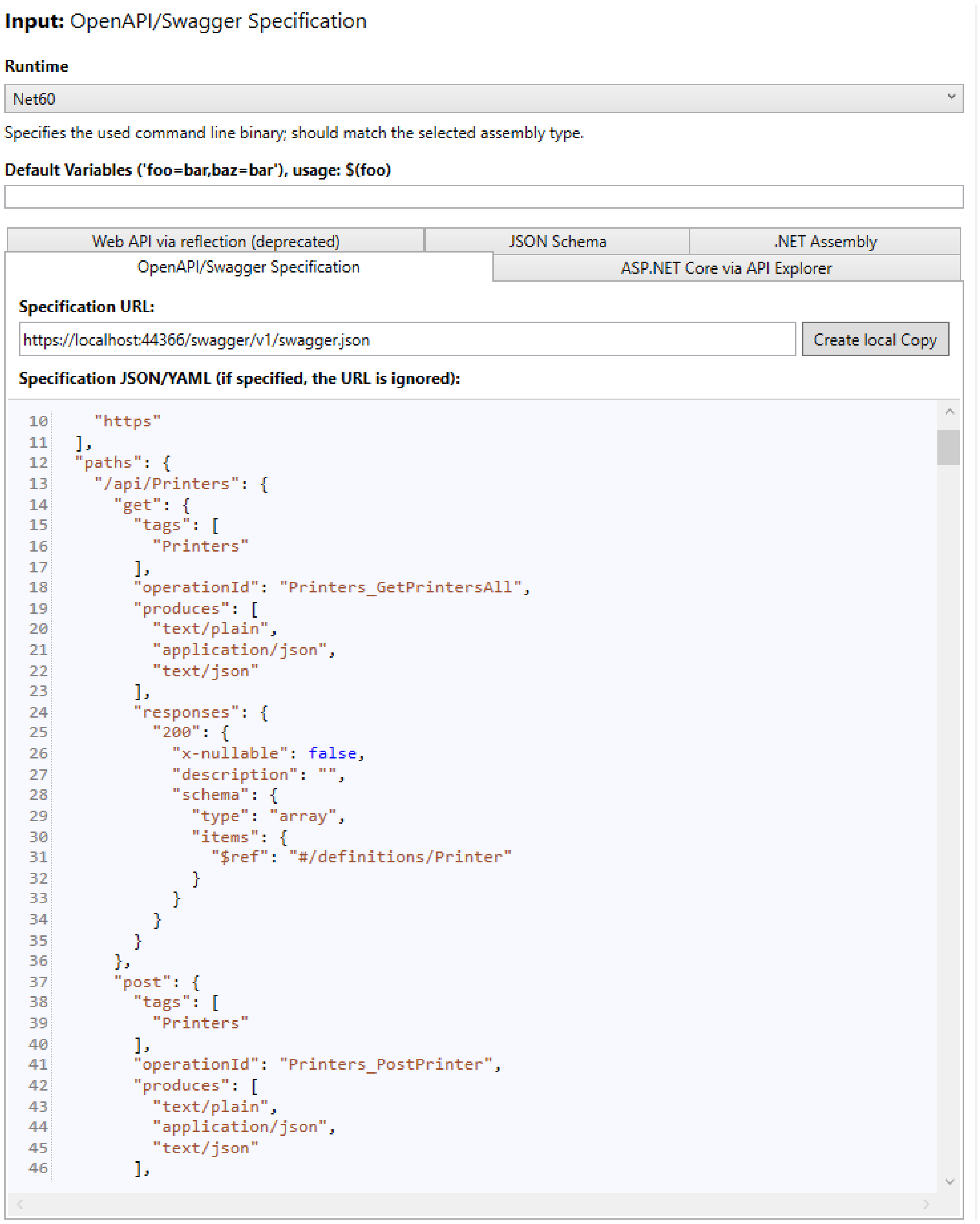
To avoid using too many if-statements checking whether or not the parameters are correct, we instead decided to use a Register Model with the requirements necessary. These requirements could be a [Require], a min/max length or text structure for Email. We could then use the built-in functionality of "ModelState.IsValid" to validate.



```csharp
2 references
public class RegisterModel
{
    [Required(ErrorMessage = "You must provide a first name")]
    [StringLength(50, MinimumLength = 2)]
    2 references
    public string FirstName { get; set; }

    [Required(ErrorMessage = "You must provide a last name")]
    [StringLength(50, MinimumLength = 2)]
    2 references
    public string LastName { get; set; }

    [Required(ErrorMessage = "You must provide an email address")]
    [DataType(DataType.EmailAddress)]
    [RegularExpression(@"[A-Åa-å0-9._%+-]+@[A-Åa-å0-9.-]+\.[A-Za-z]{2,4}", ErrorMessage = "Not a valid email address")]
    3 references
    public string Email { get; set; }

    1 reference
    public string? Phonenumber { get; set; }

    [Required(ErrorMessage = "Password is required")]
    [StringLength(50, ErrorMessage = "Must be between 5 and 50 characters", MinimumLength = 5)]
    [DataType(DataType.Password)]
    2 references
    public string Password { get; set; }

    [Required(ErrorMessage = "Confirm Password is required")]
    [StringLength(50, ErrorMessage = "Must be between 5 and 50 characters", MinimumLength = 5)]
    [DataType(DataType.Password)]
    [Compare("Password")]

    public string ConfirmPassword { get; set; }

}
```

Figure 10.19: RegisterModel class

We used the very useful tool NSWAG for API-consumption in our front-end. NSWAG would examine the JSON Schema of our SWAGGER documentation and create a perfect c# client for consumption based of our specification. When ever we changed or added functionality to the controllers we would re-do this action, and update the c# client used in the front-end.


Input: OpenAPI/Swagger Specification

Runtime

Net60

Specifies the used command line binary; should match the selected assembly type.

Default Variables ('foo=bar,baz=bar'), usage: $(foo)

Web API via reflection (deprecated)
JSON Schema
.NET Assembly
OpenAPI/Swagger Specification
ASP.NET Core via API Explorer

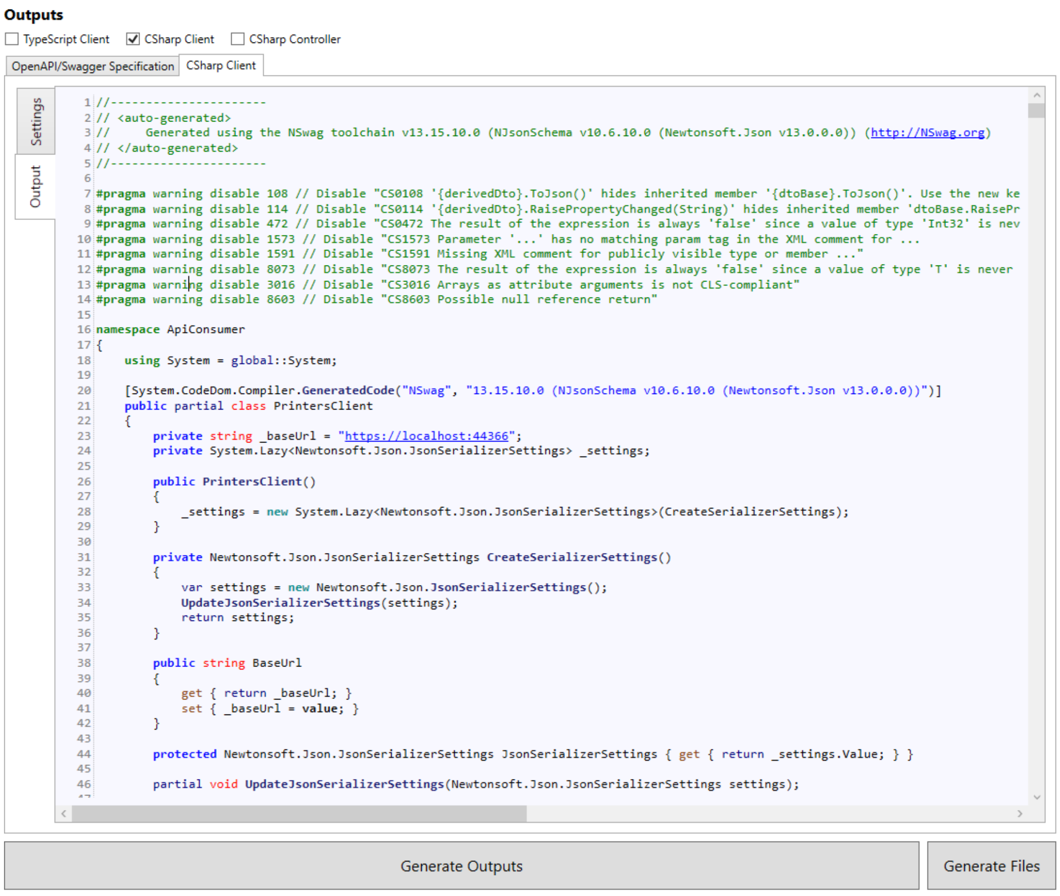
Specification URL:

https://localhost:44366/swagger/v1/swagger.json

Specification JSON/YAML (if specified, the URL is ignored):

```
    "https"
  ],
  "paths": {
    "/api/Printers": {
      "get": {
        "tags": [
          "Printers"
        ],
        "operationId": "Printers_GetPrintersAll",
        "produces": [
          "text/plain",
          "application/json",
          "text/json"
        ],
        "responses": {
          "200": {
            "x-nullable": false,
            "description": "",
            "schema": {
              "type": "array",
              "items": {
                "$ref": "#/definitions/Printer"
              }
            }
          }
        }
      },
      "post": {
        "tags": [
          "Printers"
        ],
        "operationId": "Printers_PostPrinter",
        "produces": [
          "text/plain",
          "application/json",
          "text/json"
        ],
```

Figure 10.20: NSWAG tool input



Outputs

TypeScript Client, [✓ CSharp Client], CSharp Controller

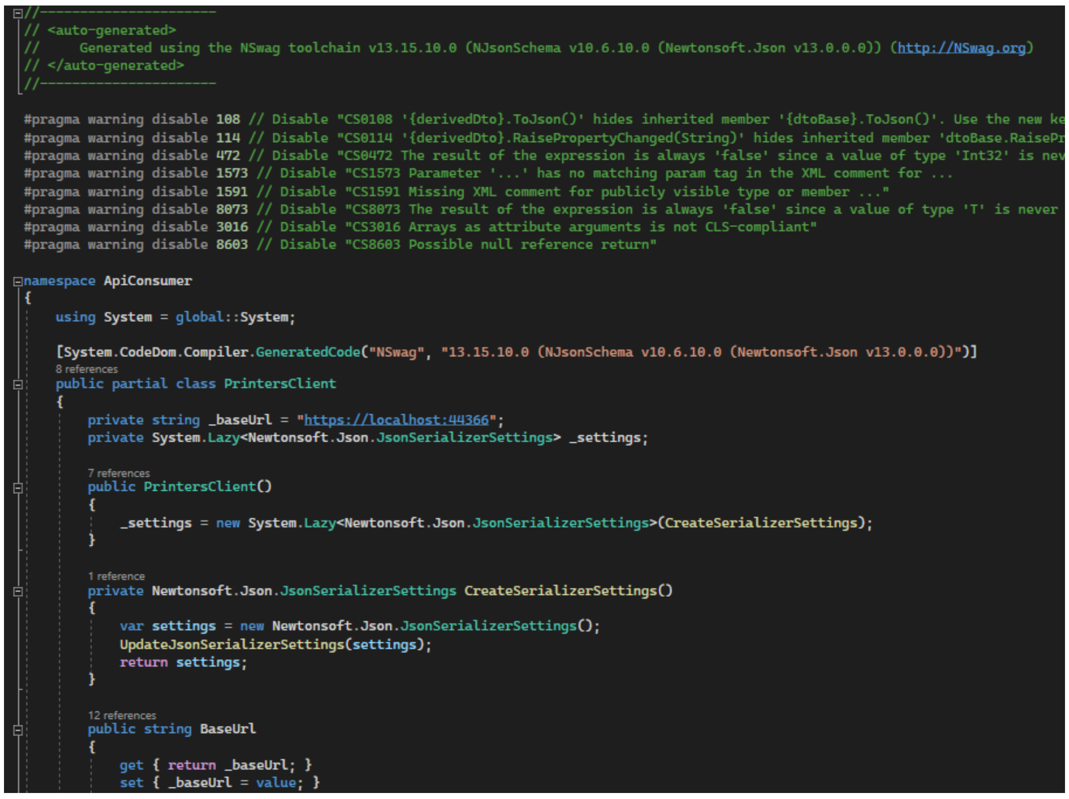
OpenAPI/Swagger Specification | CSharp Client

Settings | Output

```
//----------------------
// <auto-generated>
//     Generated using the NSwag toolchain v13.15.10.0 (NJsonSchema v10.6.10.0 (Newtonsoft.Json v13.0.0.0)) (http://NSwag.org)
// </auto-generated>
//----------------------

#pragma warning disable 108 // Disable "CS0108 '{derivedDto}.ToJson()' hides inherited member '{dtoBase}.ToJson()'. Use the new ke
#pragma warning disable 114 // Disable "CS0114 '{derivedDto}.RaisePropertyChanged(String)' hides inherited member 'dtoBase.RaisePr
#pragma warning disable 472 // Disable "CS0472 The result of the expression is always 'false' since a value of type 'Int32' is nev
#pragma warning disable 1573 // Disable "CS1573 Parameter '...' has no matching param tag in the XML comment for ...
#pragma warning disable 1591 // Disable "CS1591 Missing XML comment for publicly visible type or member ..."
#pragma warning disable 8073 // Disable "CS8073 The result of the expression is always 'false' since a value of type 'T' is never
#pragma warning disable 3016 // Disable "CS3016 Arrays as attribute arguments is not CLS-compliant"
#pragma warning disable 8603 // Disable "CS8603 Possible null reference return"

namespace ApiConsumer
{
    using System = global::System;

    [System.CodeDom.Compiler.GeneratedCode("NSwag", "13.15.10.0 (NJsonSchema v10.6.10.0 (Newtonsoft.Json v13.0.0.0))")]
    public partial class PrintersClient
    {
        private string _baseUrl = "https://localhost:44366";
        private System.Lazy<Newtonsoft.Json.JsonSerializerSettings> _settings;

        public PrintersClient()
        {
            _settings = new System.Lazy<Newtonsoft.Json.JsonSerializerSettings>(CreateSerializerSettings);
        }

        private Newtonsoft.Json.JsonSerializerSettings CreateSerializerSettings()
        {
            var settings = new Newtonsoft.Json.JsonSerializerSettings();
            UpdateJsonSerializerSettings(settings);
            return settings;
        }

        public string BaseUrl
        {
            get { return _baseUrl; }
            set { _baseUrl = value; }
        }

        protected Newtonsoft.Json.JsonSerializerSettings JsonSerializerSettings { get { return _settings.Value; } }

        partial void UpdateJsonSerializerSettings(Newtonsoft.Json.JsonSerializerSettings settings);
```

Generate Outputs | Generate Files

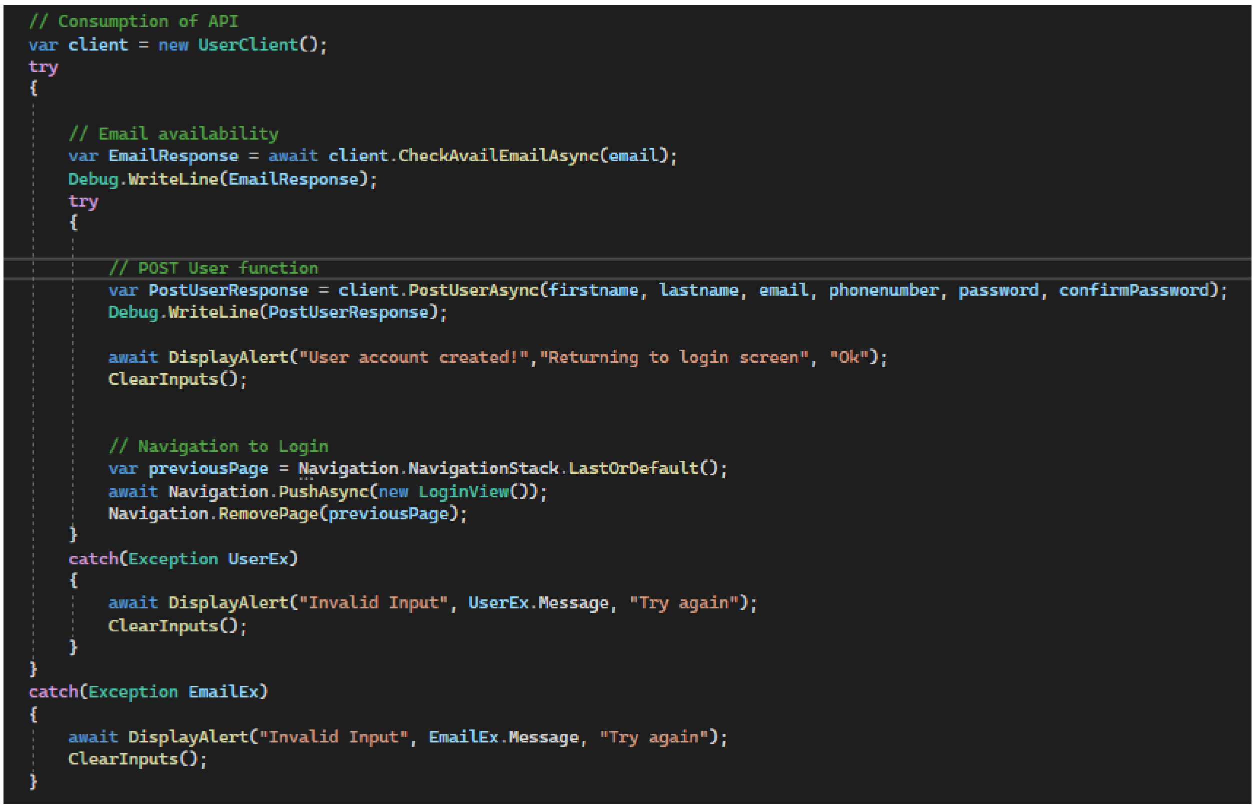
Figure 10.21: NSWAG tool output



```
//----------------------
// <auto-generated>
//     Generated using the NSwag toolchain v13.15.10.0 (NJsonSchema v10.6.10.0 (Newtonsoft.Json v13.0.0.0)) (http://NSwag.org)
// </auto-generated>
//----------------------

#pragma warning disable 108 // Disable "CS0108 '{derivedDto}.ToJson()' hides inherited member '{dtoBase}.ToJson()'. Use the new ke
#pragma warning disable 114 // Disable "CS0114 '{derivedDto}.RaisePropertyChanged(String)' hides inherited member 'dtoBase.RaisePr
#pragma warning disable 472 // Disable "CS0472 The result of the expression is always 'false' since a value of type 'Int32' is nev
#pragma warning disable 1573 // Disable "CS1573 Parameter '...' has no matching param tag in the XML comment for ...
#pragma warning disable 1591 // Disable "CS1591 Missing XML comment for publicly visible type or member ..."
#pragma warning disable 8073 // Disable "CS8073 The result of the expression is always 'false' since a value of type 'T' is never
#pragma warning disable 3016 // Disable "CS3016 Arrays as attribute arguments is not CLS-compliant"
#pragma warning disable 8603 // Disable "CS8603 Possible null reference return"

namespace ApiConsumer
{
    using System = global::System;

    [System.CodeDom.Compiler.GeneratedCode("NSwag", "13.15.10.0 (NJsonSchema v10.6.10.0 (Newtonsoft.Json v13.0.0.0))")]
    8 references
    public partial class PrintersClient
    {
        private string _baseUrl = "https://localhost:44366";
        private System.Lazy<Newtonsoft.Json.JsonSerializerSettings> _settings;

        7 references
        public PrintersClient()
        {
            _settings = new System.Lazy<Newtonsoft.Json.JsonSerializerSettings>(CreateSerializerSettings);
        }

        1 reference
        private Newtonsoft.Json.JsonSerializerSettings CreateSerializerSettings()
        {
            var settings = new Newtonsoft.Json.JsonSerializerSettings();
            UpdateJsonSerializerSettings(settings);
            return settings;
        }

        12 references
        public string BaseUrl
        {
            get { return _baseUrl; }
            set { _baseUrl = value; }
```

Figure 10.22: Result of NSWAG - APIConsumer.cs file

With the API-Consumer we could quickly declare a client based of the controllers in ASPNET Core, and subsequently use the functions available to it. As seen here:



```
// Consumption of API
var client = new UserClient();
try
{
    // Email availability
    var EmailResponse = await client.CheckAvailEmailAsync(email);
    Debug.WriteLine(EmailResponse);
    try
    {
        // POST User function
        var PostUserResponse = client.PostUserAsync(firstname, lastname, email, phonenumber, password, confirmPassword);
        Debug.WriteLine(PostUserResponse);

        await DisplayAlert("User account created!","Returning to login screen", "Ok");
        ClearInputs();


        // Navigation to Login
        var previousPage = Navigation.NavigationStack.LastOrDefault();
        await Navigation.PushAsync(new LoginView());
        Navigation.RemovePage(previousPage);
    }
    catch(Exception UserEx)
    {
        await DisplayAlert("Invalid Input", UserEx.Message, "Try again");
        ClearInputs();
    }
}
catch(Exception EmailEx)
{
    await DisplayAlert("Invalid Input", EmailEx.Message, "Try again");
    ClearInputs();
}
```

Figure 10.23: Signup code-snippet

Chapter 11

Discussion

11.1 Project goals and requirements

Two of the biggest difficulties we encountered was time and uncertainty. Since the group did not have any prior experience with 3D-printing, we were unsure of exactly how they operated. Due to this we had to spend some time understanding how exactly one could communicate with a 3D-printer from an external source such as a computer. Additionally since we chose .NET MAUI as our framework of choice, we had to spend time learning how it worked and how it all connects together. Both of these factors, as well as other minor reasons, all leads to our struggle with time. Given a few extra weeks to compensate for the time spent simply understanding the tools we had to use and the technology we had to work with, we could potentially have finished all the way up to phase three. Since most of the time went to satisfying the requirements that was agreed upon with the employers, perhaps it blinded us to other, potentially better solutions? It is hard to determine whether or not setting a list of absolute requirements detracted from our ability to think creatively and find quick and effective solutions to issues.

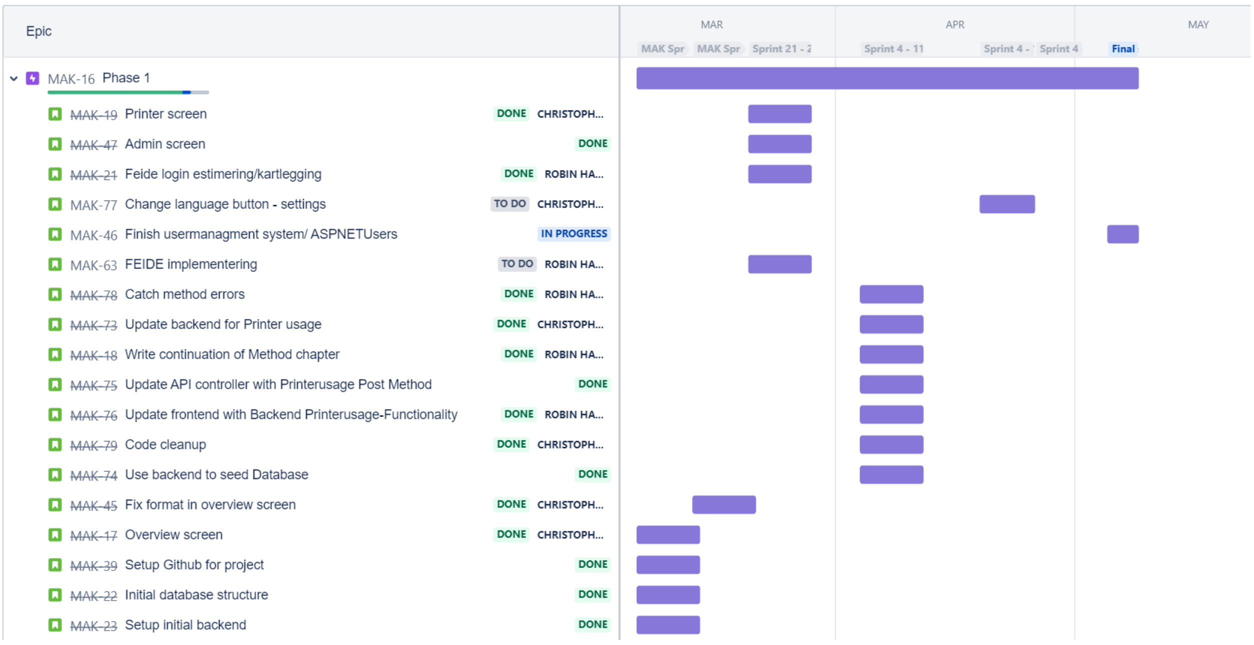
11.2 Planning

The planning phase of the project went wonderfully, a lot of time was spent discussing and planning how exactly to complete the assignment. We spent a considerable amount of time during our first meeting with our employer discussing how they wanted the program to work and if it was possible for us to complete their request. Due to this we were able to construct a solid image of what we wanted our program to do, and how it would work. Because of this extensive planning and because of the good the use of tools such as Enterprise architect or Jira, we never had to stop and think of how we wanted the final product to end up during the development phase. Therefore we could spend all our time and energy on developing the program without having to stop and think of how it should work. However this does add the question, what if we hadn't used the agile method? Could we have gained more from utilizing another method that perhaps was more suited for our task, but which we didn't go for because we were more used to the Agile method?

Time-tracking was a high priority, and as we worked with Jira to document our sprints we made sure to time-track to the best of our abilities, in which we ended up being moderately successful. Although we generally remembered to time-track, we occasionally forgot to do so. When this happened we made sure to compensate by adding time lost at the next time-track. Additionally the decision to use Jira eventually turned out to be a helpful choice. After implementing the "printer history" functionality we decided to give an update to our Signature Makerspace employers. They where generally very happy with our progress, but noted they wished they could somehow use Overseer to report errors to the system's admins. Luckily since we were using the Agile method we simply added the report feature as an issue in the next sprint. The report system is now properly implemented and users are easily able to report errors to the admins. This would have been far more difficult to implement if we were not following the Agile method, since the Agile method allows for quick changes and rewards communication with the employer. Then perhaps Agile was the correct choice after all? Considering we had very few issues during the planning phase of this project and most problems occurred due to the choice of .NET MAUI, we believe that Agile was the right choice.

11.3 Gantt schema

This Gantt schema lays out our sprints in Jira. The reason the schema is structured in such a way is because Jira sorts by sprints, rather than days or weeks, leading to this odd format. Nevertheless it shows different issues that were created, including whether or not it was completed. The gaps in progress are caused by outside factors, stopping us from working on the project. When this happened we made sure to "make back lost time" by putting in extra work on the next sprint to complete issues that has to be moved to the backlog. Admittedly this Gantt schema is somewhat lacking, and at first glance appears quite sparse. It was not possible to display the myriad of sub-issues, leading to a sparse and empty looking schema Considering the limitations of Jira's schema it would have been better to create a custom schema using another external program. However the schema still it gives a general idea of progress, as the different issues are completed in its respective sprint.



Epic			
MAK-16	Phase 1		
~~MAK-19~~	Printer screen	DONE	CHRISTOPH...
~~MAK-47~~	Admin screen	DONE	
~~MAK-21~~	Feide login estimering/kartlegging	DONE	ROBIN HA...
MAK-77	Change language button - settings	TO DO	CHRISTOPH...
MAK-46	Finish usermanagment system/ ASPNETUsers	IN PROGRESS	
MAK-63	FEIDE implementering	TO DO	ROBIN HA...
~~MAK-78~~	Catch method errors	DONE	ROBIN HA...
~~MAK-73~~	Update backend for Printer usage	DONE	CHRISTOPH...
~~MAK-18~~	Write continuation of Method chapter	DONE	ROBIN HA...
~~MAK-75~~	Update API controller with Printerusage Post Method	DONE	
~~MAK-76~~	Update frontend with Backend Printerusage-Functionality	DONE	ROBIN HA...
~~MAK-79~~	Code cleanup	DONE	CHRISTOPH...
~~MAK-74~~	Use backend to seed Database	DONE	
~~MAK-45~~	Fix format in overview screen	DONE	CHRISTOPH...
~~MAK-17~~	Overview screen	DONE	CHRISTOPH...
~~MAK-39~~	Setup Github for project	DONE	
~~MAK-22~~	Initial database structure	DONE	
~~MAK-23~~	Setup initial backend	DONE	

Figure 11.1: Gantt schema of our progress

11.4 Data handling

Following the privacy act was of utmost importance to the group, and as such large amounts of time were spent researching and attempting to understand the rules and regulations pertaining to personal information. User's personal information is saved at all times in the database, connected to the account they created when registering to Overseer. However, due to the right to deletion the users should be able to alter or delete their account at any time [35]. The law states that systems are allowed to keep information for as long as there is a clear purpose for it [76]. It was decided that keeping the information for one week would be sufficient, as possible errors, sabotage or mistakes would have been caught by the admins by that time. Therefore, after a week the information stored should be deleted permanently, as it no longer serves any use and thus we no longer have valid reason to keep the information. However, since the user agreed to the terms of service upon registering an account, if the user attempts to delete their account whilst their information is stored in the history tab they should not be able to do so immediately. The terms of service should give us the right to keep their information for up to a week after usage of our system, therefore there must be a system in place that deletes the information, and the account, after the agreed to week has passed. Regretfully, even though we understand what features must be implemented so as to follow the law, we were not able to do so in time. As of right now the program does not have a properly implemented terms of service, and users may not delete their accounts if they so wish. This is a clear violation of the right to be forgotten [35] and the privacy act [32] [33] [76].

Considering all the rules and laws behind online privacy it could perhaps have been preferable to implement a system that does not require log-in whatsoever. A system where users simply type in their information when they use a printer, instead of having to log in would have been far simpler to implement and would require less careful threading in regards to laws and regulations. At the very least, implementing a custom login service served as a good practice in following and adhering to privacy laws. It taught us the importance of TOS and how many rights users must have for the program to be legal.

11.5 Implementation

Here is a list of all major implementations that we failed to implement in time, ordered by importance:

- Terms of service and deletion of users
- Authentication of API using JWT tokens
- Finished log-in system (Feide implementation)
- Authentication of emails during registration
- Direct connection between Overseer and 3D-printer's storage
- Overview web page

Theoretically we are also missing all major features required for phase four to work. However from what we researched phase four seemed incredibly difficult and highly impractical for us to implement and was therefore heavily down-prioritised. Additionally we had several issues with .NET MAUI's cross platform functions, and suffered several crashes and errors trying to fix it. Eventually we made the decision that due to there being very little documentation to make use of on the internet, trying to fix the errors and crashes would be an inefficient use of time. Therefore our program is temporarily only compatible with Windows machines. The decision to drop cross-platform compatibility somewhat nullifies the whole point of picking .NET MAUI. Even so it is possible to implement cross platform compatibility in the future, which will hopefully be simpler and more stable in the full release of .NET MAUI.

Additionally, as mentioned in "Data handling (11.4)" the decision to implement a custom login system might not have been the correct choice, even if we stand by our current choice. Theoretically we could have implemented a simpler and faster system in where the user simply enters their information whenever they go to use a printer. This bears some other negatives though. A malicious user could enter the wrong information, and use different information each time, which would make it hard to catch or block any single person. However if the login system was completed, this malicious user would have to create accounts using authentication (not implemented, but on the list) which would check if the email was real. This barrier of having to create new legitimate email for every time they use the printer would hopefully deter them from doing malicious acts. Finally, the employers originally wanted the ability to log in using student-cards. This was dropped due to complications with UIA's policies on granting this ability away. We discussed with UIA for a while but ultimately it was decided that it would take to much time, not only to implement, but to gain the rights to even use the card in the first place. The employer, although saddened, agreed that it would be a poor use of time attempting to gain the rights to use the student cards. Therefore they agreed to a normal login system, which is what we ended up half-implementing.

11.6 What could have been done different?

In hindsight choosing .NET MAUI, which was in pre-release and in which we had no experience with might not have been the most optimal choice. Our reasoning for choosing MAUI was the future proofing of picking the newest framework, as well as its support for multiple platforms. However perhaps it would have been better to pick a more stable and well documented framework such as WPF as it could have made the implementation process smoother. With this we would theoretically have more time that would not have been spent trying to learn MAUI or fix its pre-release problems. Even so, we believe that in the grand scheme of things picking MAUI was the right choice for us, but not the right choice for Signature Makerspace as Overseer was not finished in time.

Time management in general could have been better. Although we believe our use of Jira and our consistent time-tracking was good, we fell into the trap of believing we had more time than we actually had. Despite our large amount of planning and preparing, the difficulty of learning a new framework as well as matching the requirements of our employer took us off guard, resulting in time lost. If we had realised how much time was required to properly create Overseer we would have begun with "crunch time" earlier on, rather than waiting til so late in the projects timeline to ramp up.

11.7 Uniqueness and market

Although aspects of our product bears similarities to OctoPrint by Gina Häußge [77] specifically phase four, there could have been several aspects of our program that would make it stand out from its potential competitors. These include Feide login, student card login, ability to oversee multiple printers (OctoPrint can do this, but is not designed for it) [78]. These unfinished features would have made Overseer stick out as a great alternative to monitor usage of multiple printers, specifically for school-makerspace environments. Most importantly however, is the concept of the program itself. Overseer is designed as a monitoring and management system, not meant to control or change the printers in any way. OctoPrint on the other hand is more for remote control of printers, not meant for logging when the printer was used and by whom. Regretfully we were not able to implement any features that would make Overseer stand out that much, besides it being custom made for Signature Makerspace with the possibility of being opened for other clients in the future. There is a market for Overseer, even if it might be small for now. Potentially if makerspaces become more mainstream and see more use in the future, Overseer could fill a niche allowing makerspace admins more control over who is using the printers.

11.8 Further work

As discussed earlier there are several features that is currently missing from Overseer. Future work would definitely be centered around completing and implementing these features. Features such as the the TOS missing, users not being able to delete their own account and authentication when using the API would be the top priority. After important and half-finished features have been properly implemented we would move on to continuing work on phase two. This entails connecting 3D-printer's storage directly into Overseer so users may transfer the STL file directly from Overseer, removing the need to connect to the printer's storage manually. Then, eventually we would move on to phase three, and afterwards try to implement other features the employers wished for but that we were unable to implement at the time. The two biggest features we had to abandon is a Feide login and the ability to use student-cards to log-in instead of manually typing in the username and password. After these features are implemented we might even attempt to implement phase four. This would most likely involve a lot of changes in the setup of the project and would without a doubt be the most difficult part of our future work.

Chapter 12

Conclusion

Generally we are content with how the project turned out, and although several features are missing we are still overall happy with the result. Considering we are only two students with no experience in neither MAUI, 3D-printers or having an "employer" to work for we wholeheartedly believe we did an acceptable job. Although we wanted to finish at least phase two, phase one covered all requirements put out by Signature Makerspace, and so we would be happy to at least meet their requirements.

The requirements set by us early in the project led to us having a clear goal to strive for. Requirements such as:

- User-management
- Login system
- History of 3D-Printer usage
- Custom UI using Signature Makerspace's colors.
- The ability to add/customize/remove printers and users.

always ensured we had important features in mind when designing Overseer. Despite this there were some features that for one reason or another fell to the wayside. Unfortunately larger features such as student-card login or Feide login were abandoned due to limited time.

This project has posed many challenges to overcome, though we found these challenges fun and interesting. The planning phase of the project went quite smoothly, and although we were excited to get started implementing a solution we were able to fully finish planning before doing so. This turned out to be a huge boon for future work as we always had a solid foundation to work from. Even though Overseer is currently unfinished, we are proud of our work, and believe that with just a few extra weeks worth of work would turn into a fully functional management system for Signature Makerspace.

Chapter 13

References

Bibliography

[1] Microsoft. *What is .NET?* Accessed on 27.04.2022. URL: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet.

[2] mulesoft. *What is an API? (Application Programming Interface)*. Accessed on 03.05.2022. URL: https://www.mulesoft.com/resources/api/what-is-an-api.

[3] O. V. Gorp. *How to Consume APIs With Akana*. Accessed on 03.05.2022. URL: https://www.akana.com/blog/consume-apis.

[4] Microsoft. *What is ASP.NET Core?* Accessed on 03.05.2022. URL: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core.

[5] TechTarget. *API endpoint*. Accessed on 08/05/2022. URL: https://www.techtarget.com/searchapparchitecture/definition/API-endpoint.

[6] Sparx Systems. *Benefits and Features*. Accessed on 19.02.2022. URL: https://sparxsystems.com/enterprise_architect_user_guide/15.2/benefits_and_features/benefits_and_features.html.

[7] Entityframeworktutorial. *What is Entity Framework?* Accessed on 28.04.2022. URL: https://www.entityframeworktutorial.net/what-is-entityframework.aspx.

[8] Gantt. *Hva er et Gantt-diagram?* Accessed on 07/05/2022. URL: https://www.gantt.com/no/.

[9] Git-scm. *Git*. Accessed on 12/04/2022. URL: https://git-scm.com/.

[10] GitHub. *Github*. Accessed on 12/04/2022. URL: https://github.com/.

[11] Atlissian. *Jira software*. Accessed on 12/04/2022. URL: https://www.atlassian.com/software/jira?&aceid=&adposition=&adgroup=95003656689&campaign=9124878870&creative=542638212689&device=c&keyword=jira&matchtype=e&network=g&placement=&ds_kids=p51242194730&ds_e=GOOGLE&ds_eid=700000001558501&ds_e1=GOOGLE&gclid=Cj0KCQjwsdiTBhD5ARIsAIpW8CKfnCqDMD5671FKCsjixQ4VkDcbtxVLLrq_0UoBSA_t6EZlohP-MZMaAlAWEALw_wcB&gclsrc=aw.ds.

[12] Json org. *Introducing JSON*. Accessed on 08/05/2022. URL: https://www.json.org/json-en.html.

[13] Makerspaces.com. *What is a Makerspace?* Accessed on 11/03/2022. URL: https://www.makerspaces.com/what-is-a-makerspace/.

[14] Microsoft. *What is .NET MAUI?* Accessed on 27.04.2022. URL: https://docs.microsoft.com/en-us/dotnet/maui/what-is-maui.

[15] Microsoft. *Download SQL Server Management Studio (SSMS)*. Accessed on 03.05.2022. URL: https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15.

[16] scaledagileframework. *Nonfunctional Requirements*. Accessed on 11/03/2022. URL: https://www.scaledagileframework.com/nonfunctional-requirements/.

[17] R. Suter. *NSwag: The Swagger/OpenAPI toolchain for .NET, ASP.NET Core and TypeScript*. Accessed on 12/04/2022. URL: https://github.com/RicoSuter/NSwag.

[18] António Silva. *How to Write Meaningful Quality Attributes for Software Development*. Accessed on 27.04.2022. URL: https://www.codementor.io/@antoniopfesilva/how-to-write-meaningful-quality-attributes-for-software-development-ez8y90wyo.

[19] RedHat. *What is a REST API?* Accessed on 12/04/2022. URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[20] Wikipedia. *Representational state transfer*. Accessed on 12/04/2022. URL: https://en.wikipedia.org/wiki/Representational_state_transfer.

[21] Swagger. *What is Swagger?* Accessed on 11/03/2022. URL: https://swagger.io/docs/specification/2-0/what-is-swagger/.

[22] N. Daly. *What Is a Use Case?* Accessed on 11/03/2022. URL: https://www.wrike.com/blog/what-is-a-use-case/.

[23] M. Rehkopf. *User stories with examples and a template*. Accessed on 11/03/2022. URL: https://www.atlassian.com/agile/project-management/user-stories.

[24] Microsoft. *It's how you make software.* Accessed on 16/02/2022. URL: https://visualstudio.microsoft.com/.

[25] RedHat. *What is YAML?* Accessed on 08/05/2022. URL: https://www.redhat.com/en/topics/automation/what-is-yaml.

[26] Signature Makerspace. *What is makerspace?* Accessed on 08/05/2022. URL: https://www.signaturemakerspace.com/.

[27] John Adam. *What is Agile software development?* Accessed on 19.03.2022. URL: https://kruschecompany.com/agile-software-development/.

[28] Rachaelle Lynn. *Disadvantages of Agile.* Accessed on 19.03.2022. URL: https://www.planview.com/no/resources/articles/disadvantages-agile/.

[29] Asana. *Work on big ideas, without the busywork.* Accessed on 08/05/2022. URL: https://asana.com/?utm_campaign=Brand--NO--EN--Core--EX&utm_source=google&utm_medium=pd_cpc_br&gclid=Cj0KCQjw1N2TBhCOARIsAGVHQc42AOljW7jYwjrYITSD9vpVLJqrqwKpWxFKEMWJKjhfIW1r
wcB&gclsrc=aw.ds.

[30] Microsoft. *Web Languages.* Accessed on 26.04.2022. URL: https://visualstudio.microsoft.com/vs/features/web/languages/.

[31] Datatilsynet. *Ha behandlingsgrunnlag.* Accessed on 20.04.2022. URL: https://www.datatilsynet.no/rettigheter-og-plikter/virksomhetenes-plikter/behandlingsgrunnlag/.

[32] Datatilsynet. *Personalmappe - hva lagres, hvor lenge og hvorfor?* Accessed on 20.04.2022. URL: https://www.datatilsynet.no/personvern-pa-ulike-omrader/personvern-pa-arbeidsplassen/personalmappe/.

[33] Lovdata. *Lov om behandling av personopplysninger (personopplysningsloven).* Accessed on 20.04.2022. URL: https://lovdata.no/dokument/NL/lov/2018-06-15-38/KAPITTEL_gdpr-3-3#gdpr/a17.

[34] Logpoint. *En oversikt over personvernforordningen (GDPR).* Accessed on 20.04.2022. URL: https://www.logpoint.com/no/forsta/en-oversikt-over-gdpr/?tr_campaign=en_no_LP7_launch&gclid=Cj0KCQjwpcOTBhCZARIsAEAYLuVBeKj25FuT4_ItP2dDeN5ZGNR5cXOFV07wzyv7VEc2qLa
wcB.

[35] Datatilsynet. *Rett til sletting.* Accessed on 20.04.2022. URL: https://www.datatilsynet.no/rettigheter-og-plikter/den-registrertes-rettigheter/rett-til-sletting/.

[36] EU - European commission. *Data protection in the EU.* Accessed on 20.04.2022. URL: https://ec.europa.eu/info/law/law-topic/data-protection/data-protection-eu_en.

[37] Microsoft. *Introduction to Identity on ASP.NET Core*. Accessed on 01/04/2022. URL: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-6.0&tabs=visual-studio.

[38] Microsoft. *What is ASP.NET Core?* Accessed on 03.05.2022. URL: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core.

[39] Microsoft. *Desktop Guide (WPF .NET)*. Accessed on 01/04/2022. URL: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-6.0.

[40] Microsoft. *What's a Universal Windows Platform (UWP) app?* Accessed on 01/04/2022. URL: https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide.

[41] Microsoft. *Xamarin*. Accessed on 01/04/2022. URL: https://dotnet.microsoft.com/en-us/apps/xamarin.

[42] Microsoft. *Desktop Guide (Windows Forms .NET)*. Accessed on 01/04/2022. URL: https://docs.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-6.0.

[43] Microsoft. *Why is WinForm still not dead? — Should we learn WinForm?* Accessed on 01/04/2022. URL: https://medium.com/coderes/why-is-winform-still-not-dead-should-we-learn-winform-5d776463579b.

[44] R. Peterson. *What is PostgreSQL*. Accessed on 01/04/2022. URL: https://www.guru99.com/introduction-postgresql.html.

[45] A. Yigal. *Sqlite vs. MySQL vs. PostgreSQL: A Comparison of Relational Databases*. Accessed on 01/04/2022. URL: https://logz.io/blog/relational-database-comparison/.

[46] C. Purdy. *What is so bad about Oracle?* Accessed on 01/04/2022. URL: https://www.quora.com/Whats-so-bad-about-Oracle-Whenever-engineers-I-know-say-Oracle-it-seems-to-mean-the-worst-place-to-work-and-you-should-never-do-that-Why-is-that.

[47] W3Schools. *MySQL Advantages And Disadvantages*. Accessed on 01/04/2022. URL: https://www.w3schools.blog/mysql-advantages-disadvantages.

[48] Oracle. *Oracle Project: MySQL*. Accessed on 01/04/2022. URL: https://www.oracle.com/webfolder/college-recruiting/projects/mysql.html#.Ynh5kejP1PY.

[49] ThinAutomation. *SQLite pros and cons: a <600-word overview*. Accessed on 01/04/2022. URL: https://www.thinkautomation.com/our-two-cents/sqlite-pros-and-cons-a/.

[50] Codecademy. *What is .NET?* Accessed on 01.05.2022. URL: https://www.codecademy.com/article/what-is-net.

[51] Andrea Chiarelli. *What is .NET? An Overview of the Platform*. Accessed on 01.05.2022. URL: https://auth0.com/blog/what-is-dotnet-platform-overview/.

[52] Microsoft. *What's new in .NET 6*. Accessed on 08/05/2022. URL: https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-6.

[53] B. Hildenbrand. *Friday, May 10, 2019 .NET Core and .NET Framework merging as .NET 5.0*. Accessed on 11/03/2022. URL: https://blog.hildenco.com/2019/05/net-core-and-net-merging-as-net-50.html.

[54] K. Nakamura. *.NET 5 : How Microsoft merge .NET Framework and .NET Core*. Accessed on 11/03/2022. URL: https://dev.to/kenakamu/net-5-how-microsoft-merge-net-framework-and-net-core-525m.

[55] Altexsoft. *The Good and the Bad of .NET Framework Programming*. Accessed on 11/03/2022. URL: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-net-framework-programming/.

[56] tutorialspoint. *MVC Framework - Introduction*. Accessed on 11/03/2022. URL: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm.

[57] tutorialspoint. *Entity Framework - Overview*. Accessed on 11/03/2022. URL: https://www.tutorialspoint.com/entity_framework/entity_framework_overview.htm.

[58] M. Choudhari. *Know Everything About .NET EF Core Migrations*. Accessed on 11/03/2022. URL: https://thecodeblogger.com/2021/07/24/know-everything-about-net-ef-core-migrations/.

[59] SmartBear. *What Is Swagger?* Accessed on 08/05/2022. URL: https://swagger.io/docs/specification/2-0/what-is-swagger/.

[60] SmartBear. *Swagger UI*. Accessed on 08/05/2022. URL: https://swagger.io/tools/swagger-ui/.

[61] SmartBear. *Swagger Codegen*. Accessed on 08/05/2022. URL: https://swagger.io/tools/swagger-codegen/.

[62] RicoSuter and many more contributors. *NSwag*. Accessed on 03.05.2022. URL: https://github.com/RicoSuter/NSwag.

[63] Saineshwar Bageri. *How To Use NSwag With ASP.NET Core*. Accessed on 28.04.2022. URL: https://www.c-sharpcorner.com/article/how-to-use-nswag-with-asp-net-core-and-generate-client-code-with-nswag-studio/.

[64] Microsoft. *SQL Server 2019*. Accessed on 12/04/2022. URL: https://www.microsoft.com/nb-no/sql-server/sql-server-2019.

[65] Clariontech. *Jira Software - 15 Important Features You Might Have Missed.* Accessed on 28.04.2022. URL: https://www.clariontech.com/platform-blog/jira-software-15-important-features-you-might-have-missed.

[66] Opensource. *What is a Raspberry Pi?* Accessed on 03.05.2022. URL: https://opensource.com/resources/raspberry-pi.

[67] Raspberry Pi Foundation. *Raspberry pi.* Accessed on 03.05.2022. URL: https://www.raspberrypi.org/.

[68] J. Kanjilal. *Implementing JWT Authentication in ASP.NET Core 5.* Accessed on 08/05/2022. URL: https://www.codemag.com/Article/2105051/Implementing-JWT-Authentication-in-ASP.NET-Core-5.

[69] G. Levin. *4 Most Used REST API Authentication Methods.* Accessed on 08/05/2022. URL: https://blog.restcase.com/4-most-used-rest-api-authentication-methods/.

[70] Auth0. *Introduction to JSON Web Tokens.* Accessed on 08/05/2022. URL: https://jwt.io/introduction.

[71] auth0. *JSON Web Token Structure.* Accessed on 08/05/2022. URL: https://auth0.com/docs/secure/tokens/json-web-tokens/json-web-token-structure.

[72] Akana. *What is JWT?* Accessed on 08/05/2022. URL: https://www.akana.com/blog/what-is-jwt.

[73] S. Singh. *Implement JWT Authentication in Asp.net Core 5 Web API [Token Base Auth].* Accessed on 08/05/2022. URL: https://codepedia.info/jwt-authentication-in-aspnet-core-web-api-token.

[74] G. Jangid. *How To Use JWT Authentication With Web API.* Accessed on 08/05/2022. URL: https://www.c-sharpcorner.com/article/how-to-use-jwt-authentication-with-web-api/.

[75] Microsoft. *XAML styles.* Accessed on 08/05/2022. URL: https://docs.microsoft.com/en-us/windows/apps/design/style/xaml-styles.

[76] Datatilsynet. *Fastsette formål.* Accessed on 20.04.2022. URL: https://www.datatilsynet.no/rettigheter-og-plikter/virksomhetenes-plikter/fastsette-formal/.

[77] Gina Häußge. *OctorPrint.* Accessed on 06.05.2022. URL: https://octoprint.org/.

[78] C. Favela. *OctoPrint for Multiple Printers: How to Get Started.* Accessed on 08/05/2022. URL: https://all3dp.com/2/octoprint-multiple-printers/.