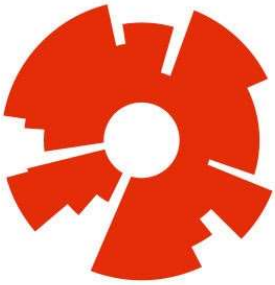

HÁSKÓLINN Í REYKJAVÍK
REYKJAVIK UNIVERSITY

UteamUP Horizon

BSc. Computer Science

Final Report

Gísli Guðmundsson
Garrett Forthofer
Jóhanna Antonía Sigsteinsdóttir
Unnar Friðrik Sigurðsson

Instructor: Hallgrímur Arnalds
Evaluator: Hrefna Sigríður Briem
Supervisor: Birgir Kristmannsson

Abstract

The project was conducted in collaboration with UteamUP in the hopes of creating a Computer Maintenance Management System, colloquially referred to as CMMS software, called *Horizon*. The system serves to track maintenance of assets and foresee problems before they might occur with the aim to enhance the quality and lifetime of machinery. The secondary, albeit still an important benefit, is to make the workplace a safer and more efficient environment. This project aims to modernize existing systems' approaches, bringing them into the digital age as many of these existing systems have yet to make that leap.

In this report we outline our approach to the project itself, while many of the detailed description of processes can be found in several accompanying documents for the project, such as our *Design Report, Developer Handbook, Operating Manual* and *User Manual*.

Foreword and Acknowledgements

This project was conducted from the middle August to the beginning of December 2021 in collaboration with UteamUP as a final project for B.Sc. degree in Computer Science at Reykjavík University.

We would like to thank UteamUP for enabling us to carry out this project and bring it into the professional world. Furthermore, we would like to thank Hallgrímur Arnalds for his course supervision. The lectures that were given at the beginning of the semester were extremely useful in our initial organization of the project. To our supervisor, Birgir Kristmannsson, we would like to extend a special thank you. His guidance and feedback during the semester were invaluable in helping us to stay the course and deliver a product which we are proud of.

Table of Contents

1 Introduction

The roots of the project began as part of the course “Nýsköpun og stofnun fyrirtækja” at Reykjavík University. From there UteamUP was founded by current alumni and the seeds for our project *Horizon* were planted.

One of the major challenges in managing, operating, and maintaining machinery is logistics. The ability to keep track of assets and their status is a huge task. Current methods of management with large inventories are often archaic, some not even digital. Such systems are prone to human error and inefficiency in lookup time. UteamUP aims to offer a software solution that looks to the future; a new and easy way to view and manage inventories of assets, parts, networks of vendors, and workorders. With a simple user interface, employees and administrators have a consistent point of contact with the system, minimizing the aforementioned issues.

For our project we attempted to design and implement a software solution to solve these tasks based on requirements and feedback from UteamUP and the resulting *Horizon* application is a Computerized Maintenance Management System (CMMS) for organizational solutions. Its core functionality is to track assets or machines, spare parts, vendors, and workorders to facilitate smooth and reliable operations in a machine heavy environment.

In this report we will go over how we tackled this task over a fifteen-week period, the results of our work and what lessons we learned from the process.

We used the Scrum framework to keep track of our progress and to ensure that we were staying within our goals and defined four sprints. A “pre-sprint” sprint was used to help us familiarize ourselves with the project, create a mockup and garner user feedback on it. This sprint was followed by four programming sprints, and we then used the final weeks to prepare and finalize all the documentation for the project.

2 UteamUP Horizon

Our project contained two types of software engineering: a backend, and a frontend user interface. The backend communicates with a relational database to save information and makes data available to the frontend. The frontend is the link between the user and the backend. The backend is created based on the RESTful API Design Methodology, which helps the developer standardize the application communication between the frontend and the backend.

For the system to function correctly it must have several types of objects for the user to interface with: assets, parts, vendors, users, and workorders. The assets require parts and are one-to-many connections, that help the user identify what parts are required to operate the asset. The vendor is connected one-to-one, both to the part and the asset. The lynchpin in the whole process are workorders which serve to organize when and how the assets should be managed, such as if the asset requires maintenance or spare parts and who should carry out the task.

In the beginning the plan was to use Python's FastAPI library for the backend, Vue.js for the frontend and a PostgreSQL database. However, we soon realized that this route might not be the most suitable for us to take and after some consideration we settled on a PostgreSQL database, .NET framework for the backend API and Blazor for the frontend. The most obvious benefit to us was that by using .NET and Blazor, we would be able to flatten the learning curve somewhat since the C# programming language is used in both. Another benefit, which we did not completely foresee in the beginning, was the added benefit provided by Syncfusion. Syncfusion has many pre-made components for Blazor and is free for startup companies, so we ended up securing a license and utilizing a few of their components in *Horizon*. A more thorough and detailed review of *Horizon* can be found in the *Horizon Developer Manual* which covers the application's structure, components, dependencies, and coding rules.

Probably the most benefit we got from these framework decisions was the added time it allowed us for development. If we had gone another route, we might have had to reduce the scope of the project due to the steeper learning curve.

3 Project Management

For a project to work as expected it needs good project management. It is imperative to have good organization since wasted hours quickly accumulate and can completely derail a project. We were cognizant of this fact and spent some time in the beginning outlining and planning how we would progress the work.

The group decided to use Scrum as a project management methodology. The project was divided into a total of four sprints in addition to our pre-sprint which was used to familiarize ourselves with the project as well as conducting the first user testing based on the mockup. Our estimation for time spent working on the project is around 320 hours per member. With that in mind, a time schedule was set up so that each member could deliver around 17 hours per week and up to 25 hours in the last few weeks of the project.

We encountered some obstacles along the way and had to make some changes. In the beginning of the project, we had no set days for working on the project and the group seldomly met up in person. However, after a couple of weeks it became apparent that this format was not as efficient as we had hoped, and the group reached a consensus that we would meet up in person at least two days per week to work on the project together. Ultimately this proved to be a success and on reflection helped our teamwork greatly. Rather than sending messages to each other regarding any issues or problems we were currently working on, we could talk about them in person and solve in unison.

3.1 Methodology

We chose Scrum as our project management methodology. The reason we chose this methodology is that we all have had previous experience with it having seen and used it in various courses throughout our studies.

At its core, Scrum is a framework for working on projects in an iterative manner. A product backlog is used, and each sprint should try to add value to the product being developed. Therefore, value adding components find their way into each sprint and the product develops incrementally. It is an excellent way to break down complex projects into smaller components and enables teams, such as ours, to divide the work into manageable chunks.

For our project we used a rather loose form of Scrum. We defined sprints and a backlog as well as assigning a Scrum master to keep track of the progress and make sure we were keeping up with the schedule but did not delve deep into the finer details the framework also offers.

We used Azure for our product backlog and sprint organization. It enabled us to define sprints and tasks for each sprint in a reasonably simple manner. There was a slight learning curve which we had to deal with in the beginning, but we were quick to get to grips with it and in the end, we were using it without any issues. Although this document is in no way meant as a review of the tools we used for the project, we might note that the user interface on Azure is rather clunky and some of our team members have remarked that they are unlikely to use it again in the future.

Pre Sprint 16. Aug – 24.Sept	Sprint 1 24. Sep – 05. Oct	Sprint 2 06. Oct – 17.oct	Sprint 3 18.Oct - 29.oct	Sprint 4 30. Oct – 30.Nov
• Mockup design • User testing	• Initializing Azure environment • Initialize API • Creating models for SQL • Setting up the pages based on the mockup	• API connectors from pages to Backend • Structuring pages • CRUD logic • Authentication • Adding relations between models.	• Logic design between backend and frontend • Implementing notifications coverage and unit testing	• Refinement of logic design • Beautification • Second user testing • Bug fixing

Figure 1 - Sprints

3.2 Development

In this chapter we will go over our sprints and try to give an overview of how each sprint went for our project. Rather than providing a dry summary of the tasks that we completed in each sprint, we aim to give a subjective review of how the sprints progressed, what challenges we faced, our team dynamics, what went well, and what we could have perhaps done differently. In the last subchapter we give an overview of the project as a whole and what takeaways we can take from the project into the future.

3.2.1 Pre-sprint

The pre-sprint did not receive its formal definition until after a week or so into the project. We had loosely discussed how we wanted to start the project and what we needed to accomplish before the work could begin in earnest. The key was to design a mockup for the user interface and get some feedback on that so that we might have an idea how to lay everything out in a clear manner for our users.

We set to work and designed an interactive mockup using Figma which would allow users to click on parts of the pages to navigate to other pages connected to it. This gave us a surefire way to see how a person might interact with the interface, where the pain points lie and where we needed to make changes or clarify the layout.

This was quite an undertaking, and when all was said and done, we ended up with 48 separate pages in the mockup, each with navigation possibilities to one or more pages.

Around the middle of September, we had gotten the mockup to a state where we could begin user testing with it. We first got the guys at UteamUP to try it out and give us feedback. Based on their feedback, we then went back to it and made changes and tried to iron out any kinks and get it to flow properly. This was then followed by another round of user testing by two other companies, Mjólkursamsalan and Arnarlax. We gathered more feedback from them and, again, made any necessary changes based on that. Our findings and a more detailed overview of the process and mockup can be found in our *Design Report*.

The mockup was not the only thing that we needed to accomplish in our pre-sprint. Since we had made the decision to use Blazor and .NET for our front and backends, there was also some preliminary work that needed to be accomplished before coding could begin. In this sprint, we also laid the groundwork by defining some API endpoints, connecting to our PostgreSQL database, and creating shared model classes that would be used between both ends. Although a lot of this work would continue in sprint 1, it was necessary to have something up and running so that we would not be starting development on the frontend blindly.

Reflecting on this sprint, it was probably one of the more educational ones. Since we had not settled on a sprint methodology right at the beginning of the course everything was a bit looser at this stage of the project. We did not define a proper sprint with tasks and backlog. The group dynamic was still forming, and we were finding out what would be the best way for us

to work together. It would probably be safe to say that during this sprint we were not cohesive enough and would most likely have benefitted from more frequent in-person meetings. Thankfully we recognized this before it became too late and were able to make adjustments to our workflow which aided our progress going further as well as having a better handle on the organizational side of the project.

3.2.2 Sprint 1

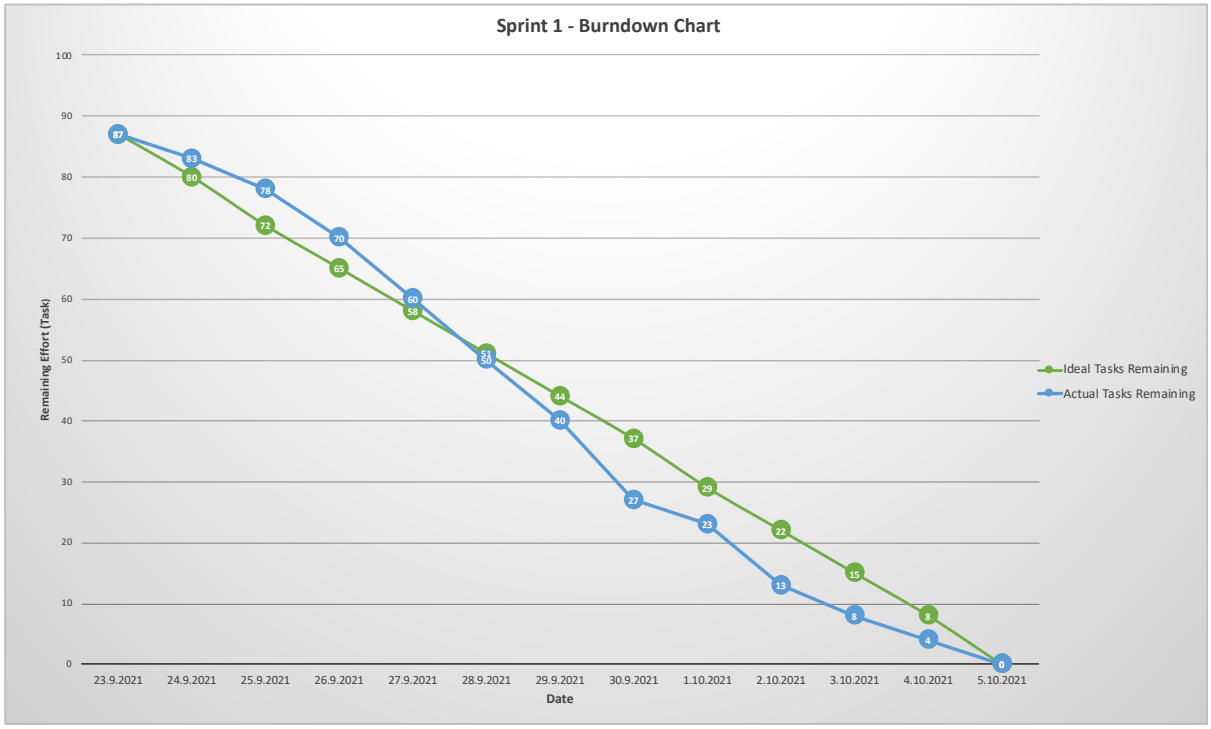

Figure 2 - Sprint 1 burndown chart

In sprint 1 we began programming. On the backend side this meant setting up and configuring our Azure environment, as well as continuing with the API and model classes. On the frontend side we started laying out the Razor pages based on the mockup.

In the beginning we had 87 tasks to complete as can be seen from the burndown chart in Figure 2 - Sprint 1 burndown chart. We divided the total number of tasks with the amount of time to determine the ideal number of tasks to finish with respect to the timeline. At the end of the sprint, we had to move a few items to the next sprint. This was due to unforeseen dependencies that we had not thought of during the pre-sprint.

The work completed for the backend was the completion of the major model relations and related API calls, dealing largely with database migrations. For the frontend all of the major pages and navigation options laid out in the side bar for the mock-up were created with a basic layout and functionality implementation with stand-in data while the API calls were being completed.

We were still coming together as a group during the sprint but by the end of this sprint we had worked out how best to organize ourselves and were increasing our efficiency with each passing day. We continued with our scheduled in-person meetings and workdays.

All things told this sprint was a continuation of the previous sprint in every way. The scope of the project was beginning to dawn on everyone, stress levels were raised a bit and we gained experience in dealing with the unforeseen and finding solutions to progress our work.

3.2.3 Sprint 2

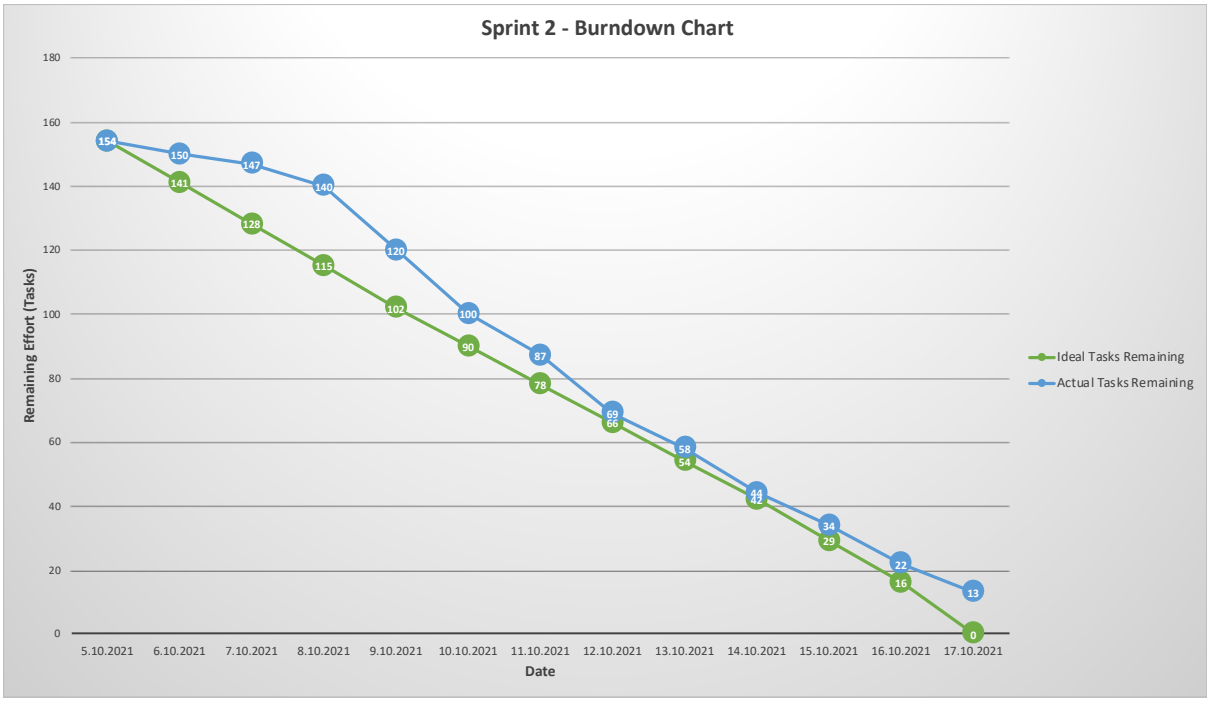

Figure 3 - Sprint 2 burndown chart

We knew that sprint 2 would be a big sprint for us. We had run into some issues with dependencies in sprint 1 that necessitated us moving some tasks to sprint 2. Due to that the

overall number of tasks rose drastically. This was further compounded by the realization that our design and implementation were conflicting in some cases.

The goal of the sprint was to get the API connectors from the backend working with the frontend, continuing structuring our Razor components, implementing CRUD logic on the backend, implementing authentication for *Horizon,* and adding relations between model classes – quite the tall order for one sprint.

However, we managed to pull it off. As you can see from Figure 3 - Sprint 2 burndown chart, we started the sprint with 154 tasks. At the end we had to move 13 tasks over to sprint 3, which considering the situation was a very positive result for us.

The design and implementation conflicts, we were able to resolve through hard work and determination. It took us a while but in the end we got there. Additionally, we were able to shave off some development time by introducing Syncfusion into the mix. Since many of the functionalities that we were looking to implement were already available as Syncfusion components for Blazor, we could save time by not having to implement all the functionality from scratch ourselves. As an added benefit, the Syncfusion components also give the pages a cleaner look and are fairly easy to maintain.

This sprint was definitely one of the more challenging ones. We had a huge number of tasks and development issues to solve and the risk of becoming stuck was ever looming. Thankfully, we had already encountered some difficulties in the project at earlier stages and were a very efficient team by this point. That enabled us to pull through and persevere but during the sprint we were all feeling the pressure of the project and at times we wondered if we were falling behind schedule and if there was a risk that we would be unable to complete the project. To prepare for that eventuality, and to have all our bases covered, we decided that going into sprint 3, we should do a review and decide which functionality was most important in the case that we ran out of development time.

3.2.4 Sprint 3

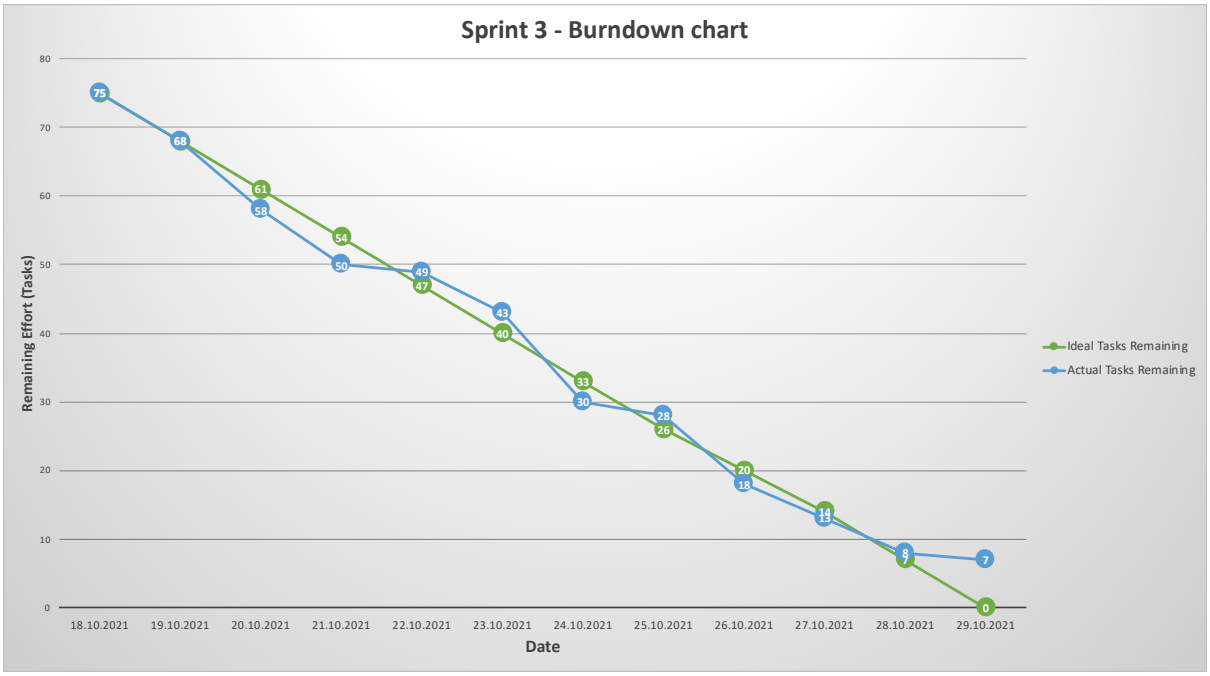

Figure 4 - Sprint 3 burndown chart

When we started with sprint 3, we had to do a review and identify which functionality was most important to implement in case we ran out of development time. The pressure really was on after sprint 2 and even though we were confident in meeting our goals, we still wanted to have a backup plan in case time became an issue and that, in that eventuality, we would only be skipping nice to have functionality rather than any core functionality.

During the sprint the plan was to complete any remaining tasks that we had moved from the previous sprint. Additionally, we needed to implement quite a lot of logic related to user and backend with regards to which data users are allowed to see e.g., a user not being able to get workorders for a tenant (organization) he has no permissions for. We also continued to implement Syncfusion components to replace Razor pages that we had already created, for example, our add / edit pages, and fixing some remaining issues and implementing any remaining page functionality.

For the backend this meant dealing with issues found in the frontend and design conflicts. This would entail many changes in our model relations as we began to examine how to implement the most complex of the API calls and several obstacles related to our many to many relationships. These issues would carry over into sprint 4 for the first few days.

We had 75 tasks at the beginning of the sprint and at the end there were 7 tasks remaining which we moved on to sprint 4 as can be seen from Figure 4 - Sprint 3 burndown chart.

Sprint 3 went by fairly smoothly. We were focused on our tasks, and everyone had a clear idea of what we needed to accomplish and how to get there. We had already been operating at a high efficiency level as a team for some time now, as well as being so far into the project that everyone had a clear understanding of where we were at and where we were heading. We had planned to do unit testing in this sprint but in order to stay on schedule and be able to finish the project, we decided that we would do the testing as user tests rather than unit tests.

3.2.5 Sprint 4

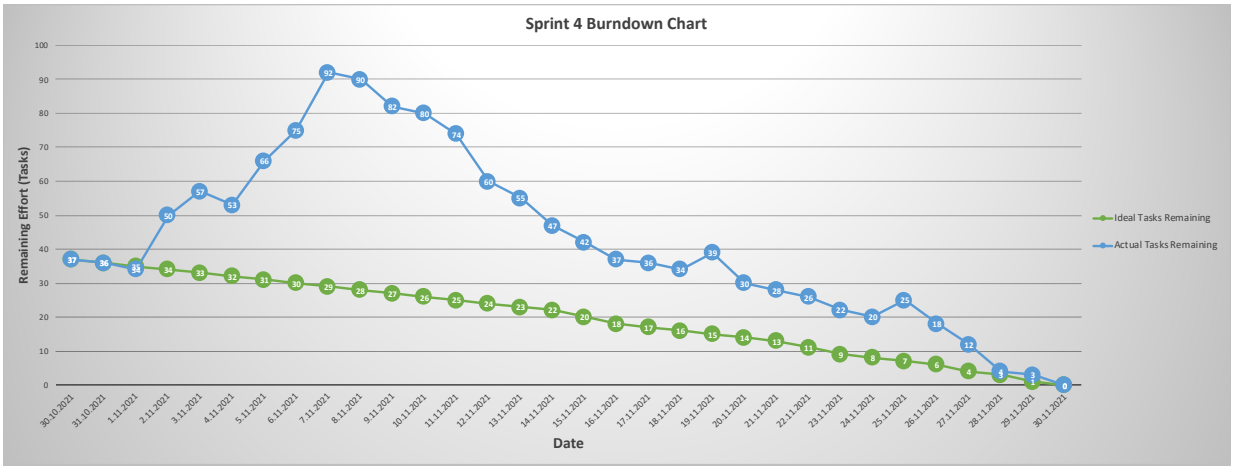

Figure 5 - Sprint 4 burndown chart

Sprint 4 came with the context that it would be our last sprint for development and would require us to complete the design to a functional level for use and presentation. Due to this it would be where the team really banded together and worked diligently in order to finish out the project strong. We would begin with the remaining tasks from sprint 3 and continue to put the final touches on our design. This included a lot of bug fixing leading up to our final period of user testing.

This sprint had long been planned as our beautification and bug fixing sprint but the scope of the sprint was initially concerning. From our perspective in the sprint there always seemed to be more bugs or things not implemented as well as we would have liked them to, but we persevered. This is represented in the growing number of tasks we added as the sprint continued. In addition to this, our initial dates for the end of the sprint ended up cutting us short.

This was due to a miscommunication in the time allotted to work on the project. This resulted in the extension of the sprint to include both the exam period and first week of the 3-week course. By the end we had gone through all of the tasks and were happy with the results going into our second user testing period, the details and results of which may be found in the *Design Report.*

We started the sprint with 37 tasks but as the sprint progressed and we discovered bugs or issues that needed corrected we added new tasks to reflect this as can be seen from Figure 5 - Sprint 4 burndown chart. At its highest we had 92 tasks on the backlog for the sprint but none remaining at the end.

The work completed for this sprint is not as easily separated between the front and backend as the other sprints as the majority of the bugs ended up being a combination of problems in both. This meant finding issues in the frontend and analyzing the errors to determine whether it was an issue with the input from the frontend or where the backend was conflicting with the design.

This sprint was rather stressful for us. As the number of bugs and issues we encountered began to rise, it became apparent to us that this would not be a simple beautification and final touches sprint but that we had larger tasks still to take care of. Being the last sprint there was also no longer a buffer where we had the availability to push any last remaining issues on to the next sprint if we were unable to finish. To add to this, the sprint also came in the middle of the exam period at the university and, therefore, some team members were unable to work on the project for a few days at different times while preparing for their final exams. However, in our case it worked out and we believe that is in large due to the work habits we had formed over the course of the semester which allowed everyone to quickly jump back in and pick up where they last left off.

3.2.6 Sprint retrospective and takeaways

Forming a group with individuals for a project can be a challenging task. In the beginning the group does not know each other well, communication can often be hampered by group members not having the confidence to express themselves fully, disagreements and clashes can arise where strong personalities have differing ideas on how to start the work and on the

direction the project should take. During this time, the efficiency of the work produced is also reduced since the group has not come together fully and that was our experience during this project. Looking back, it is clear that in the beginning we were not always all pulling in the same direction and at times had differing ideas of how to proceed. However, by improving our organization and team communication we were also able to increase the efficiency and teamwork within the group. Overall, our group dynamic was very good. We think it is safe to say that internal strife in the group was never an issue for us but rather lack of communication at the start which we fixed along the way as we became more used to working with one another. An anecdotal example of this is after the last sprint finished and we discussed what documents needed to be made ready for hand-in, everyone knew what we needed to do and set to work without the explicit need for us to write up a detailed division of the remaining work.

A project of this nature and scope introduces a lot of obstacles and at any moment there might be a problem lurking around the corner ready to rear its ugly head around. By having to deal with this in a larger project is a great learning experience and we have all added new tools to our arsenal when it comes to organization of a larger project, and gained new problem-solving skills which we can take with us into future projects.

4 Timekeeping

For a project such as ours there are many different aspects that make up the project. This includes time spent on meetings, planning, learning new tools, coding, and writing documentation. In this chapter we provide an overview to show how much time the team spent on the project in total, and individually, as well as showing how much time was spent on different aspects of the project.

4.1 Breakdown of hours

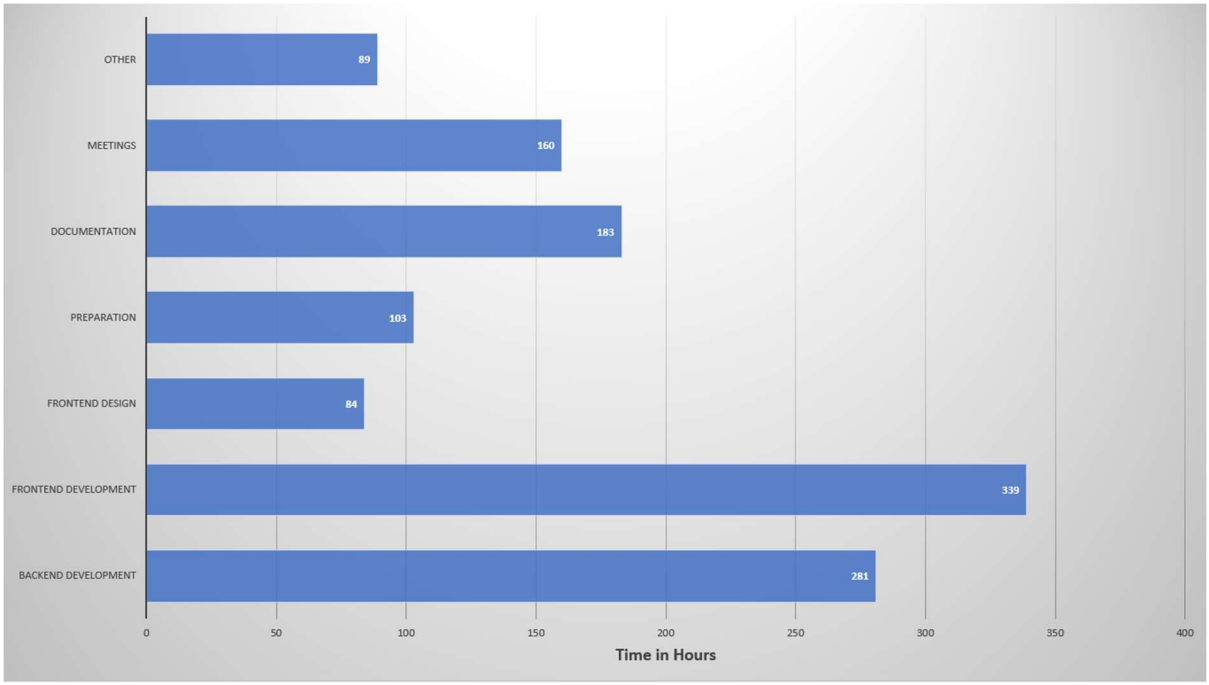

Figure 6 - Worked hours per category

The total amount of worked hours for the whole team was 1239 hours and Figure 6 - Worked hours per category shows a categorized breakdown of the hours.

The total development time for the project was 620 hours, 281 hours for backend development and 339 hours for frontend development. The design process took 84 hours and meetings 160 hours in total. The documentation for the project took 183 hours and that includes time taken to write all hand-in documents i.e., final report, design report, user manual, operating manual, and developer manual. The preparation category shows the time spent

preparing for status meetings, both preparing the presentations and rehearsals. Finally, the other category includes entries that do not belong in one of the other categories, such as lecture time at the beginning of the course and for learning Blazor and other tools that the team had not used before.

4.2 Hours worked per team member

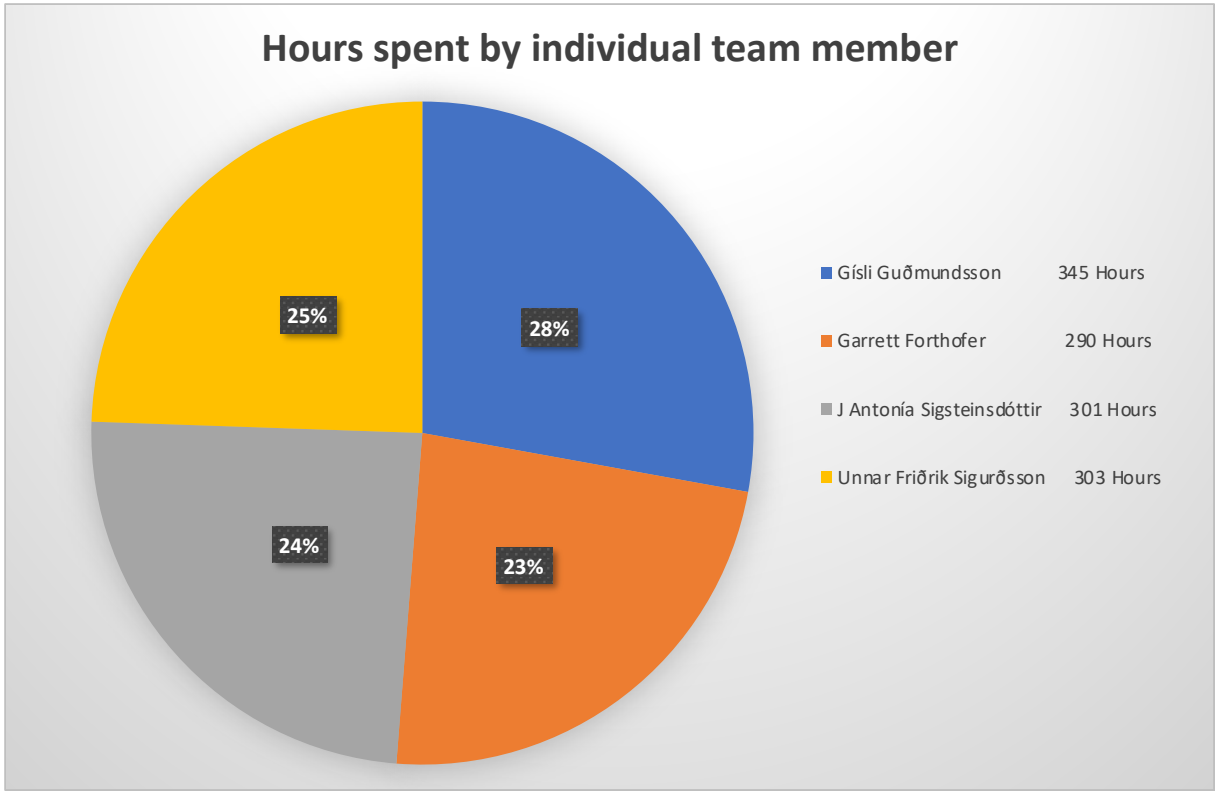

Figure 7 - Hours worked per team member

Figure 7 - Hours worked per team member shows a breakdown of the hours worked by each team member on the project from start to finish.

5 Conclusion

Over the course of the project, we have developed greatly. Both in our ability as a team and in our personal capacity to create professional software services. We learned to work as a team, taking feedback from our advisors and the subjects of our user tests to improve our service and ability to develop it. We responded to the changing circumstances of the project and our lives to plan and execute the development of our service to completion. All this is to say that our team considers the project a success, in our product and what we have learned along the way.

5.1 Future of the project

The project was created for the company UteamUP which has the objective of improving the quality of software for users that work within the maintenance business. The initial project was created with the minimum requirements to be functional such as adding parts, workorders, assets and giving users access to different organizations. After our project is finished as a university project, UteamUP will continue its development by adding features and improving the user experience. First the application will be upgraded from the current version of .NET 5 to .NET 6. Additionally, a subscription system and invoicing system will be added so that companies can purchase different services. The user interface will undergo further development and made accessible and unified on different devices such as mobile, tablet or tailored to a specific application that will be intended for a specific market group.

The application should be accessible and possible to view from anywhere and the ideology of the application should be based on the web 3.0 where the application will be decentralized, and the data AI monitored to maximize the information for the user. Therefore, the data can be accessible to anyone who has access, such as using API calls for 3rd party applications and services.